UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN
TEORÍA DE LA COMPUTACIÓN

OBLIVIOUS DISTRIBUTION, A VARIANT TO THE OBLIVIOUS TRANSFER PROBLEM

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRÍA EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:
KYLE ELBJORN FLORES

TUTOR:
DR. JOSÉ DAVID FLORES PEÑALOZA
FACULTAD DE CIENCIAS, UNAM

CIUDAD UNIVERSITARIA, CIUDAD DE MÉXICO, NOVIEMBRE 2024

Acknowledgments

A mi madre; por tu eterno amor y cariño, por tu dedicación, por tu esfuerzo, y por todo el apoyo incondicional, gracias. Espero que este trabajo sirva como retribución parcial a mi infinita deuda.

A mi padre; por tus enseñanzas, por tu rectitud, por tu humor, y por todos tus sacrificios en vida, gracias. Espero que este trabajo demuestre que tu espíritu vive en mí.

A mi hermana; por tu compañía, por inspirarme, por entenderme, y porque este trabajo no hubiera sido posible sin ti, gracias.

Al Doctor David; por su apoyo, por su paciencia, por su instrucción, por todas las pláticas interesantes, y por lo que nos espera, gracias.

A Miguel; por acompañarme en este proyecto, y por toda la ayuda brindada, gracias.

A la Universidad Nacional Autónoma de México; por brindar las condiciones que permiten una educación de calidad de carácter público, gracias.

Version

Document version: 1.1

Document history

Version	Author	Date	Changes
1.1	Kyle Elbjorn Flores	2024–09–12	Minor corrections
1.0	Kyle Elbjorn Flores	2024–09–11	Initial release

Document repository

The latest version of this document is available at:

GitHub repository: https://github.com/Kyle-Elbjorn/Master-Thesis

Contents

Preface

Secure multi-party computation (MPC) is the collection of cryptographic protocols intended to provide a group of participants with the means to collectively evaluate a function using their personal private data with the explicit intention of preserving the secrecy of their respective inputs without having to depend on a trusted third-party.

This thesis proposes a family of protocols named Oblivious Distribution in which a single sender communicates a set of messages to a group of receivers, keeping the privacy for everyone involved. At the end of the exchange, the sender may not be able to determine which message is held by each receiver, while each receiver may only access a message if it was destined to them.

The present work provides two proofs: the first proof states that the Oblivious Distribution primitive is universal for MPC, i.e., it serves as a construction by which any computable function may be solved in a distributed system; the second proof demonstrates that these protocols are non-tolerant to malicious actors (impossibility).

Lastly, a set of protocol implementation examples is provided for use in a practical setting.

Chapter 1

Introduction

1.1 Background

In modern day society, computer networks are a fundamental tool for everyday activities and, naturally, certain data in the never-ending stream of information is to be kept concealed from prying eyes, while being available to be used by individuals with the clearance to do so.

There are two major labels for the concealment of information: private data is ideally limited to one individual, while secret data is intended to be of common knowledge or—in its defect—properly distributed among members of a group. Thus, the need arises for mechanisms which allow participants in a system to store, exchange and ultimately use this information while keeping the required confidentiality boundaries.

1.1.1 Information worth hiding

Information security is the collection of practices that intend to protect data from adverse occurrences, whether it occurs while storing or communicating it.

These practices can be grouped according to their purpose; the first of these is data integrity, which is concerned with the consistency of information: no unintended modifications may occur by random noise in the medium used nor by malicious third parties. Then, there is data concealment, which is concerned with restricting or permitting access to information: no partial nor complete readability may be allowed to anyone that does not possess the appropriate credentials. Lastly, there is data authentication and non-repudiation, which centers around the proper attribution of information generated: no individual may wrongly claim to have produced any material, while no individual can deny having produced something they have.

To keep information secure during communications, the notion of cryptography came to be; a series of interdisciplinary techniques rooted on mathematical theory.

Cryptography, traditionally, has been built upon long studied problems for which there is a *strong belief* no efficient solution exists. Such is the case with the integer factorization hardness conjecture, which is central to computational number theory [Gol01].

In more recent years, research has shifted towards the fields of computational complexity, abstract algebra, and information theory. The last one being of particular interest for achieving privacy [Gol01] [Gol03].

1.1.2 Secrets worth sharing

Even when information is highly sensitive, the methods used for safeguarding it must not make it unusable. A lock is useless if no key to open it exists, but it is equally useless if it can be opened unwarrantedly regardless of the means utilized to pierce through.

Modern cryptography goes beyond locks and keys; algorithms can be composed into protocols, allowing for complex operations with information while adhering to different high-level security properties.

A protocol designed to operate on private information may provide a solution, however, it may also expose more than is desired. Furthermore, there are functions whose solution can be utilized to calculate a participant's input [BGW88] which may—in some cases—make the whole process counterproductive.

Thus, the main goal for modern cryptographic protocols is to yield results within solid frameworks, allowing users to determine how much they are compromising by partaking in them; cautioned of the risks by means of compelling proof.

1.2 Motivation

This thesis' purpose is to present the Oblivious Distribution primitive—a variant to the renowned Oblivious Transfer approach. Throughout the years, the Oblivious Transfer primitive has been revisited numerous times, and several improvements and optimizations to the protocol proposals have been introduced (e.g., [Bea95] [NP01] [Can+02] [CO15] [Qin15]).

The goal of this work is to provide a new *multi-party* perspective to a conventionally *two-party* strategy, to explore possible reductions for communication costs—i.e., number of messages required—and temporal costs—i.e., number of steps required—in certain scenarios, while preserving the privacy constraints.

Oblivious Distribution can be understood as a generalization to the Oblivious Transfer problem in two dimensions—messages and receivers—inducing a family of protocols for secure function evaluation.

Chapter 2

Literature Review

2.1 Distributed computing model

This section presents the necessary notions that describe the computational model on which the cryptographic protocols presented on this thesis are build upon. These concepts where originally presented in the context of minimum-knowledge (zero-knowledge) interactive proof systems [GHY85] [GMR85], and adapted to allow for privacy-preserving (secure) computation [Yao86] [MR91].

2.1.1 Turing machines

The Turing machine (TM) is a computational model of an abstract mechanism that utilizes an infinite unidimensional tape divided into cells and a tape head. The machine starts only with an input string, leaving the rest of the tape *blank*. Proceeding according to well-defined *transition functions* for the tape head that allow for reading, writing and moving along the tape, cell by cell, in either direction. Halting, if ever, when it has reached a final state that allows to produce a decision to either *accept* or *reject* the input provided [Sip13]. For this thesis, the input alphabet for all TMs is assumed to be the Boolean alphabet $0, 1$.

A multi-tape Turing machine extends the previous model by considering more than one tape per machine, each one with its respective tape head. The *transition functions* are then modified to allow for reading, writing and moving the heads on any number of its respective tapes simultaneously [Sip13]. Even though a multi-tape TMs are equivalent in computational power to single-tape TMs [Sip13], the former are useful to illustrate more complex models.

Formal definitions are omitted for better readability. Proper in depth explanations can be consulted on [Sip13].

Probabilistic Turing machines

The probabilistic Turing machine (PTM) computational model can be expressed as a multi-tape Turing machine which is provided with a read-only tape of random values, denominated *random tape*. The *random tape* is expected to keep a uniform distribution; for a Boolean input alphabet this is analog to a tape produced by a "coin-tossing" device [Gol01].

A probabilistic polynomial-time Turing machine is a PTM that always halts after a polynomial number of steps—in the length of the input [Gol01].

2.1.2 Interactive Turing machines

The interactive Turing machine (ITM) computational model is based upon the probabilistic polynomial-time Turing machine [Yao86], and was originally defined for interactive proof systems [GMR85]. As the name suggests, this model allows representing machines that communicate (interact) among them, to produce any joint computation.

Each ITM possesses [Gol01]:

- A *read-only* input tape.
- A *read-only* random tape.
- A *read-write* work tape.
- A *write-only* output tape.

  For every other machine:

  – A *read-only* communication tape (for receiving messages).

  – A *write-only, non-modifiable* communication tape (for sending messages).

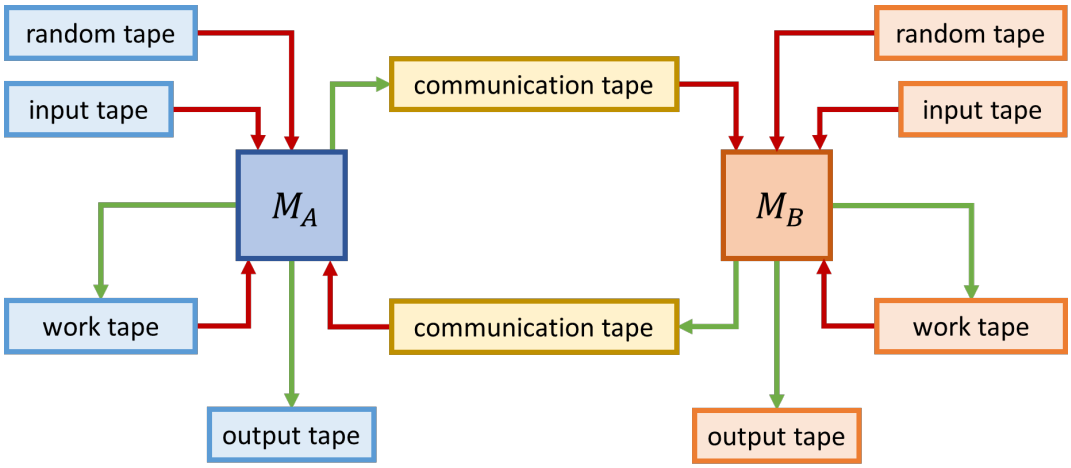

Figure 2.1: Example of two interconnected ITMs (M_A, M_B).
Red arrows denote *reading* capabilities. Green arrows denote *writing* capabilities.

2.1.3 Communication model

The communication tapes between two machines serve as bidirectional *secure* channels (i.e., only machines with an explicit *read-capable* tape head over a communication tape, may access the contents of said tape) [MR91]. The system induced by the interconnectivity among machines is, effectively, a full peer-to-peer network.

The model described (also referred as *game network*), as presented in [GMW87], also considers a *public broadcasting* tape (i.e., each machine has a *write-only* tape that can be *read* by all other machines parallelly) as well as common clock for message synchronization. It is also argued that these elements are not strictly necessary; the notions of security presented in said work are essentially *independent* of the underlying communication model [GMW87] [MR91].

2.1.4 Protocol

A set of n participants, for $n \geq 2$, that desire to communicate in order to jointly compute the result for any function f is denoted as a *distributed system*.

The set of finite steps that the participants have to perform in order to compute the desired function f is denoted as a *protocol*.

A protocol $\mathcal{P}$ in a distributed system of n interactive Turing machines is expressed by the n-tuple: $\mathcal{P} = (M_0, M_1, \ldots, M_{n-1})$. Each machine is given an input to be utilized and a limiting value k. The machines follow the steps prescribed by the protocol, exchanging messages using their one-to-one communication tapes, and performing the necessary local computations each time. All machines halt within a number of steps bounded by some polynomial in k, leaving some string in their respective output tapes [GMR85] [Yao86] [MR91].

2.2 Security in cryptographic protocols

This section examines the notion of security pertaining to cryptographic protocols. By means of describing the behaviors that can be expected of the participants in any form of interaction they may engage in—while participating in a distributed system—and the implications over the information, its accessibility, and its manipulability.

2.2.1 Adversarial behavior

In an ideal execution of a protocol, a participant acts only by following the steps prescribed by the protocol. This behavior is referred to as honest, while any other behavior is considered dishonest. In a realistic execution two mayor dimensions of dishonest behavior are to be expected [GMW87] [Gol03].

Definition 2.1 (Honest participant [GMW87] [Gol03])**.** A participant in a protocol is considered honest if:

- The messages emitted by the participant are produced adhering to the protocol's prescription.

The first dimension of dishonest behavior differentiates whether a dishonest party actively disrupts the execution of the protocol or passively gathers information. Both variants of dishonest participants are considered coalition prone [Gol03].

Definition 2.2 (Semi-honest adversary [Gol03])**.** A participant in a protocol is considered a semi-honest adversary (also called honest-but-curious adversary, passive adversary, or restricted adversary) if:

- The messages emitted by the participant are produced adhering to the protocol's prescription (like an honest participant).
- The participant gathers any amount of data from the communication tapes and shares it with other semi-honest participants.

Definition 2.3 (Malicious adversary [Gol03])**.** A participant in a protocol is considered a malicious adversary (also called Byzantine adversary [Mal19], active adversary, or unrestricted adversary) if:

- The messages emitted by the participant are produced diverting from the protocol's prescription.
- The participant communicates with other malicious parties to coordinate an attack using all the information available to them and any inferable from it.

The second dimension of dishonest behavior considers the capability of a dishonest party to corrupt initially honest participants during the execution of the protocol [Can00a] [Gol03].

Definition 2.4 (Non-adaptive adversaries [Can00a] [Gol03])**.** The set of dishonest participants is fixed, prior to the execution.

It is considered that the set of dishonest parties is not known by the honest participants [Gol03].

Definition 2.5 (Adaptive adversaries [Can00a] [Gol03])**.** Dishonest participants use the partial information gathered to corrupt honest participants throughout the execution.

This thesis only contemplates non-adaptive semi-honest adversaries. Further reasoning is elaborated in section 3.1 and chapter 6.

2.2.2 Types of security

Security in a system is defined by the types of adversarial behavior it can withstand within the established bound of dishonest parties.

Passive security—or semi-honest security—considers a correct execution of a protocol where a set of corrupt parties may collude to *learn* more than intended, in a passive security environment the following statements are true [EKR18]:

- Adversaries are semi-honest and non-adaptive.
- The view of an honest party consists of its private input, its random tape, and its communication tapes (i.e., the collection of messages it sends and receives during an execution).
- The view of a semi-honest adversary consists of the collective private inputs, random tapes, and communication tapes of all semi-honest adversaries.
- An adversary *learns* whatever can be efficiently computed from its view.

Active security, or malicious security, considers that an adversarial coalition generates their messages deviating from the protocol in order to obtain private information or alter the honest participants' outputs, in an active security environment the following statements are true [EKR18]:

- Adversaries are malicious and may be adaptive.
- The malicious coalition share their respective views, reporting any individual or collective action taken, and obtain any information that can be efficiently computed from them, during and after the execution.
- The malicious adversaries may take any action (or inaction) during the execution.

2.2.3 Ideal function evaluation

As stated before, the intention behind multi-party protocols is to allow a set of participants to jointly evaluate a function.

An *ideal function evaluation* is a conceptual scenario which contemplates the existence of an external trusted party, which intakes the respective inputs of all participants and *correctly* evaluates the desired function. The trusted party then delivers the corresponding output for each participant. This process is done while preserving the privacy of all data involved [MR91].

For this thesis, several "ideal protocols" are presented, which illustrate the desired functionality to be replicated by a protocol's implementation. An in depth explanation of this paradigm is provided in section 3.1.1.

2.3 Secure multi-party computation

Secure computation encompasses all methods for the evaluation of functions where the inputs are kept private [EKR18].

The first main type is outsourced computation, where the parties involved send the data to a third party who performs the corresponding operations and returns the results following the specifications provided. In order to preserve privacy, complex encryption schemes—e.g., fully homomorphic encryption—have been proposed [EKR18].

The second type of secure function evaluation is the multi-party computation. For this work, only the latter type is pertinent, although the security model used takes inspiration from the notion of outsourced computation.

Secure multi-party computation (MPC) protocols are understood as follows [Cra15]:

- Let $P_1, \ldots, P_n$ be a set of participants.
- Let each participant P_i hold a private input x_i.
- Let f be a function with n inputs such that $f(x_1, \ldots, x_n) = \{y_1, \ldots, y_n\}$.

A MPC protocol for the function f must satisfy the subsequent conditions:

Correctness. For any set of inputs $\{x_1, \ldots, x_n\}$, with probability 1, the correct values for the output vector $\{y_1, \ldots, y_n\}$ are computed.

Privacy. The only new information a party P_i may obtain from the execution of the MPC protocol is the value y_i.

This construction contemplates an individual output for each participant, but for many applications this condition may be weakened by considering the special case for a single public output y where $y = y_1 = \cdots = y_n$.

2.3.1 Secure two-party computation

The subfamily of MPC protocols where $n = 2$ is a significant case. The denominated secure two-party computation (2PC) protocols often require a custom solution outside the general n-party case [EKR18].

Similar to the MPC consideration, the special case of single public output y, where $y = y_1 = y_2$, is to be considered.

2.3.2 Application examples

Millionaires problem

Problem: A, B are two millionaires who wish to determine which one is wealthier, with the constraint that both want to keep their respective net worth a secret.

This problem is used in the Yao's foundational article [Yao82] to set the notion of secure computation, along other possible applications.

The proposed solutions for this 2PC problem considers the function $f(x_1, x_2)$ as:

$$y = \begin{cases} 1 & \text{if } x_1 < x_2 \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

The article details a protocol model for the general solution, discussing the privacy constraint, and concludes on the generalization to the n-party case.

Secret voting problem

Problem: $P_1, \ldots, P_m$ are a group of participants who wish to partake on a binary consensus $f(x_1, \ldots, x_m) \in \{0, 1\}$, with the constraint that no participant may learn any of its peers' vote.

Listed as an application example in [Yao82]. Given the context, it may be desirable to extend the properties of the voting protocol to include non-malleability (generation of a vote from the input of any number of honest participants) and coercion resistance (inability to prove a third party the content of an emitted vote)—both of which have been proven compatible with MPC [EKR18].

Mental poker problem

Problem: A, B are two parties wishing to play a *fair* game of poker over the phone. The physical cards and the interactions between A and B must be accomplished purely by messages.

This problem was proposed in a report [SRA79], prior to the solidification of the concepts surrounding MPC. The work concludes that the card-dealing problem is insoluble without the assumption of computationally hard problems, as the information available to each player should be enough to determine the opponents hand.

This problem is later revisited in [GMW87], being used as an example for a general solution (completeness theorem) to multi-party protocol problems in distributed systems with an honest majority of players.

Auctions

Problem: $S_1, \ldots, S_m$ is a group of sellers, $B_1, \ldots, B_n$ is a group of bidders. Both groups desire to interact in order to conduct a fair auction where bids are to be kept private.

This application has a large scale real-life example—commonly referred to as the Danish Sugar Beet Auction—which was thoroughly analyzed in [Bog+09].

The MPC protocol implemented allowed not only for the parties to preserve bid privacy per se, but to impede the mischievous generation of a bid from another participant's bid (non-malleability).

Machine Learning

MPC protocols can be used in two different stages of machine learning systems [EKR18]:

Problem (Private inference)**:** $C_1, \ldots, C_n$ is a group of clients. S is a server wishing to keep its model private, while C_i withholds its test inputs. Each client receives only its corresponding prediction.

Problem (Private training)**:** $P_1, \ldots, P_m$ is a group of participants combining their data to train a model, without exposing it.

While the usage of MPC protocols to solve these problems is technically achievable, the consensus is that the scale of complexity in communication makes it unfeasible to operate in real environments. Hybrid proposals have been presented which integrate MPC alongside fully homomorphic encryption schemes [EKR18].

2.4 Oblivious Transfer

The notion of Oblivious Transfer originates from the paper "How to Exchange Secrets with Oblivious Transfer" [Rab05], where it is utilized as a sub-protocol in the Exchange of Secrets (EOS) protocol implementation.

The EOS protocol is a two-party scheme that gives a solution to the problem of information transaction between two non-trustable participants without a trusted third party, or a trusted mechanism for simultaneous message trade [Rab05]. The protocol implementation provided depends on cryptographic assumptions—namely computational hardness for integer factorization (section 5.1.1) and the existence of the one-way function primitive (section 5.1.2).

The Oblivious Transfer implementation proposed was discussed further in [FMR96], where it is argued that no proof of resistance to a dishonest receiver is given; proposing an intermediate step where the receiver *proves* to the sender—within a context of probability—the possession of the underlying information utilized, without revealing the value itself. This allows the sub-protocol to keep a relative robustness.

The idea of Oblivious Transfer developed throughout the years and, what was originally a single protocol implementation, became a family of two-party protocols [Kil88] [Bea95] [NP99], which can be categorized in two subfamilies: the *probabilistic* OT protocols and the *deterministic* OT protocols.

2.4.1 Probabilistic OT protocols

Original OT definition

The original approach to the Oblivious Transfer (OT) problem, defines the protocol as an on-line two-party primitive where a sender S sends a message m to a receiver R such that R has a $\frac{1}{2}$ probability of learning the contents of m; while S has no capability of discerning if R received m or not [Bea95] [Bar07].

The desired functionality is described by the following ideal protocol [Bea95], where the message is minimized to a single bit b. The \$ symbol represents the value is sampled via a random function that returns a Boolean value ($\in \{0, 1\}$) with equal probability.

Ideal protocol 2.1 (Probabilistic Oblivious Transfer)**.**

Setting. Sender S determines its input bit b.

Step 1. S sends b to the trusted third-party $\mathcal{T}$.

Step 2. $\mathcal{T}$ assigns with a $\frac{1}{2}$ probability the bit c.

Step 3. If c is 0 then R receives from $\mathcal{T}$ the tuple (c, c) indicating the message has been lost.

If c is 1 then R receives the tuple (c, b) which explicitly indicates the second value is the message.

$S:$	input b
$S \to \mathcal{T}:$	b
$\mathcal{T}:$	$c \leftarrow \$$
$\mathcal{T} \to R:$	$\begin{cases} (c, c) & \text{if } c \text{ is } 0 \\ (c, b) & \text{if } c \text{ is } 1 \end{cases}$

Figure 2.2: Ideal probabilistic Oblivious Transfer protocol.

In this ideal protocol, it is notable that R learns the input of S if and only if the random variable c equals 1 (which is sampled *randomly*), thus, it is intrinsically probabilistic.

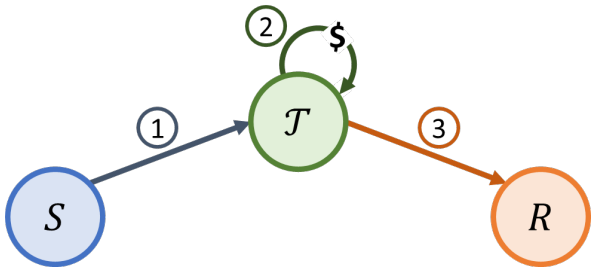

Figure 2.3: Graphical representation for the ideal probabilistic Oblivious Transfer protocol.

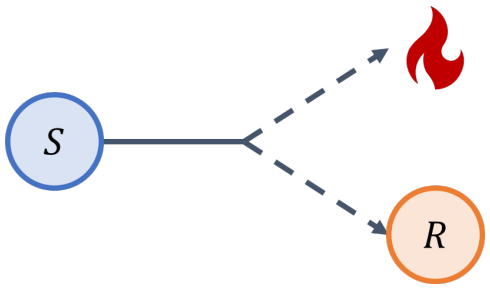

Figure 2.4: Graphical representation for an outsider's view of a probabilistic Oblivious Transfer protocol execution. The dashed arrows represent the two possible *results*, either the message is delivered to R or it is effectively *lost*.

One-out-of-two OT

The one-out-of-two OT protocol ($\frac{1}{2}$OT)—as proposed in [EGL85]—is another approach to the Oblivious Transfer problem where, instead of one message with a 50/50 probability of arriving or being lost in transmission, the sender S establishes two messages from which one—and only one—will be received by the receiver R. Properly, R can not learn any information related to the message not received, while S can not discern which of the messages does R hold at the end of the protocol [BK17]. The protocol remains probabilistic.

Ideal protocol 2.2 (One-out-of-two Oblivious Transfer)**.**

Setting. Sender S determines two input bits b_0, b_1.

Step 1. S sends the tuple (b_0, b_1) to the trusted third-party $\mathcal{T}$.

Step 2. $\mathcal{T}$ assigns with a $\frac{1}{2}$ probability the bit c.

Step 3. If c is 0 then R receives from $\mathcal{T}$ the bit b_0.

If c is 1 then R receives from $\mathcal{T}$ the bit b_1.

$S:$	input b_0, b_1
$S \to \mathcal{T}:$	(b_0, b_1)
$\mathcal{T}:$	$c \leftarrow \$$
$\mathcal{T} \to R:$	b_c

Figure 2.5: Ideal (probabilistic) one-out-of-two Oblivious Transfer protocol.

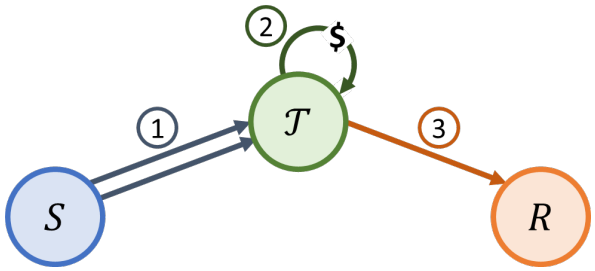

Figure 2.6: Graphical representation for the ideal one-out-of-two Oblivious Transfer protocol.

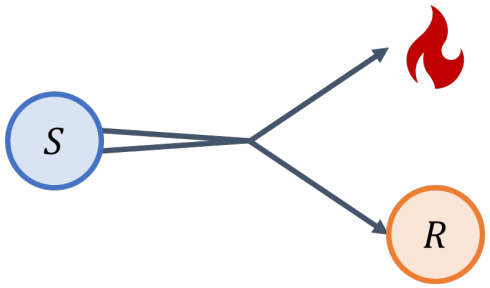

Figure 2.7: Graphical representation for an outsider's view of a one-out-of-two Oblivious Transfer protocol execution. Two messages are sent, one message is delivered to R the other message is effectively *lost*.

2.4.2 Deterministic OT protocols

Chosen one-out-of-two OT

A variant to the $\frac{1}{2}$OT protocol is the chosen one-out-of-two OT ($\binom{2}{1}$OT). Where the receiver R takes an active role by supplying an input bit to select the message that will be received [Bea95]. By removing the random function, it becomes a deterministic protocol.

An implementation example is provided in appendix A.

Ideal protocol 2.3 (Chosen one-out-of-two Oblivious Transfer)**.**

Setting. Sender S determines two input bits b_0, b_1.

Receiver R determines an input bit c.

Step 1. S sends the tuple (b_0, b_1) to the trusted third-party $\mathcal{T}$.

Step 2. R sends its selection bit c to the trusted third-party $\mathcal{T}$.

Step 3. If c is 0 then R receives from $\mathcal{T}$ the bit b_0.

If c is 1 then R receives from $\mathcal{T}$ the bit b_1.

$S:$	input b_0, b_1
$R:$	input c
$S \to \mathcal{T}:$	(b_0, b_1)
$R \to \mathcal{T}:$	c
$\mathcal{T} \to R:$	(c, b_c)

Figure 2.8: Ideal chosen one-out-of-two Oblivious Transfer protocol.

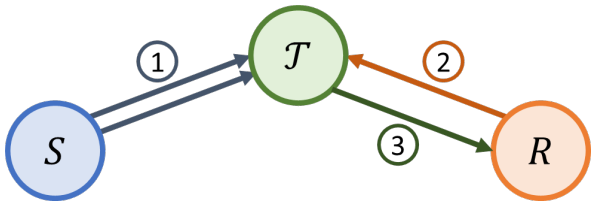

Figure 2.9: Graphical representation for the ideal chosen one-out-of-two Oblivious Transfer protocol.

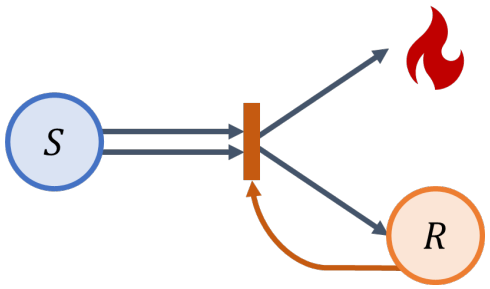

Figure 2.10: Graphical representation for an outsider's view of a chosen one-out-of-two Oblivious Transfer protocol execution. Two messages are sent, R uses its input to determine which message is kept and which message is discarded.

One-out-of-N OT

Based on the previous variant, the tuple of messages can be extended to a vector of N messages $X_1, \ldots, X_N$, this deterministic protocol is named one-out-of-N OT ($\binom{N}{1}$OT). The receiver R determines an input I, $1 \leq I \leq N$, the message received will be the one in that position of the message vector X_I [NP99].

Ideal protocol 2.4 (One-out-of-N Oblivious Transfer)**.**

Setting. Sender S determines an input vector $X_1, \ldots, X_N$.

Receiver R determines an input natural number $I \in [1, N]$.

Step 1. S sends the vector $X_1, \ldots, X_N$ to the trusted third-party $\mathcal{T}$.

Step 2. R sends its selection number I to the trusted third-party $\mathcal{T}$.

Step 3. According to the value of I, R receives from $\mathcal{T}$ the message X_I.

$S:$	input $X_1, \ldots, X_N$
$R:$	input I
$S \to \mathcal{T}:$	$X_1, \ldots, X_N$
$R \to \mathcal{T}:$	I
$\mathcal{T} \to R:$	(I, X_I)

Figure 2.11: Ideal one-out-of-N Oblivious Transfer protocol.

k-out-of-N OT

The $\binom{N}{1}$OT would require a sequence of executions in order to communicate multiple messages from the input vector $X_1, \ldots, X_N$. To reduce the number of communication rounds the k-out-of-N OT ($\binom{N}{k}$OT) protocol was designed [NP99] [Hsu+18]. To achieve this, R determines an input $\{\sigma_1, \ldots, \sigma_k\} \subset \{1, \ldots, N\}$ [Qin15].

Ideal protocol 2.5 (k-out-of-N Oblivious Transfer)**.**

Setting. Sender S determines an input vector $X_1, \ldots, X_N$.

Receiver R determines an input vector $\{\sigma_1, \ldots, \sigma_k\} \subset \{1, \ldots, N\}$.

Step 1. S sends the vector $X_1, \ldots, X_N$ to the trusted third-party $\mathcal{T}$.

Step 2. R sends its selection number $\{\sigma_1, \ldots, \sigma_k\}$ to the trusted third-party $\mathcal{T}$.

Step 3. According to the values $\{\sigma_1, \ldots, \sigma_k\}$, R receives from $\mathcal{T}$ the collection of messages $X_{\sigma_1}, \ldots, X_{\sigma_k}$.

$S:$	input $X_1, \ldots, X_N$
$R:$	input $\{\sigma_1, \ldots, \sigma_k\}$
$S \to \mathcal{T}:$	$X_1, \ldots, X_N$
$R \to \mathcal{T}:$	$\{\sigma_1, \ldots, \sigma_k\}$
$\mathcal{T} \to R:$	$X_{\sigma_1}, \ldots, X_{\sigma_k}$

Figure 2.12: Ideal k-out-of-N Oblivious Transfer protocol.

2.5 Garbled Circuits

Yao's Garbled Circuits (GCs), also known as scrambled circuits, is a passively-secure two-party computation primitive that was originally implied in the extended abstract of "How to generate and exchange secrets" [Yao86] and later developed during oral presentations related to the paper [Kap00].

GCs were constructed as a framework to extend the notion of OT; hence the problem is also known as oblivious circuit evaluation, which was devised as a complete solution for 2PC [Kil88].

Intuitively, the main idea is to reduce the evaluation of the function to the direct addressing in a look-up table—benefiting from the fact that a function with a small enough domain allows enumerating all possible inputs efficiently [EKR18].

Formally, given a function $\mathcal{F}(x, y)$ where A's input is $x \in X$ and B's input is $y \in Y$, then $\mathcal{F}$ can be represented as a look-up table T, consisting of $|X| \cdot |Y|$ rows, $T_{x,y} = \langle \mathcal{F}(x, y) \rangle$ [EKR18].

2.5.1 Protocol

The two parties A, holding an input x, and B, holding an input y, interact in order to evaluate a previously agreed upon function $\mathcal{F}(x, y)$, for which there is a Boolean circuit of common knowledge that computes the value [Gol03].

A thorough exploration of the protocol is provided in appendix B.

Protocol 2.1 (Garbled Circuit protocol [BK17] [CB19])**.**

Step 1. A generates the Boolean circuit C for the functionality $\mathcal{F}$.

Step 2. A transforms C into a garbled circuit C_G:

- For each gate $g_i \in C$, A calculates the respective truth table T_{g_i}.
- Every row in T_{g_i}'s output column is encrypted using a pair of one-time keys, according to the respective input for the row.
- The order of the rows is randomized, generating a *garbled* gate.

Step 3. A sends C_G (the collection of *garbled* gates), along with the keys corresponding to its inputs for each gate, to B.

Step 4. B learns the keys corresponding to its own inputs, for each gate, by means of various chosen 1-out-of-2 oblivious transfer protocol executions.

Step 5. B evaluates C_G by decrypting each exit wire, thus, obtaining the final output.

2.5.2 Universality

As corollary to the proof presented in [Kar72] for the Cook-Levin theorem, it is concluded that for any Turing Machine there exists a family of Boolean circuits parameterized by the length of their inputs which is equivalent, i.e., computes the same output value.

The Garbled Circuit primitive is an implementation that allows for the construction of Boolean circuits gate by gate, thus, providing a framework for implementing any Turing computable functionality.

Therefore, the Garbled Circuit primitive is proven universal for MPC.

2.6 Secret sharing

Secure secret-sharing schemes—or linear threshold schemes—are a general solution for MPC.

Two different approaches were simultaneously, but independently, developed in [Bla79] and [Sha79]. In principle, these schemes are defined as follows [HDS12]:

Definition 2.6 ((k, n) secret-sharing schemes)**.**

For $k \leq n$, a (k, n) secret-sharing scheme is a construction for distributing any secret information S among a group of n participants (*shareholders*), $P_0, \ldots, P_{n-1}$, by generating n fragments of information (*shares*), $S_0, \ldots, S_{n-1}$, such that:

- A shareholder P_i holds exactly one share, namely S_i.
- Any set of, at least, k shareholders can *combine* their respective shares to easily compute S.
- Any set of less than k shares cannot be used to reconstruct S, or learn any valuable information from S.

2.6.1 Blakley's scheme

Definition 2.7 (Blakley's secret-sharing scheme [HDS12])**.**

Solution based on hyperplane geometry; the secret is embodied as a point in a k-dimensional space.

The n shares are constructed from a set of n hyperplanes intersecting at the point.

Each share corresponds to a coefficient of a different hyperplane.

A secret and its shares can be represented as a linear system $Cx = y$, where the matrix C and the vector y are obtained from the hyperplane equations.

2.6.2 Shamir's scheme

Definition 2.8 (Shamir's secret-sharing scheme [HDS12])**.**

Solution based on polynomial interpolation; a $(k-1)$ degree polynomial which uses the secret as the constant term a_0 while the remaining coefficients $a_1, \ldots, a_{k-1}$ are randomly generated.

The secret is reconstructed via Lagrange's interpolation.

The share computation can be represented in linear algebra by using a Vandermonde matrix.

Chapter 3

Cryptographic background

3.1 Security analysis

The two security properties for MPC formerly established in section 2.3—i.e., correctness and privacy—are satisfied by a protocol implementation if it meets the following conditions [Cra15]:

Condition 3.1 (Perfect correctness)**.**

With probability 1, all participants receive outputs that are correct based on the inputs supplied.

Condition 3.2 (Perfect t-privacy)**.**

For any subset C of corrupt participants, of size at most $t < \frac{n}{2}$, $P_i \in C$ learns no information beyond $\{x_j, y_j\}_{P_j \in C}$ by participating in the protocol's execution, regardless of its computing power.

More precisely, the privacy condition dictates that a participant $P_i \in C$, $C \subset P_1, \ldots, P_n$, learns only the collection of inputs and outputs of all other corrupt participants, as long as the cardinality of C is less than half of all participants.

But these conditions are a theoretical perfect standard that cannot be met by all protocols. For instance, certain functions may reveal the input of another participant independently of the protocol utilized to compute the result (i.e., there are functions for which there are no $\frac{n}{2}$-private protocols [BGW88]), e.g., two participants jointly computing the logical disjunction of their inputs ($x_1 \vee x_2 = y$) may trivially learn the other participant's input once they obtain y if their input was 0.

The condition of correctness is valued above else, in order to prevent unintended behavior that may lead to vulnerabilities or failures; as well as to ensure functional integrity that would otherwise render a protocol unreliable. If this condition is not met, the protocol is unusable in real-life scenarios.

Consequently, a robust approximation is provided to garner all protocols that keep privacy within the capabilities of the execution itself, not considering the information leak that may occur as a side effect of the function solved.

Condition 3.3 (Function-independent privacy)**.**

For any subset C of corrupt participants, of size at most $t < \frac{n}{2}$, $P_i \in C$ learns no information beyond $\{x_j, y_j\}_{P_j \in C}$ and whatever can be efficiently computed from $\{x_j, y_j\}$ by participating in the protocol's execution, regardless of its computing power.

3.1.1 Real-ideal paradigm

The real-ideal paradigm—also referred to as the trusted party paradigm—establishes a comprehensive and measurable prediction method for the flow of information during the execution of a particular cryptographic protocol in the presence of semi-honest adversaries. This approach is first described in the MPC foundational work "How to play ANY mental game" [GMW87].

This paradigm aims to contrast what would be a real-life scenario of a protocol execution with the expected functionality of said protocol.

A functionality is the ideal setting for the secure computation of a function by multiple parties. It is described by a black box construction, in terms of inputs and outputs for each party. This operation may be equated to a trusted third party that outsources the respective processes and procures all security guarantees [EKR18]:

Definition 3.1 (Ideal setting)**.**

Let $P_1, \ldots, P_n$ be a set of participants.

Let $\mathcal{T}$ be a trusted third party.

Let each participant P_i hold a private input x_i.

Each participant sends its input to $\mathcal{T}$.

$\mathcal{T}$ computes the desired functionality $(y_1, \ldots, y_n) = \mathcal{F}(x_1, \ldots, x_n)$.

$\mathcal{T}$ sends the corresponding output y_i to each participant P_i.

The previous model considers a potential adversarial behavior from any participant $P_1, \ldots, P_n$, but not from $\mathcal{T}$. In contrast, an actual interaction requires the execution of a well-defined protocol that achieves the same results [EKR18]:

Definition 3.2 (Real setting)**.**

Let $P_1, \ldots, P_n$ be a set of participants.

Let ψ be a protocol for computing $\mathcal{F}$.

ψ prescribes for each party P_i a collection of functions $\{\psi_{i,0}, \psi_{i,1}, \ldots\}$.

$\psi_{i,j}$ takes as input a security parameter κ, P_i's private input x_i, a random tape, and P_i's communication tapes.

$\psi_{i,j}$ outputs either a message and its addressee, or a termination instruction with a specific value.

A real setting protocol ψ is considered κ-secure if any effect achievable by an adversary in the real setting can be replicated by an adversary in the ideal setting. For this real-ideal models equivalence, the set of computational assumptions must be provided [EKR18].

3.1.2 Statistical indistinguishability

A family of random variables is a function X with its domain in the non-negative integers and range in random variables [Cra15]:

Definition 3.3 (Family of random variables)**.**

$\forall \kappa \in \mathbb{N}$, $X(\kappa)$ is a random variable.

$$X = \{X(\kappa)\}_{\kappa \in \mathbb{N}} \tag{3.1}$$

Two families of random variables X_0, X_1 are considered statistically indistinguishable if the statistical distance between $X_0(\kappa)$ $X_1(\kappa)$ is a negligible function [Cra15]:

Definition 3.4 (Negligible function)**.**

A function $\delta : \mathbb{N} \to [0,1]$ is considered negligible in κ if for all $c \in \mathbb{N}$ there exists $\kappa_c \in \mathbb{N}$ such that $\delta(\kappa) \leq \kappa^{-c}$ for all $\kappa \geq \kappa_c$.

Definition 3.5 (Statistical indistinguishability)**.**

Two families of random variables, X_0, X_1, are statistically indistinguishable ($X_0 \overset{\text{stat}}{\equiv} X_1$) if $\delta(X_0(\kappa), X_1(\kappa))$ is negligible in κ.

3.1.3 Passively-secure protocols

To provide proof that a protocol is resistant to semi-honest adversaries in the real-ideal paradigm, an ideal-setting adversary must be capable of simulating the real-setting adversary's view. This view must be constructed purely from the exchanges made with $\mathcal{T}$ [EKR18].

Formally, the following random variable distributions are defined [EKR18]:

Definition 3.6 (Real-ideal paradigm distributions)**.**

Let π be the protocol to be executed.

Let $\mathcal{F}$ be the functionality that characterizes π.

Let $P_1, \ldots, P_n$ be the set participants.

Let $C \subset P_1, \ldots, P_n$ be the set of adversarial participants.

Let κ be a security parameter.

Let Sim be a probabilistic expected-polynomial-time simulator algorithm.

Let $x_1, \ldots, x_n$ be a set of inputs where x_i denotes P_i's private input.

Let $y_1, \ldots, y_n$ be a set of outputs where y_i denotes P_i's output.

Let $V_1, \ldots, V_n$ be a set of views where V_i denotes P_i's view of the protocol after execution.

The real-ideal paradigm distributions are respectively:

- $\text{Real}_\psi(\kappa, C; x_1, \ldots, x_n)$: On an execution of ψ with security parameter κ,

  Output $\{V_i \mid i \in C\}, (y_1, \ldots, y_n)$.
- $\text{Ideal}_{\mathcal{F},Sim}(\kappa, C; x_1, \ldots, x_n)$: Computing $(y_1, \ldots, y_n) \leftarrow \mathcal{F}(x_1, \ldots, x_n)$,

  Output $Sim(C, \{(x_i, y_i) \mid i \in C\}), (y_1, \ldots, y_n)$.

Given these distributions—expressed as families of random variables—a passively-secure protocol is defined as [EKR18]:

Definition 3.7 (Passively-secure protocol)**.** A protocol π securely realizes $\mathcal{F}$ in presence of semi-honest adversaries if there exists a simulator algorithm Sim such that for every subset of corrupt parties $C \subset P_1, \ldots, P_n$ and all inputs $x_1, \ldots, x_n$ the real-ideal distributions are indistinguishable:

$$\text{Real}_\pi(\kappa, C; x_1, \ldots, x_n) \overset{\text{stat}}{\equiv}_\kappa \text{Ideal}_{\mathcal{F},Sim}(\kappa, C; x_1, \ldots, x_n) \tag{3.2}$$

The previous definition is consistent ([BGW88], [CK89]) with the ensuing condition:

Condition 3.4 (t-privacy)**.**

A protocol is t-private if for any subset of corrupt participants $C \subset P_1, \ldots, P_n$, of size at most t, sharing all the information available to them; a corrupt participant $P_i \in C$ learns no information beyond $\{x_j, y_j\}_{P_j \in C}$ and whatever information can be computed in expected polynomial time from the values $\{x_j, y_j\}$ by participating in the protocol's execution.

Chapter 4

Results

The Oblivious Transfer primitive is devised as a two-party protocol that allows for a universal solution for MPC, where any distributed function among n parties may be evaluated by a well-defined series of one-to-one private transfers.

This chapter presents the conceptual variant denominated Oblivious Distribution (OD), that branches from the idea of OT—where a sender may interact with n participants at a time—in order to establish tools that help solve various scenarios of secure transmission of information.

Three settings are introduced. Firstly, the case where the number of messages is equal to the number of receivers, and it is desired that each receiver gets only one message, while each message is delivered to one receiver. Namely, the messages are delivered *bijectively* to the receivers.

Secondly, the generalization for the first case, considering that the number of messages may be less than the number of receivers.

Lastly, the case where there is any number of messages, but each of message is independently delivered to *exactly one* receiver, uniformly selected from the n receivers. This way, a receiver may not get any message, or may get all of them.

The ideal protocols provided follow the real-ideal paradigm—discussed in section 3.1.1—by describing the interaction with a trusted party that delineates the appropriate procedure for the implementations of the protocols. The table-like diagram representations used are based on [Bea95].

Along with the previous scenarios, three different arguments are explored: an argument of reducibility among variants, an argument of universality for MPC, and an argument of impossibility in presence of malicious parties; this allows for a throughout examination of the relations, capabilities and limits for the OD family.

Side result

As a side result of the research, an alternative Garbled Circuit (GC) message reduction technique was developed. This approach is thoroughly revised in section 4.9 and a derived observation (section 4.9.2) is employed to minimize the communication cost for certain instances of the 3-party Garbled Circuit (3GC) framework.

4.1 Summary of concepts

The following concepts are to be considered for the presented results:

- The interactive Turing Machines computational model utilized consists of interconnected probabilistic polynomial-time Turing machines, a full peer-to-peer network with bidirectional secure communication channels (see section 2.1.2 and section 2.1.3).

- The *ideal protocols* provided below, utilize a *trusted* third party to describe the functionality to be emulated by the implementations. These constructions are based on the real-ideal paradigm (see section 3.1.1).

- The expected adversaries are non-adaptive and semi-honest (see section 2.2.1), thus, the protocols described are passively-secure (see section 3.1.3).

4.2 Oblivious Distribution definition

The high-level definition for the Oblivious Distribution primitive is as follows:

Definition 4.1 (Oblivious Distribution)**.**

Let R be a group of n participants.

Let M be a set of k messages.

Let S be a participant, $S \notin R$, that wants to distribute M to R, while maintaining privacy.

This definition encompasses three noteworthy variant subfamilies:

- The *one-for-each* subgroup includes all protocols for which the number of messages is strictly equal to that of receivers, i.e., $k = n$.

  The purpose of these protocols is to allow a sender to privately provide each receiver with one—and only one—unique message.

  This can be exemplified by a *secret Santa* gift exchange:

  > Each person in a group writes its name in a piece of paper, folds it, and deposits it in a bowl.
  >
  > One by one, the participants remove a random piece of paper from the bowl, the name each obtained is the person to whom they must give a gift. The individual name assignation must remain unknown to all other participants.
  >
  > In this scenario, a participant may draw its own name from the bowl. But sometimes one needs but an excuse to treat oneself.

- The *n-choose-k* subgroup generalizes the *one-for-each* subgroup.

  The number of messages is less than or equal to that of receivers, i.e., $k \leq n$. Thus, a subset of k participants will receive each one a unique message.

  This can be exemplified by an *office party* tombola raffle:

  > In order to boost employee morale, a set of household appliances will be given away to a lucky few at the New Year's office party.
  >
  > All employees' names are inserted into a tombola. After a few spins, a name is drawn to select a winner. Its name is effectively disposed, as no employee may receive more than one prize.

- The *subset* subgroup includes all protocols for which the number of messages varies.

  The purpose of these protocols is to allow a sender to privately dispense the k messages among n receivers; for each message m_i, all receivers have probability $\frac{1}{n}$ of receiving m_i.

  Inherently, this signifies that a receiver may obtain any amount of messages in the interval $[0, k]$.

  This can be exemplified by a *golden ticket* random invitation:

  > An eccentric chocolatier hides five golden tickets among thousands of chocolate bars in order to invite five random lucky children to his chocolate factory.
  >
  > Ideally, any identification trait or the chocolate distribution itself is to be kept private—to deter any potential hijackers.

4.3 One-for-each Oblivious Distribution

This section explores the *one-for-each* subfamily.

For naming convention, the members of this family are denoted by the acronym "OD".

4.3.1 n-on-n Oblivious Distribution

Definition 4.2 ((n, n)OD)**.**

Let R be a set of n receivers.

Let M be a set of n messages.

Let S be a sender, $S \notin R$, that wants to distribute M to R fulfilling the following conditions.

For any pair of indexes $i, j \in [0, n-1] \mid i \neq j$:

1. Each receiver R_i obtains one unique message in M.
2. S must not be able to determine, by means of the protocol:
   a. The identifier of the message received by R_i.
   b. The contents (partial or complete) of the message received by R_i.
3. R_i must not be able to determine, by means of the protocol:
   a. The identifier of the message received by R_j.
   b. The contents (partial or complete) of the message received by R_j.

This can be categorized in two scenarios; depending on whether the receivers have a passive behavior (non-interactive) or an active behavior (interactive) on determining the destination of the messages.

Ideal protocol 4.1 (Non-interactive (n, n)OD)**.**

Setting. Sender S determines an input message vector M.

Step 1. S sends M to the trusted third-party $\mathcal{T}$.

Step 2. $\mathcal{T}$ samples a uniformly distributed permutation π of the integer range $[0, n-1]$.

Step 3. $\mathcal{T}$ delivers each message m_i to the corresponding receiver $R_{\pi(i)}$.

$S:$	input $M : \{m_0, \ldots, m_{n-1}\}$
$S \to \mathcal{T}:$	M
$\mathcal{T}:$	$\pi \leftarrow \$$
$\mathcal{T} \to R_{\pi(i)}:$	m_i

Figure 4.1: Ideal non-interactive (n, n)OD protocol.

Ideal protocol 4.2 (Interactive (n, n)OD)**.**

Let f be a function such that $f : x_1 \times \cdots \times x_n \to \pi$ where $x_i \in [0, n-1]$ and π be a permutation for the sequence $\{0, 1, \ldots, n-1\}$.

Setting. Sender S determines an input message vector M.

Each receiver R_i determines an input x_i.

Step 1. S sends M to $\mathcal{T}$.

Step 2. R_i sends x_i to $\mathcal{T}$.

Step 3. $\mathcal{T}$ computes the permutation $\pi = f(x_0, \ldots, x_{n-1})$.

Step 4. $\mathcal{T}$ delivers each message m_i to the corresponding receiver $R_{\pi(i)}$.

S :	input $M : \{m_0, \ldots, m_{n-1}\}$
R_i :	input x_i
$S \to \mathcal{T}$:	M
$R_i \to \mathcal{T}$:	x_i
$\mathcal{T}$:	$\pi \leftarrow f(x_0, \ldots, x_{n-1})$
$\mathcal{T} \to R_{\pi(i)}$:	m_i

Figure 4.2: Ideal interactive (n, n)OD protocol.

Remark 1. In the interactive protocol, f is a deterministic function. An unbiased permutation, akin to the non-interactive version, would require that each receiver samples its input x_i with uniform distribution from $[0, n-1]$ and f be an algorithm analogue to the Fisher-Yates shuffle (reference recommended for understanding the Fisher-Yates algorithm is available at [Ebe16]).

Remark 2. Since π induces a bijective function between messages and receivers, it is interchangeable to which of them the permutation applies.

4.3.2 Interactive 2-on-2 Oblivious Distribution

Of special interest is the case where $n = 2$, where the protocol becomes a three-party interaction. This scenario, being the case for the least amount of participants, is the basis for the arguments later discussed.

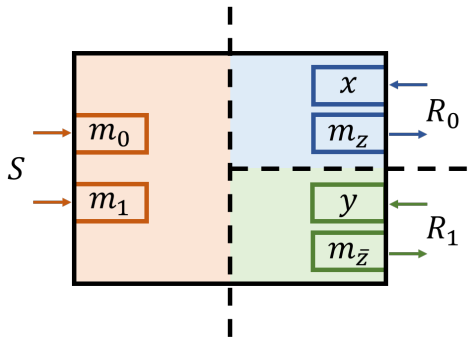

Figure 4.3: Black box representation of the interactive $(2, 2)$OD protocol.

The interactive $(2, 2)$OD protocol (figure 4.3) operates by calculating a bit z which determines what message is going to be obtained by each receiver.

Definition 4.3. Interactive $[(2,2)\text{OD}]$

Let R_0, R_1 be two receivers.

Let $\{m_0, m_1\}$ be two messages.

Let S be a third party sender that wants to distribute $\{m_0, m_1\}$ to R_0, R_1 such that:

1. R_0 receives either m_0 or m_1. R_1 receives the message that R_0 did not receive.
2. S must not learn the value of z or $\overline{z}$ by means of the protocol.
3. S must not learn, by means of the protocol, the contents (partial or complete) of the message obtained by either receiver.
4. R_0 and R_1 must not learn the value of z or $\overline{z}$ by means of the protocol.
5. R_0 must not learn, by means of the protocol, the contents (partial or complete) of the message received by R_1 and vice versa.

Ideal protocol 4.3 (Interactive $(2,2)$OD)**.**

Let f be any 2-input Boolean function (logic gate).

Setting. Sender S determines a pair of messages $\{m_0, m_1\}$.

Receiver R_0 determines an input bit x.

Receiver R_1 determines an input bit y.

Step 1. S sends $\{m_0, m_1\}$ to $\mathcal{T}$.

Step 2. R_0 sends x to $\mathcal{T}$.

Step 3. R_1 sends y to $\mathcal{T}$.

Step 4. $\mathcal{T}$ computes z by computing $f(x, y)$.

Step 5. $\mathcal{T}$ sends m_z to R_0.

Step 6. $\mathcal{T}$ sends $m_{\overline{z}}$ to R_1.

$S:$	input m_0, m_1
$R_0:$	input $x \in \{0,1\}$
$R_1:$	input $y \in \{0,1\}$
$S \to \mathcal{T}:$	$\{m_0, m_1\}$
$R_0 \to \mathcal{T}:$	x
$R_1 \to \mathcal{T}:$	y
$\mathcal{T}:$	$z \leftarrow f(x, y)$
$\mathcal{T} \to R_0:$	m_z
$\mathcal{T} \to R_1:$	$m_{\overline{z}}$

Figure 4.4: Ideal interactive $(2,2)$OD protocol.

4.4 n-choose-k Oblivious Distribution

This section explores the *n-choose-k* subfamily, which generalizes the *one-for-each* subfamily. For naming convention, the members of this family are denoted by the acronym "COD".

Definition 4.4 ($\binom{n}{k}$COD)**.**

Let R be a set of n receivers.

Let M be a set of k messages, $k \leq n$.

Let S be a sender, $S \notin R$, that wants to distribute M to R fulfilling the following conditions.

For any pair of indexes $i, j \in [0, n-1] \mid i \neq j$:

1. A subset of k receivers obtain one unique message in M.
2. S must not be able to determine, by means of the protocol:
   a. If R_i received a message.
   b. The identifier of the message received by R_i (if received).
   c. The contents (partial or complete) of the message received by R_i (if received).
3. R_i must not be able to determine, by means of the protocol:
   a. If R_j received a message.
   b. The identifier of the message received by R_j (if received).
   c. The contents (partial or complete) of the message received by R_j (if received).

Ideal protocol 4.4 (Non-interactive $\binom{n}{k}$COD)**.**

Setting. Sender S determines an input message vector M.

Step 1. S sends M to the trusted third-party $\mathcal{T}$.

Step 2. T samples a uniformly distributed permutation π of the integer range $[0, n-1]$.

- For each message m_i in M:
  a. $\mathcal{T}$ sends the message m_i to the corresponding receiver $\mathcal{R}_{\pi(i)}$.

S :	input $M : \{m_0, \ldots, m_{k-1}\}$
$S \rightarrow \mathcal{T}$:	M
$\mathcal{T}$:	$\pi \leftarrow \$$
for $i \in [0, k-1]$ **do**	
$\mathcal{T} \rightarrow R_{\pi(i)} : m_i$	
end for	

Figure 4.5: Ideal non-interactive $\binom{n}{k}$COD protocol.

Ideal protocol 4.5 (Interactive $\binom{n}{k}$COD)**.**

Let f be a function such that $f : x_1 \times \cdots \times x_n \to \pi$ where $x_i \in [0, n-1]$ and π be a permutation for the sequence $\{0, 1, \ldots, n-1\}$.

Setting. Sender S determines an input message vector M.

Each receiver R_i determines an input x_i.

Step 1. S sends M to $\mathcal{T}$.

Step 2. R_i sends x_i to $\mathcal{T}$.

Step 3. T computes $\pi = f(x_0, \ldots, x_{n-1})$.

- For each message m_j in M:

  a. $\mathcal{T}$ sends the message m_j to the corresponding receiver $\mathcal{R}_{\pi(j)}$.

S : input $M : \{m_0, \ldots, m_{k-1}\}$
R_i : input x_i

$S \to \mathcal{T}$: M
$R_i \to \mathcal{T}$: x_i
$\mathcal{T}$: $\pi \leftarrow f(x_0, \ldots, x_{n-1})$
for $j \in [0, k-1]$ **do**
  $\mathcal{T} \to R_{\pi(j)} : m_j$
end for

Figure 4.6: Ideal interactive $\binom{n}{k}$COD protocol.

4.5 Oblivious Subset Distribution

This section explores the *subset* subfamily.

For naming convention, the members of this family are denoted by the acronym "OSD".

4.5.1 k-on-n Oblivious Subset Distribution

Definition 4.5 ((k, n) OSD)**.**

Let R be a set of n participants (receivers).

Let M be a set of k messages.

Let S be a third party that wants to distribute M to R fulfilling the following conditions.

For any pair of indexes $i, j \in [0, n-1] \mid i \neq j$:

1. Each participant R_i finishes with an output set O_i of messages in M.
2. The sets of messages O_i and O_j are disjoint (i.e., every message is received by one, and only one, receiver).
3. S must not be able to determine, by means of the protocol:
   (a) The cardinality of O_i.
   (b) The identifiers of the messages in O_i.
   (c) The contents (partial or complete) of any single message in O_i
4. R_i must not be able to determine, by means of the protocol:
   (a) The cardinality of O_j.
   (b) The identifiers of the messages in O_j.
   (c) The contents (partial or complete) of any single message in O_j

Remark 3. The sets are not directly constructed by S; they are an emergent result from the distribution of the k messages.

Ideal protocol 4.6 (Non-interactive (k, n)OSD)**.**

Setting. Sender S determines an input message vector M.

Step 1. S sends M to the trusted third-party $\mathcal{T}$.

- For each message m_i in M:
  a. $\mathcal{T}$ samples a random number $r \in [0, n-1]$, uniformly distributed.
  b. $\mathcal{T}$ sends the message m_i to the corresponding receiver R_r.

S : input $M : \{m_0, \ldots, m_{k-1}\}$

$S \to \mathcal{T}$: M
for $i \in [0, k-1]$ **do**
  $\mathcal{T} : r \leftarrow \$$
  $\mathcal{T} \to R_r : m_i$
end for

Figure 4.7: Ideal non-interactive (k, n)OSD protocol.

Ideal protocol 4.7 (Interactive (k, n)OSD)**.**

Let f be a function such that $f : x_1 \times \cdots \times x_n \to r$ where $x_i \in [0, n-1]$ and r be an integer in $[0, n-1]$.

Setting. Sender S determines an input message vector M.

Each receiver R_i determines an input vector X_i.

Step 1. S sends M to $\mathcal{T}$.

Step 2. R_i sends X_i to $\mathcal{T}$.

Step 3. T computes $\pi = f(x_0, \ldots, x_{n-1})$.

- For each message m_j in M:
  a. $\mathcal{T}$ computes $r = f(x_{0,j}, \ldots, x_{n-1,j})$.
  b. $\mathcal{T}$ sends the message m_j to the corresponding receiver R_r.

S : input $M : \{m_0, \ldots, m_{k-1}\}$
R_i : input $X_i : \{x_{i,0}, \ldots, x_{i,k-1}\}$

$S \to \mathcal{T}$: M
$R_i \to \mathcal{T}$: X_i
for $j \in [0, k-1]$ **do**
  $\mathcal{T} : r \leftarrow f(x_{0,j}, \ldots, x_{n-1,j})$
  $\mathcal{T} \to R_r : m_j$
end for

Figure 4.8: Ideal interactive (k, n)OSD protocol.

4.6 Intervariant reductions

In this section, it is proved that the COD subfamily and the OSD subfamily are reducible to *one-for-each* OD instances by using the notion of *garbage* messages— i.e., messages that do not contain any valuable information and can be trivially recognized as such.

An interactive protocol may only be reduced to the respective interactive version of the target protocol. Similarly, non-interactive protocols only reduce to other non-interactive protocols.

4.6.1 Reduction chain

Theorem 4.1.

A (n, n)OD (*one-for-each* subfamily) protocol can be used to implement any Oblivious Distribution protocol, belonging to any subfamily.

Proof.

In order to prove theorem 4.1, the following reduction chain is proven (from right to left), via a corollary of lemma 4.1 and lemma 4.2.

$$(k, n)OSD \leq (1, n)OSD = \binom{n}{1}\text{COD} \leq \binom{n}{k}\text{COD} \leq \binom{n}{n}\text{COD} = (n, n)\text{OD} \tag{4.1}$$

The following observations related to the previous chain are provided:

Observation 1. The (n, n)OD case is functionally equivalent to the $\binom{n}{n}$COD case.

Observation 2. The $\binom{n}{1}$COD is a particular case of the $\binom{n}{k}$COD subfamily.

Observation 3. The $\binom{n}{1}$COD case is functionally equivalent to the $(1, n)$OSD case.

□

Lemma 4.1.

The $\binom{n}{k}$COD problem is reducible to the $\binom{n}{n}$COD problem.

$$\binom{n}{k}\text{COD} \leq \binom{n}{n}\text{COD} \tag{4.2}$$

Proof.

Assuming the existence of a protocol $\mathcal{P}_{\binom{n}{n}}$ that solves $\binom{n}{n}$COD, a protocol $\mathcal{P}_{\binom{n}{k}}$ that solves the $\binom{n}{k}$COD problem can be constructed as follows:

Protocol 4.1 ($\mathcal{P}_{\binom{n}{k}} \leq \mathcal{P}_{\binom{n}{n}}$)**.**

Setting. S's input is a message vector $M : \{m_0, \ldots, m_{k-1}\}$.

If $\mathcal{P}_{\binom{n}{k}}$ is an interactive protocol, each receiver R_i holds an input x_i.

Step 1. S samples a vector $\mathcal{G}$ of $(n - k)$ distinct *garbage* messages.

Step 2. S generates a vector M' by concatenating M and $\mathcal{G}$.

Step 3. All participants engage in a $\mathcal{P}_{\binom{n}{n}}$ instance:

S uses M' as its input.

R_i uses x_i as its input (if the instance is interactive).

Post-protocol. Every receiver that got a valid message, takes it as its respective output.

Every receiver that got a *garbage* message discards it, as if no message had been received.

The protocol is correct as it complies with all the necessary conditions:

1. Given that the messages in $\mathcal{G}$ are sampled to contain non-significant information, only a subset of k receivers obtain one unique (valid) message of the original message vector M.
2. Per definition, S does not learn the destination of any message, including the *garbage* ones, nor does it learn the contents of the messages once received.

   S can not discern if R_i received a message in M or in $\mathcal{G}$, thus, S does not learn if R_i received a valid message.
3. Per definition, R_i does not learn the identifier, nor the contents, of the message received by R_j.

   R_i can not discern if R_j received a message in M or in $\mathcal{G}$, thus, R_i does not learn if R_j received a valid message.

□

Lemma 4.2.

The (k, n)OSD problem is reducible to the $(1, n)$OSD problem.

$$(k, n)OSD \leq (1, n)OSD \tag{4.3}$$

Proof.

Assuming the existence of a protocol $\mathcal{P}_{(1,n)}$ that solves the $(1, n)$OSD problem, a protocol $\mathcal{P}_{(k,n)}$ that solves the (k, n)OSD problem can be constructed in k rounds as follows:

Protocol 4.2 ($\mathcal{P}_{(k,n)} \leq \mathcal{P}_{(1,n)}$)**.**

Setting. S's input is a message vector $M : \{m_0, \ldots, m_{k-1}\}$.

If $\mathcal{P}_{(k,n)}$ is an interactive protocol, each receiver R_i holds an input X_i : $\{x_{i,0}, \ldots, x_{i,k-1}\}$.

Execution. For each message m_j in M, the participants engage in a $\mathcal{P}_{(1,n)}$ instance:

S uses m_j as its input.

R_i uses $x_{i,j}$ as its input.

Post-protocol. For each message obtained by R_i, the receiver inserts said message to its output set O_i.

The protocol is correct as it complies with all the necessary conditions:

1. The output message set O_i for each receiver R_i is formed only from messages in M.
2. Given that each message is sent individually, and per definition no two receivers obtain the same message, the output sets O_i, O_j are disjoint.
3. Given that $(1, n)$OSD is a special case of (k, n)OSD, the properties hold for each instance.

   No participant can learn meaningful information, other than its respective output, from the execution of any number of rounds, as it would contradict the privacy-preserving condition.

□

4.6.2 Remarks

Diagram

For a fixed value of n, the reducibility of all the possible instances of Oblivious Distribution, in any of its subfamilies, can be illustrated by the following diagram (figure 4.9).

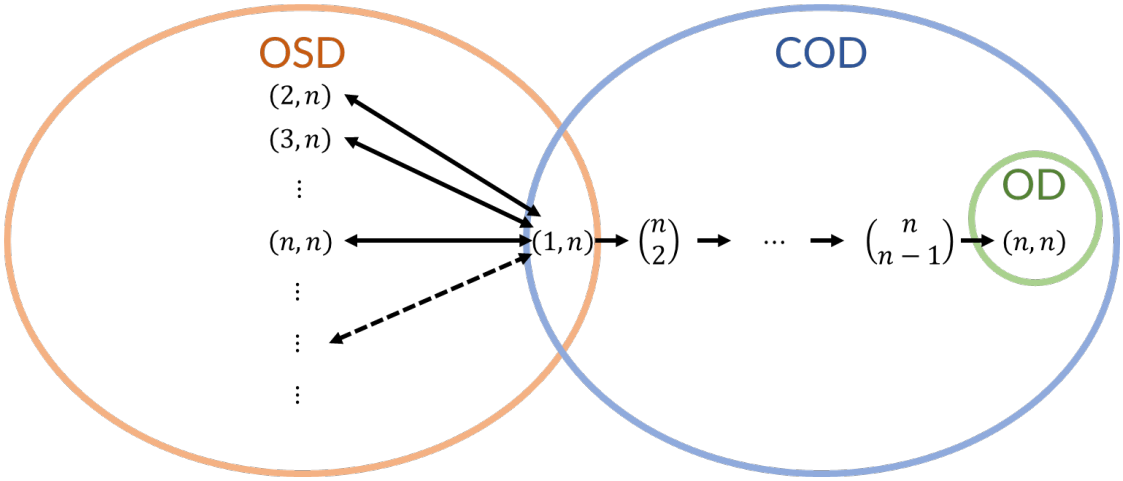

Figure 4.9: Intervariant reducibility diagram.

OD as the family-universal element

Remark 4. Although no proof is provided, it is strongly suggested that the *one-for-each* subfamily (OD) singleton is the only universal element. Due to the privacy-preserving properties, it is highly unlikely (if not impossible) to keep a proper record of the distribution of the messages between rounds, in order to achieve the *messages-receivers* bijection, without risking information leak.

Worst-case time cost

Remark 5. The reductions presented are polynomial-time (in n and k) transformations.

OSD equivalence class

Corollary 4.1.

The OSD subfamily forms an equivalence class.

Proof.

The statement is proved via a bi-reduction.

In lemma 4.2 the reduction $(k, n)OSD \leq (1, n)OSD$ is proven.

To be proven: $(1, n)OSD \leq (k, n)OSD$.

The protocol consists of the following procedure; for any value of k, the sender S generates $(k-1)$ *garbage* messages, and uses them to "fill" the empty spaces in M.

Given that $(1, n)$OSD is a special case of (k, n)OSD, the defining properties are properly inherited for each instance.

□

4.7 OD is universal for MPC

This section provides proof that the *one-for-each* OD subfamily is utilizable to solve any functionality $\mathcal{F}(x_0, \ldots, x_{n-1})$, corresponding to a Turing computable function by integrating a GC-based mechanism to solve the associated family of Boolean circuits.

First, a construction for a GC variant operating among three participants is established—denominated 3-party Garbled Circuit (3GC)—assuming the existence of an $(2, 2)$OD implementation.

- **3GC generation phase.** Given a Boolean circuit $\mathcal{C}$ for a functionality $\mathcal{F}$, S transforms it into a 3-party garbled circuit version. This phase is entirely computed by S.
- **3GC evaluation phase.** S sends the garbled circuit $\mathcal{C}_g$ to R_0, R_1. Some receiver—without loss of generality, in this work is set to R_0—proceeds to sequentially evaluate each gate (for gates where it has an input) by partaking in an $(2, 2)$OD protocol. At the end, the only significant new information R_0 and R_1 *learn* is the output bit for $\mathcal{C}_g$ evaluated on the private inputs of R_0 and R_1.

4.7.1 3GC generation

Lemma 4.3 (3-party garbled gate generation)**.**

For any gate g, there exists a construction by which three parties can interact to generate a 3-party garbled gate.

Proof.

Pre-protocol phase

The message pair that contains the ciphertext for the circuit's output $\{E_{k^0_{result}}(0), E_{k^1_{result}}(1)\}$, encrypted with the last gate's output key pair, must be provided to R_0, but not to R_1.

Protocol phase

For each gate, the participants proceed according to the following protocol:

Protocol 4.3 (Standalone 3-party gate generation)**.**

Setting. All keys are of fixed size m.

The gate to be evaluated is described by $z = g(x, y)$. Where g is any 2-input gate.

The possible output values (keys) for the gate are considered: $\{k^0_{out}, k^1_{out}\}$.

Step 1. S samples 2 pairs of one-time keys: $\{k^0_{R_0}, k^1_{R_0}\}, \{k^0_{R_1}, k^1_{R_1}\}$.

S sends $\{k^0_{R_1}, k^1_{R_1}\}$ to R_1.

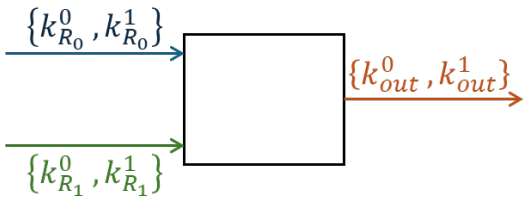

Figure 4.10: Standalone 3-party gate. Each input wire x, y (left) and the output wire (right) are labeled with the corresponding key pair.

Step 2. S constructs the truth table of the g gate.

x	y	z
0	0	$v_{0,0}$
0	1	$v_{0,1}$
1	0	$v_{1,0}$
1	1	$v_{1,1}$

Step 3. S substitutes the output values in the table to the key-encrypted version.

x	y	z
$k^0_{R_0}$	$k^0_{R_1}$	$k^{v_{0,0}}_{out}$
$k^0_{R_0}$	$k^1_{R_1}$	$k^{v_{0,1}}_{out}$
$k^1_{R_0}$	$k^0_{R_1}$	$k^{v_{1,0}}_{out}$
$k^1_{R_0}$	$k^1_{R_1}$	$k^{v_{1,1}}_{out}$

$\Longrightarrow$

Encrypted rows
$E_{k^0_{R_0}}(E_{k^0_{R_1}}(k^{v_{0,0}}_{out}))$
$E_{k^0_{R_0}}(E_{k^1_{R_1}}(k^{v_{0,1}}_{out}))$
$E_{k^1_{R_0}}(E_{k^0_{R_1}}(k^{v_{1,0}}_{out}))$
$E_{k^1_{R_0}}(E_{k^1_{R_1}}(k^{v_{1,1}}_{out}))$

Step 4. S randomly permutes the encrypted rows.

The resulting 4 messages, a *3-party garbled gate*, are sent as a unit to R_0

□

Remark 6. Whenever the output of a gate connects to the input of another gate, S does not sample a new key pair for that wire, instead, the output key pair becomes the input key pair for that wire.

Throughout exemplification is provided in appendix C.2.

4.7.2 3GC evaluation

Lemma 4.4 (3-party garbled gate evaluation)**.**

For any 3-party garbled gate, there exists an evaluation procedure that uses an interactive $(2,2)$OD instance instead of an OT instance.

Proof.

Protocol phase

For each gate, the participants proceed according to the following protocol:

Protocol 4.4 (Standalone 3-party garbled gate evaluation)**.**

Step 1. The participants engage in a $(2,2)\text{OD}_{XOR}$ execution:

S inputs the messages: $\{m_0 = k^0_{R_0}, m_1 = k^1_{R_0}\}$.

R_0 inputs its selection bit x.

R_1 inputs 0.

R_0 receives $m_x = k^x_{R_0}$.

R_1 receives $m_{\overline{x}} = k^{\overline{x}}_{R_0}$.

Step 2. R_1 sends $k^y_{R_1}$ to R_0.

Step 3. R_0 uses $k^x_{R_0}$, and $k^y_{R_1}$ to decrypt the garbled circuit.

R_0 obtains $k^{v_{x,y}}_{out}$.

Post-protocol phase

The last key computed/received by R_0 is used to open one of to circuit output messages $\{E_{k^0_{result}}(0), E_{k^1_{result}}(1)\}$ sent by S.

The decrypted value is published by R_0, to be of common knowledge to all participants.

□

Remark 7. The framework effectively emulates the function of an OT by fixing R_1's input (in step 1) without leaking significant information to any participant.

Throughout exemplification is provided in appendix C.3.

Nevertheless, this is an inefficient procedure which would be optimized by directly using an OT protocol between sender and evaluator. Its utilization—while not ideal—demonstrates that OD can be used to implement any function.

The protocol is more useful if thought of as an ad-hoc optimization in certain multi-party scenarios rather than a substitute for other MPC solutions.

4.7.3 nGC

To extend the universality result to the (n, n)OD protocols, the 3GC technique is properly modified to a general scenario.

First, an evaluator must be declared, from the list of receivers only. The gate's functionality will be that of an n-to-1 multiplexer—as illustrated by figure 4.11.

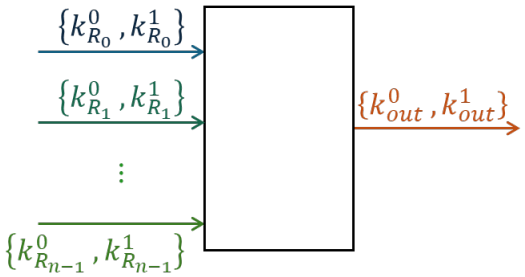

Figure 4.11: Stand alone $n+1$-input gate. Acting as an n-to-1 multiplexer.

Effectively, the desired functionality is that the resulting bit z is equal to the evaluators input. Assuming R_0 is the evaluator, the function must be equivalent to:

$$\begin{array}{ccccccccc} & & R_0 & & R_1 & & & & R_n \\ z & = & x_1 & \oplus & 0 & \oplus & \cdots & \oplus & 0 \end{array} \tag{4.4}$$

The Boolean value z is not contemplated for the (n, n)OD specification—where messages are addressed by means of a permutation vector.

To implement this functionality the range of permutations is reduced to a binary output:

$$\begin{array}{ccccccccccc} & & & R_0 & R_1 & R_3 & \cdots & R_n & & \\ \pi^0 & = & (& m_z, & m_{\bar{z}}, & 0, & \cdots, & 0 &) & \text{on } R_0\text{'s input } x_1 = 0 \\ \pi^1 & = & (& m_{\bar{z}}, & m_z, & 0, & \cdots, & 0 &) & \text{on } R_0\text{'s input } x_1 = 1 \end{array} \tag{4.5}$$

This is achieved by fixing every participant's input, except for the evaluator. The evaluation of the circuit proceeds by utilizing the n keys R_0 obtains from $R_1, \ldots, R_n$ to decrypt k^z_{out}.

4.8 Impossibility result in the presence of malicious parties

In this section, it is proven that no OD protocol implementation exists which is tolerant to malicious parties.

Definition 4.6 (Honest-participants scenario)**.**

Let the following settings be an execution of the $(2, 2)$OD protocol:

- S's input: $m_0 = 0, m_1 = 1$.
- R_0's input: $x = 0$
- R_1's input: $y = 0$.

Let the function $g(x, y)$ be an exclusive disjunction such that $z = x \oplus y$.

Let $M_P(m_0, m_1, x, y)$ be the snapshot of the collection of messages in the communication tapes generated by the TMs.

In an honest-participants scenario, each party follows the protocol strictly by *piping* its input to the protocol such that:

m_0	m_1	x	y	m_z	$m_{\overline{z}}$	messages
0	1	0	0	0	1	$M_P(0,1,0,0)$

Lemma 4.5 (Byzantine sender scenario)**.**

No $(2, 2)$OD implementation exists that can guarantee functional correctness when the sender has Byzantine behavior.

Proof.

An S participant incurs in Byzantine behavior by inverting the order of its input messages m_0, m_1 in the execution.

m_0	m_1	x	y	m_z	$m_{\overline{z}}$	messages
0	1	0	0	0	1	$M_P(0,1,0,0)$
1	0	0	0	1	0	$M_P(1,0,0,0)$

If the protocol were to correct the Sender's behavior, it would mean that $M_P(1, 0, 0, 0)$ would have to be mapped to the result of $M_P(0, 1, 0, 0)$, by inverting the outputs $m_z, m_{\overline{z}}$.

As a consequence, whenever the input of S would be $m_0 = 1, m_1 = 0$, the output messages would be wrongfully switched. □

Lemma 4.6 (Byzantine receiver scenario)**.**

No $(2, 2)$OD implementation exists that can guarantee functional correctness when a receiver has Byzantine behavior.

Proof.

Without loss of generality we assume R_0 is the receiver incurring in Byzantine behavior, who deviates from the protocol by negating its input bit x.

m_0	m_1	x	y	m_z	$m_{\overline{z}}$	messages
0	1	0	0	0	1	$M_P(0,1,0,0)$
0	1	1	0	1	0	$M_P(0,1,1,0)$

If the protocol where to correct the Receiver's behavior, it would mean that $M_P(0,1,1,0)$ would have to be mapped to the result of $M_P(0,1,0,0)$, by inverting the outputs $m_z, m_{\overline{z}}$.

As a consequence, whenever the input of R_0 would be $x = 1$, the output messages would be wrongfully switched. □

As a corollary of the previous lemmas, we conclude the following theorem:

Theorem 4.2 ($(2,2)$OD impossibility result)**.**

No $(2,2)$OD implementation exists that is tolerant to Byzantine participants.

Corollary 4.2 ((n,n)OD impossibility result)**.**

The Byzantine scenario previously discussed can be generalized to any receiver, thus, no *correction* can be made without impacting the correctness property of the protocol.

Therefore, no (n,n)OD implementation exists that is tolerant to Byzantine participants.

Corollary 4.3 (OD protocols are only passively-secure)**.**

From the previous statements is concluded that no actively-secure protocol can be implemented for the OD family.

Remark 8. Although OD protocols are only passively-secure, a possible *alternate route* is included in chapter 6.

4.9 Side results

The original GC protocol—as described in appendix B—has been scrutinized since its conception. Consequently, several optimizations have been proposed, minimizing the communication, and processing costs per gate by *shaving off* ciphertexts.

GRR3. The first garbled row reduction technique decreases the messages sent by composing the labels in the wires such that one of the four ciphertexts equals a fixed-size all-zeroes string, which can be omitted from communication [NPS99] [ZRE15].

Free-XOR. Exclusive to the XOR gate and achieved by assuming a common knowledge secret R, the input wires are labeled as the pairs $(A, A \oplus R)$ and $(B, B \oplus R)$. Thus, any XOR operation among the labels will yield either $A \oplus B$ or $A \oplus B \oplus R$ [NPS99] [KS08] [ZRE15].

GRR2. By using degree two polynomials over the four points formed by the row numbers and their respective keys, the interpolation allows to replace the garbled table by a pair of keys [Pin+09] [ZRE15].

Half gates. Exclusive to the AND gate; serves as an alternative to the GRR2 method that is compatible with the Free-XOR method. The generation of the messages assumes that the garbling party knows its input beforehand and constructs them in such a way that the two non-significant messages (i.e., those related to the truth value opposite to the input) are ciphered to all-zeroes strings, much like the GRR3 method [ZRE15].

These optimizations are stable message reductions, i.e., independent of the origin of the input for each gate.

In section 4.9.1, an alternative approach for message reduction in Garbled Circuits is presented; this technique works for gates where A has a direct input, by means of a partial function application.

4.9.1 Truth table partition technique for GC

The following original technique achieves a message (row) reduction from four to two for any gate with inputs from A and B by pre-computing A's input into the gate, inherently this requires A to know its respective input for a gate before the *garbling* phase starts.

This assumption is not limiting, as A is always required to have access to its input before the circuit evaluation starts, therefore, unless the execution is occurring on the rather specific scenario where forceful timing is needed, in which the circuit is generated before A has access input—which, in itself, would require an enforcing mechanism outside the protocol prescription; the proposed optimization is applicable.

Partition procedure

The conventional GC protocol *garbles* a gate $g \in C$, as described by the following abstract truth table:

x	y	z
0	0	$v_{0,0}$
0	1	$v_{0,1}$
1	0	$v_{1,0}$
1	1	$v_{1,1}$

Given that A knows the value of its input $x = \phi$, it can apply a partial function application process (from a binary function to a unary function), the resulting table is equal to either the upper or lower half of the previous one:

x	y	z
ϕ	0	$v_{\phi,0}$
ϕ	1	$v_{\phi,1}$

The normal procedure would transform the truth table to its sampled-key version. When the rows are reduced as stated, the OT can be used directly.

Consequently, no key pair sampling for x and y is necessary. Only the output values need to be transformed.

x	y	z
ϕ	0	k^0_{out}
ϕ	1	k^1_{out}

Partitioned gate evaluation

The gate evaluation is accordingly simplified:

- The OT protocol takes the input messages $\{k^0_{out}, k^1_{out}\}$, and B's y as the selection bit.
- B obtains k^y_{out}, which is either used as input for another gate, or as final output for the circuit.
- B does not learn $k^{\overline{y}}_{out}$.

Considerations for the partitioning technique

- The reduction is only applicable for gates where A has a direct input in a wire.
- The case in which a gate receives inputs from B and the output of another gate would be the most costly one, as it would require four ciphertexts in generation, and an OT for evaluation.
- The partition technique is no more efficient than the *half gates* approach, nor is it proven compatible with the *Free-XOR* method.

4.9.2 Message reduction on full participation for 3GC

The following observation is related to the previous reduction, but does not make use of a partial application. Instead, whenever there is full participation from all receivers for a certain gate, the *non-evaluating* receivers may participate directly; S does not generate individual key pairs for each receiver (only the evaluating receiver), thus, the communication cost is minimized.

Remark 9. In fact, no technique based on partial application could be used for 3GC; S, being the circuit generator, does not provide an input of its own to the logic gates.

Gate generation

In this scenario, no generation is required. No 3-party garbled circuit is generated.

Gate evaluation

The conventional 3GC protocol *garbles* a gate $g \in C$, as described by the following abstract truth table:

x	y	z
0	0	$k_{out}^{v_{0,0}}$
0	1	$k_{out}^{v_{0,1}}$
1	0	$k_{out}^{v_{1,0}}$
1	1	$k_{out}^{v_{1,1}}$

The participants engage in an interactive $(2,2)$OD of g, instead of the fixed gate XOR:

The sender S inputs the corresponding keys, as prescribed.

The receivers R_0, R_1 use their inputs, x, y respectively.

Assuming R_0 as the evaluating receiver:

R_0 receives k_{out}^{z} as intended.

R_1 receives $k_{out}^{\overline{z}}$.

Even in the scenario where R_1 receives all the complementary keys for all gates, they do not provide information of the evaluation of each gate. Furthermore, even if R_1 received a copy of the encrypted circuit output keys, the only new information that R_1 would learn would be the complement of the output bit. This value can be trivially computed by all participants at the end of the protocol.

Chapter 5

Implementation proposals

This chapter presents the following:

1. A listing of the necessary cryptographic assumptions made for the implementations presented.
2. An implementation for the $(2, 2)$OD protocol that relies on the Oblivious Transfer primitive.
3. A *weak-privacy* implementation for the (n, n)OD protocol that relies on secret-sharing schemes. The t-privacy of this implementation is compromised, as it needs a strictly honest sender to be preserved.
4. A construction for a *truly* private (n, n)OD implementation, addressing the necessary changes to the previous implementation, to strengthen its security.

All implementations presented depend upon cryptographic primitives. While it would be preferable to provide an information-theoretic implementation for each protocol, no such solutions could be produced.

For purposes of comparison, a chosen one-out-of-two OT implementation example is provided in appendix A, its security depends entirely on the cryptographic assumptions made.

It is worthy to note that implementations for Oblivious Transfer argued to be information-theoretic have their own considerations, for example:

- The implementation presented in [GMW87] uses 1-bit messages and relies on the notion of random bits in trap-door permutations.
- The implementations presented in [NP01] offer only partial information-theoretic protection, either to the sender or the receiver, while providing computational protection to the other participant.

5.1 Cryptographic assumptions

Cryptographic assumptions are fundamental conjectures (hypotheses) that allow for the construction of *seemingly*—yet not provable—secure communication protocols.

Of special interest for this work are the *generic* assumptions which, as described in [GT15], consist of those who postulate the existence of cryptographic primitives.

Listed are the *intuitive* notions required to express the assumptions undertaken by this work, yet not properly describe them.

5.1.1 Computational hardness

A function f is considered hard (difficult) to solve if no expected polynomial time algorithm $\mathcal{A}$ exists that on every valid input x outputs $f(x)$ [Gol01] [Gol03].

$$\nexists \mathcal{A} \mid \mathcal{A}(x) = f(x)$$

To proof the previous condition, would be impossible; nonetheless, there are problems for which there is a *strong enough* evidence that there is no efficient solution for them—such as [Gol01]:

Integer factorization. It is hard, on any number x product of two large primes p, q (i.e., $x = p \cdot q$), to determine the values of p and q from x.

Discrete logarithm. For a prime p, it is hard, given only $x, y \in \mathbb{Z}_p^* \mid x \neq 1$, to compute an integer k when $x^k = y$.

Computationally bound adversaries

An adversary (section 2.2.1) on a cryptosystem is considered bounded if—due to its inherent computational hardness—for every algorithm that it executes is solvable in expected polynomial time [Gol03].

5.1.2 Cryptographic primitives

One-way functions

A function $f : \{0,1\}^* \to \{0,1\}^*$ is considered *one-way* if the following conditions are met [Gol03]:

Easy to compute. There exists an efficient algorithm $\mathcal{A}$ that on input x, outputs $f(x)$.

Hard to invert. Any feasible algorithm that *tries* to find a preimage for $f(x)$, succeeds with negligible probability.

One-way functions have not been demonstrated to exist. A precise definition and in depth details can be found in [Gol01].

Collision-free hash functions

A hash function is any function that maps any arbitrary-size input data string to a fixed-size output data sting, $f : \{0,1\}^* \to \{0,1\}^m$.

A hash function is considered collision-free if it is computationally infeasible to find different inputs that produce the same output value [Dam87].

A precise definition and in depth details can be found in [Dam87].

Random number generator

A function is considered a *random number generator* (RNG) if it generates a sequence of bits which pattern is uniformly distributed.

In practice, *pseudorandom* approximations provide deterministic sequences which appear random; even if the produced output is not *truly* random [Gol01].

A precise definition and in depth details can be found in [Gol03].

Encryption schemes

An encryption scheme is a collection of mechanisms that allow for the processing of information in order to restrict its access to, ideally, only a defined set of participants. All encryption schemes consists of three algorithms; *key generation*, *encryption*, and *decryption* [Gol01]. A message is *encrypted*—using an *encryption key*— making it unfeasible for any party that does not have access to the respective *decryption key* to recover the contents of the message. Depending on their definition of security, the schemes are classified as follows [Gol01]:

Private-key encryption (symmetric-key encryption)**.** The same key is used for encryption and decryption.

Public-key encryption (asymmetric-key encryption)**.** Different keys are used for encryption and decryption, and its security should hold when an adversary has access to the encryption key.

Proper definitions for both encryption schemes and in depth analysis of the formalization of the notion of security can be found in [Gol01].

5.2 Proposal 1

Interactive 2-on-2 OD implementation using OT, collision-free hash functions, symmetric cryptography, and asymmetric cryptography

The $(2, 2)$OD protocol implementation addressed below assumes the following conditions are met:

- A random number generator function.
- An Oblivious Transfer implementation.
- A collision-free hash function H.
- A private-key scheme (E representing the encryption function).
- A public-key scheme.
- All the previous algorithms are of common knowledge and usable by all participants.
- There are bidirectional peer-to-peer private communication channels among all participants.

Implementation 5.1.

Setting. S's input: (m_0, m_1).

R_0's input: x.

R_1's input: y.

Step 1. R_0 and R_1 generate each a 1-time public key, pk_{R_0} and pk_{R_1} respectively.

Step 2. R_1 sends R_0 its key pk_{R_1}

Step 3. R_0 forms the tuple (pk_{R_0}, pk_{R_1}).

Step 4. R_0 samples a pair of nonces $(\text{nonce}_0, \text{nonce}_1)$.

R_0 hashes each nonce to generate $(h_0 = H(\text{nonce}_0), h_1 = H(\text{nonce}_1))$.

Step 5. R_0 samples a pair of symmetric keys (k_0, k_1).

Step 6. R_0 generates 2 encrypted messages according to the truth table of the respective function $g(x, y)$.

x	y	
0	0	$h_{g(0,0)}$
0	1	$h_{g(0,1)}$
1	0	$h_{g(1,0)}$
1	1	$h_{g(1,1)}$

$$\longrightarrow \begin{cases} E_{k_0}(h_{g(0,0)}), E_{k_1}(h_{g(0,1)}) & \text{if } x \text{ is } 0 \\ E_{k_0}(h_{g(1,0)}), E_{k_1}(h_{g(1,1)}) & \text{if } x \text{ is } 1 \end{cases}$$

Forming the tuple C.

$$C = (E_{k_0}(h_{g(x,0)}), E_{k_1}(h_{g(x,1)})) \tag{5.1}$$

Step 7. R_0 sends R_1 the tuple C.

Step 8. R_0 and R_1 engage in an OT:

- R_0's input: (k_0, k_1)
- R_1's input: y
- R_1's output: k_y

Step 9. R_1 decrypts each message in the tuple C using k_y to obtain $h_{g(x,y)}$.

Step 10. R_1 sends R_0 the hash $h_{g(x,y)}$.

Step 11. R_0 generates π according to the hash obtained:

$$\pi = \begin{cases} (pk_{R_0}, pk_{R_1}) & \text{if } h_{g(x,y)} \text{ is } h_0 \\ (pk_{R_1}, pk_{R_0}) & \text{if } h_{g(x,y)} \text{ is } h_1 \end{cases} \tag{5.2}$$

Step 12. R_0 sends S the tuple π.

Step 13. S encrypts (m_0, m_1), with the keys in π.

Step 14. S sends R_0 and R_1 a copy of the vector of asymmetrically encrypted messages.

Step 15. Each receiver decrypts the vector to obtain its corresponding output message.

On privacy

To demonstrate that the property of privacy is preserved, the following views are discussed.

Lemma 5.1 (S's view)**.**

S cannot determine which message is sent to each receiver $\{R_0, R_1\}$.

Proof.

The only message received by S is the tuple π (step 12). This tuple contains both of the one time public keys $\{pk_{R_0}, pk_{R_1}\}$.

Given that the permutation of the keys is determined by the value $z = g(x, y)$ (step 11); S cannot discern which key belongs to either receiver, thus, S cannot determine which message will each receiver be able to decrypt. □

Lemma 5.2 (R_0's view)**.**

R_0 cannot determine the contents of the message received by R_1, nor does it learn the value of y by means of the protocol.

Proof.

The first message received by R_0 is pk_{R_1} (step 2). Being a 1-time public key, it does not reveal any relevant information.

R_0 and R_1 engage in the OT protocol (step 8) to evaluate the 1-gate GC of $g(x, y)$ (step 5–8). Given that OT is assumed private, R_0 does not learn y by mere communication.

The second message received by R_0 is $h_{g(x,y)}$ (step 11), a hash that was already known to R_0 (step 4)—it only indicates to R_0 the output for the function $g(x, y)$, which must be of common knowledge to both participants.

The last message received by R_0 is the vector of the messages $\{m_0, m_1\}$ (step 15) each asymmetrically encrypted by one of the public keys $\{pk_{R_0}, pk_{R_1}\}$ (step 14). Given that R_0 does not know the private key related to pk_{R_1}; R_0 cannot read the contents of the message destined to R_1. □

Remark 10. In the previous proof it is indicated that R_0 does not learn y by mere communication in the OT protocol. Although there are functions for which certain inputs will always leak the input y—e.g., for the AND gate—whenever $x = 1$ once the output is known to R_0 it can trivially calculate the value of y; this does not contradict our definition of privacy, for this information is not leaked by the protocol.

Lemma 5.3 (R_1's view)**.**

R_1 cannot determine the contents of the message received by R_0, nor does it learn the value of x by means of the protocol.

Proof.

First, R_1 receives the tuple C (step 7), then R_1 obtains the key k_y (step 8). Given that OT is assumed private, R_1 does not learn x by mere communication.

The last message received by R_1 is the vector of the messages $\{m_0, m_1\}$ (step 14) each asymmetrically encrypted by one of the public keys $\{pk_{R_0}, pk_{R_1}\}$ (step 13). Given that R_1 does not know the private key related to pk_{R_0}, R_1 cannot read the contents of the message destined to R_0. □

Remark 11. Similarly to remark 10, even when there are certain contexts where R_1 can learn x, it is as a secondary effect of the function and the inputs, but not the protocol.

Given that no coalition can occur among participants—as it would exceed the limit, $t < \frac{n}{2}$—the privacy argument comprises only the collection of views of the participants.

Corollary 5.1 ($(2, 2)$OD privacy)**.**

As a conclusion of the proof provided for lemma 5.1, lemma 5.2, and lemma 5.3; the protocol is proven secure.

5.3 Proposal 2

Weak-privacy interactive n-on-n OD implementation using secret-sharing schemes and asymmetric cryptography

The following *weak-privacy* (n, n)OD protocol implementation presents a solution that does not adhere to the condition of t-privacy (discussion further below).

This *weakened* version directly violates the third condition stated:

> For any pair of indexes $i, j \in [0, n-1] \mid i \neq j$, R_i must not be able to determine, by means of the protocol, which message was received by R_j.

Thus, it can not be used to implement the COD and OSD variants.

Nevertheless, it serves as a lower-cost implementation that could suffice in certain scenarios, while it exposes the core ideas on which the *strong* version is build.

The implementation assumes the following conditions are met:

- A random number generator function.
- A (k, n) secret-sharing scheme implementation.
- An algorithm $\mathcal{A}$ that calculates a permutation vector from the individual inputs of each participant $x_i \in [1, n]$ through the previous secret-sharing scheme (see remark 13).

$$\mathcal{A}(x_0, \ldots, x_{n-1}) = (\pi_1, \ldots, \pi_n) \tag{5.3}$$

- An public-key scheme ($\mathscr{E}$ representing the encryption algorithm).
- All the previous algorithms are of common knowledge and usable by all participants.
- There are bidirectional peer-to-peer private communication channels among all participants.

Implementation 5.2.

Inputs: S's input: $\{m_0, \ldots, m_{n-1}\}$.

For $i \in [0, n-1]$, R_i samples a random number as its input $x_i \xleftarrow[[0,n-1]]{} \$$.

Step 1: Each receiver R_i generates a 1-time public key pk_{R_i} and makes it public to all other receivers (but not to the sender).

R_i generates a public key vector $\{pk_{R_0}, \ldots, pk_{R_{n-1}}\}$.

Step 2: Receivers $R_0, \ldots, R_{n-1}$ engage in the secret-sharing solution for $\mathcal{A}(x_0, \ldots, x_{n-1})$.

All receivers obtain the output permutation vector $(\pi_1, \ldots, \pi_n)$.

Step 3: R_0 uses the vector $(\pi_1, \ldots, \pi_n)$ to rearrange the public key vector:

$$(pk_{R_{\pi(0)}}, \ldots, pk_{R_{\pi(n-1)}}) \tag{5.4}$$

Step 4: R_0 sends the reordered public key vector to S.

Step 5: S encrypts each message:

$$\{\mathscr{E}_{pk_{R_{\pi(0)}}}(m_0), \ldots, \mathscr{E}_{pk_{R_{\pi(n-1)}}}(m_{n-1})\} \tag{5.5}$$

Step 6: S sends a copy of the resulting permuted public-key encrypted message vector to every receiver.

Step 7: R_i uses its private key to decrypt the respective message, obtaining the message $m_{R_{\pi(i)}}$.

Remark 12. In steps 3 and 4, R_0 is set as the intermediate to S. This role can be delegated to any single receiver or any subset of receivers.

Remark 13. As stated anteriorly, threshold secret-sharing schemes are complete for MPC (i.e., there is proof that any arithmetic circuit may be implemented by means of the Circuit Evaluation with Passive Security (CEPS) [Cra15])—therefore—the existence of an algorithm that generates the permutation is assumed.

Algorithm 5.1 Random permutation [Dur64]

Require:
Array a, to be permuted.
Integer n, value of the length of a.
Random number generator function `random`, uniformly distributed over the interval $(0, 1)$.

Ensure:
`SHUFFLE` applies a random permutation to the sequence $a[i] \mid i = 1, \ldots, n$.

```
procedure SHUFFLE(a, n, random)
    for i := n step −1 until 2 do
        j := int(i×random+1)
        swap a[i] and a[j]
    end for
end procedure
```

Akin to Durstenfeld's approach (algorithm 5.1), the computational algorithm must provide an unbiased permutation. Since the scenario is constrained to semi-honest participants, a proper distribution is expected.

On privacy

The following observations are proffered:

1. Any sole receiver R_j that (passively) colludes with S can compromise the execution:

   ***S*.** Learns which receiver gets each message.

   ***R_j*.** Learns the content of the messages received by its peers.

As a consequence, a *weakened* privacy condition is defined:

Condition 5.1 (t-privacy [*OD receiver-coalitions only*])**.**

An OD protocol is t-private to receiver coalitions if for any subset of corrupt receivers $C \subset R_0, \ldots, R_n \mid S \notin C$—of size at most t—sharing all the information available to them:

- A corrupt participant $R_i \in C$ learns no information beyond $\{x_j, y_j\}_{R_j \in C}$ and whatever information can be computed in expected polynomial time from the values $\{x_j, y_j\}$ by participating in the protocol's execution.
- S cannot determine which receiver obtains which message.

The following views are discussed, to provide proof of adherence to the t-privacy [*OD receiver-coalitions only*] condition.

Lemma 5.4 (S's view)**.**

S cannot determine which message each receiver obtains.

Proof.

The only message received by S (step 4) is the permuted public key vector (equation (5.5)).

Given that S does not learn the permutation vector $\pi_1, \ldots, \pi_n$ and ignores the origin of each key; S cannot determine which message will each receiver be able to decrypt. □

Lemma 5.5 (R_i's view)**.**

$\forall i \forall j \mid i, j \in [0, n-1] \mid i \neq j$; R_i cannot determine the contents of the message received by R_j, nor does it learn the value of its input (x_j) by means of the protocol.

Proof.

R_i obtains the public keys of its peers (step 1); without the private keys, this information is contextually unusable.

R_i obtains the permutation vector (step 2); given the utilization of this vector, this information must be public to all receivers.

R_i receives a copy of the output public-key encrypted message vector (step 5); without the respective private keys, R_i can not read the messages destined to its peers.

Through the public keys, R_i can only relate each message to its receiver; which was already of knowledge to R_i by the permutation vector. □

Theorem 5.1 ((n, n)OD privacy [*receiver-coalitions only*])**.**

Undertaking the non-colluding S assumption, no t-coalition may learn more than intended, for $t < \frac{n}{2}$.

Proof.

Any coalition $C \subset R_0, \ldots, R_n$ of at most t receivers that fully shares their views on the protocol can not read the output message for any receiver:

- Plaintext messages are never provided by an honest S.
- Ciphertext messages can not be decrypted without the proper private keys.

□

5.4 Proposal 3

Strong-privacy interactive n-on-n OD implementation using secret-sharing schemes and asymmetric cryptography

In order to provide an implementation that adheres to the *non-weakened* t-privacy condition (condition 3.4) the following changes are to be made to the previous protocol:

1. The permutation vector—$\pi_1, \ldots, \pi_n$—must remain unknown to all receivers.
2. The permuted vector of public keys—$pk_{R_{\pi(1)}}, \ldots, pk_{R_{\pi(n)}}$—must be computed privately.

The operative description through a trusted third party $\mathcal{T}$ is as follows:

Ideal protocol 5.1 (t-private (n, n)OD)**.**

Setting. Sender S determines the message set $\{m_0, \ldots, m_{n-1}\}$.

Each receiver R_i determines samples an input public key pk_{R_i}.

Step 1. S sends $\{m_0, \ldots, m_{n-1}\}$ to $\mathcal{T}$.

Step 2. Each receiver R_i sends pk_{R_i} to $\mathcal{T}$.

Step 3. $\mathcal{T}$ samples a random permutation $\pi_1, \ldots, \pi_n$.

Step 4. $\mathcal{T}$ sends to each receiver R_i the message that corresponds to the position after the permutation is applied encrypted with its respective public key pk_{R_i}.

$S:$	input $m_0, \ldots, m_{n-1}$
$R_i:$	input $pk_{R_i} \leftarrow \$$
$S \to \mathcal{T}:$	$\{m_0, \ldots, m_{n-1}\}$
$R_i \to \mathcal{T}:$	pk_{R_i}
$\mathcal{T}:$	$\pi_1, \ldots, \pi_n \leftarrow \$$
$\mathcal{T} \to R_i:$	$\mathscr{E}_{pk_{R_i}}(m_{\pi(i)})$

Figure 5.1: Ideal t-private (n, n)OD protocol.

To achieve this, the protocol's implementation would require to be modified accordingly:

Implementation 5.3.

Inputs: S's input: $\{m_0, \ldots, m_{n-1}\}$.

R_i's input consists of the following vectors:

- R_i samples a random permutation vector $(\pi_1^{R_i}, \ldots, \pi_n^{R_i})$.
- R_i generates a 1-time public key pk_{R_i} and writes it in a vector such that it is in the position corresponding to its identification number i, filling the remaining positions with zeroes:

0	$\ldots$	$i-1$	i	$i+1$	$\ldots$	n
0	$\ldots$	0	pk_{R_i}	0	$\ldots$	0

Step 1: Receivers $R_0, \ldots, R_n$ engage in a secret-sharing execution such that:

- All partial public key vectors are added to form the total public key vector $\{pk_{R_0}, \ldots, pk_{R_n}\}$.
- Every permutation vector is operated to generate the final permutation:

$$(\pi_1^{R_0}, \ldots, \pi_n^{R_0}) \times \cdots \times (\pi_1^{R_n}, \ldots, \pi_n^{R_n}) = (\pi_1^F, \ldots, \pi_n^F) \tag{5.6}$$

- The final permutation vector is applied to the public key vector:

$$\{pk_{R_{\pi^F(1)}}, \ldots, pk_{R_{\pi^F(n)}}\} \tag{5.7}$$

Step 2: The resulting permuted public key vector is sent to S.

Step 3: S encrypts each message accordingly:

$$(\mathscr{E}_{pk_{R_{\pi^F(1)}}}(m_0), \ldots, \mathscr{E}_{pk_{R_{\pi^F(n)}}}(m_{n-1})) \tag{5.8}$$

Step 4: S sends a copy of the resulting permuted public-key encrypted message vector to every receiver.

Step 5: R_i uses its private key to decrypt the respective message, obtaining the message $m_{R_{\pi^F(i)}}$.

Remark 14. Secret-sharing schemes are compatible with vector and matrix operations—as demonstrated in the article *An Image Secret Sharing Method Based on Matrix Theory* [Din+18]—but due to the complexity of such operations the step-by-step procedure is omitted.

Chapter 6

Conclusion and Future Work

The main drive of this thesis was to explore the Oblivious Distribution problem—an original variant to the Oblivious Transfer problem. The OT primitive, while proven complete for MPC, is by design a two-party approach, which in turn presents limitations in terms of efficiency, scalability, and practicality. OD provides a family of ad-hoc protocols that address the unfeasibility of OT.

Three subfamilies—*one-for-each*, *n-choose-k* and *subset*—are defined, and reductions are provided to prove that the *n-choose-k* and *subset* protocols can be constructed from *one-for-each* instances.

Meanwhile, by leveraging the $(2, 2)$OD protocol with a modified Garbled Circuit framework the 3-party approach is proven complete for MPC. Which by extension demonstrates universality for the (n, n)OD.

On the privacy aspect, proof is provided of the inherent limit of the OD approach—as it is non-tolerant to *malicious* adversaries—for which a contingency plan may be needed; the Universal Composition framework [Can00b] presents a general-purpose privacy-preserving procedure to incorporate cryptographic protocols as modules in the presence of active Byzantine participants by amplifying the simulation technique (introduced by the real-ideal paradigm in section 3.1.1) into a protocol emulation architecture.

Within this context, 2PC protocol may be processed by a compiler and transformed into Byzantine-tolerant protocols; this is achieved by *forcing* the participants to act honestly by having them commit their input, then setting their random tapes uniformly. The information so far is of common knowledge—therefore—any new message computed from it is an NP statement. This allows for the operational principle of zero-knowledge proofs, by which, a participant obtains proof that with a satisfyingly high probability its peer is following the prescribed protocol [Can+02].

Due to sheer complexity, further research is needed to ensure the OD protocols are universally composable.

Lastly, on the practical side, implementation examples are explored. Although both examples internally operate on other MPC approaches, this does not undermine the utility of OD as a flexible optimization framework, nor it is proved that these are the only possible implementations. Further research may allow decreasing the proposed requirements to only depending on basic primitives.

Laterally, a useful GC message reduction was developed, based on partial function application; while a communication minimization for 3GC, on gates with full participation, was achieved.

In the last days of writing this thesis, it was observed that the universality argument is applicable on any *n-choose-k* instance, for any $k \geq 2$. This observation requires further research to reach any valuable conclusion.

References

[Dur64] R. Durstenfeld. "Algorithm 235: Random permutation". In: *Communications of the ACM* 7.7 (July 1964), p. 420. DOI: 10.1145/364520.364540.

[Kar72] R. M. Karp. "Reducibility among Combinatorial Problems". In: *Complexity of Computer Computations*. Springer US, 1972, pp. 85–103. ISBN: 9781468420012. DOI: 10.1007/978-1-4684-2001-2_9.

[Bla79] G. R. Blakley. "Safeguarding cryptographic keys". In: *Managing Requirements Knowledge, International Workshop on*. Los Alamitos, CA, USA: IEEE Computer Society, June 1979, p. 313. DOI: 10.1109/AFIPS.1979.98.

[Sha79] A. Shamir. "How to share a secret". In: *Communications of the ACM* 22.11 (Nov. 1979), pp. 612–613. DOI: 10.1145/359168.359176.

[SRA79] A. Shamir, R. L. Rivest, and L. M. Adleman. "Mental Poker". In: *MIT/LCS/TM-125* (1979). DOI: 10.1007/978-1-4684-6686-7_5.

[Yao82] A. C. Yao. "Protocols for secure computations". In: *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. IEEE, Nov. 1982. DOI: 10.1109/sfcs.1982.38.

[EGL85] S. Even, O. Goldreich, and A. Lempel. "A randomized protocol for signing contracts". In: *Communications of the ACM* 28.6 (June 1985), pp. 637–647. DOI: 10.1145/3812.3818.

[GHY85] Z. Galil, S. Haber, and M. Yung. "A private interactive test of a boolean predicate a minimum-knowledge public-key cryptosystems". In: *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*. IEEE, 1985, pp. 360–371. DOI: 10.1109/sfcs.1985.1.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. "The knowledge complexity of interactive proof-systems". In: *Proceedings of the seventeenth annual ACM symposium on Theory of computing - STOC '85*. STOC '85. ACM Press, 1985, pp. 291–304. DOI: 10.1145/22145.22178.

[Yao86] A. C.-C. Yao. "How to generate and exchange secrets". In: *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. 1986, pp. 162–167. DOI: 10.1109/SFCS.1986.25.

[Dam87] I. B. Damgård. "Collision Free Hash Functions and Public Key Signature Schemes". In: *Advances in Cryptology — EUROCRYPT '87*. Springer Berlin Heidelberg, Apr. 1987, pp. 203–216. ISBN: 9783540191025. DOI: 10.1007/3-540-39118-5_19.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. "How to play ANY mental game". In: *Proceedings of the nineteenth annual ACM conference on Theory of computing - STOC '87*. STOC '87. ACM Press, 1987. DOI: 10.1145/28395.28420.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. "Completeness theorems for non-cryptographic fault-tolerant distributed computation". In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. STOC '88. Chicago, Illinois, USA: Association for Computing Machinery, 1988, pp. 1–10. ISBN: 0897912640. DOI: 10.1145/62212.62213.

[Kil88] J. Kilian. "Founding cryptography on oblivious transfer". In: *Proceedings of the twentieth annual ACM symposium on Theory of computing - STOC '88*. STOC '88. ACM Press, 1988. DOI: 10.1145/62212.62215.

[CK89] B. Chor and E. Kushilevitz. "A zero-one law for Boolean privacy". In: *Proceedings of the twenty-first annual ACM symposium on Theory of computing - STOC '89*. STOC '89. ACM Press, 1989. DOI: 10.1145/73007.73013.

[MR91] S. Micali and P. Rogaway. *Secure Computation (Preliminary Report) [MIT-LCS-TR-511]*. Tech. rep. Massachusetts Institute of Technology, Aug. 1991.

[Bea95] D. Beaver. "Precomputing Oblivious Transfer". In: *Advances in Cryptology—CRYPT0' 95*. Ed. by D. Coppersmith. Vol. 963. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 97–109. ISBN: 978-3-540-44750-4. DOI: https://doi.org/10.1007/3-540-44750-4_8.

[FMR96] M. J. Fischer, S. Micali, and C. Rackoff. "A secure protocol for the oblivious transfer (extended abstract)". In: *Journal of Cryptology* 9.3 (June 1996), pp. 191–195. DOI: 10.1007/bf00208002.

[NP99] M. Naor and B. Pinkas. "Oblivious transfer and polynomial evaluation". In: *Proceedings of the thirty-first annual ACM symposium on Theory of Computing*. STOC99. ACM, May 1999. DOI: 10.1145/301250.301312.

[NPS99] M. Naor, B. Pinkas, and R. Sumner. "Privacy preserving auctions and mechanism design". In: *Proceedings of the 1st ACM conference on Electronic commerce*. EC99. ACM, Nov. 1999. DOI: 10.1145/336992.337028.

[Can00a] R. Canetti. "Security and Composition of Multiparty Cryptographic Protocols". In: *Journal of Cryptology* 13.1 (Jan. 2000), pp. 143–202. DOI: 10.1007/s001459910006.

[Can00b] R. Canetti. *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. Cryptology ePrint Archive, Paper 2000/067. https://eprint.iacr.org/2000/067. 2000.

[Kap00] B. Kapron. *Andrew Chi-Chih Yao*. English. 2000. URL: https://amturing.acm.org/award_winners/yao_1611524.cfm (visited on 02/04/2024).

[Gol01] O. Goldreich. *Foundations of Cryptography: Volume 1, Basic Tools. Oded*. Ed. by O. Goldreich. First. Vol. 1. Literaturverz. S. [355] - 365. Cambridge, U.K: Cambridge University Press, 2001. 372 pp. ISBN: 9780521791724.

[NP01] M. Naor and B. Pinkas. "Efficient oblivious transfer protocols". In: *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '01. Washington, D.C., USA: Society for Industrial and Applied Mathematics, 2001, pp. 448–457. ISBN: 0898714907.

[Can+02] R. Canetti et al. "Universally composable two-party and multi-party secure computation". In: *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. STOC02. ACM, May 2002. DOI: 10.1145/509907.509980.

[Gol03] O. Goldreich. "Cryptography and cryptographic protocols". In: *Distributed Computing* 16.2–3 (Sept. 2003), pp. 177–199. DOI: 10.1007/s00446-002-0077-1.

[Rab05] M. O. Rabin. "How to Exchange Secrets with Oblivious Transfer". In: *IACR Cryptol. ePrint Arch.* 2005 (2005), p. 187.

[Bar07] B. Barak. "Lecture 19 - Oblivious Transfer (OT) and Private InformationRetrieval (PIR)". Nov. 2007.

[KS08] V. Kolesnikov and T. Schneider. "Improved Garbled Circuit: Free XOR Gates and Applications". In: *Automata, Languages and Programming*. Ed. by L. A. H. M. M. I. A. W. I. Damgård Ivanand Goldberg. Springer Berlin Heidelberg, 2008, pp. 486–498. ISBN: 9783540705833. DOI: 10.1007/978-3-540-70583-3_40.

[Bog+09] P. Bogetoft et al. "Secure Multiparty Computation Goes Live". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009, pp. 325–343. ISBN: 9783642035494. DOI: 10.1007/978-3-642-03549-4_20.

[Pin+09] B. Pinkas et al. "Secure Two-Party Computation Is Practical". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009, pp. 250–267. ISBN: 9783642103667. DOI: 10.1007/978-3-642-10366-7_15.

[HDS12] X. Hei, X. Du, and B. Song. "Two matrices for Blakley's secret sharing scheme". In: *2012 IEEE International Conference on Communications (ICC)*. IEEE, June 2012. DOI: 10.1109/icc.2012.6364198.

[Sip13] M. Sipser. *Introduction to the theory of computation*. Third edition, international edition. Hier auch später erschienene, unveränderte Nachdrucke. Australia: Cengage Learning, 2013. 458 pp. ISBN: 9780357670583.

[CO15] T. Chou and C. Orlandi. "The Simplest Protocol for Oblivious Transfer". In: *Progress in Cryptology – LATINCRYPT 2015*. Ed. by K. Lauter and F. Rodríguez-Henríquez. Cham: Springer International Publishing, 2015, pp. 40–58. ISBN: 978-3-319-22174-8.

[Cra15] R. J. F. Cramer. *Secure Multiparty Computation and Secret Sharing*. Ed. by I. B. Damgård and J. B. Nielsen. Literaturverzeichnis: Seite 359-367. Cambridge, United Kingdom: Cambridge University Press, 2015. 373 pp. ISBN: 9781107043053.

[GT15] S. Goldwasser and Y. Tauman Kalai. "Cryptographic Assumptions: A Position Paper". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Dec. 2015, pp. 505–522. ISBN: 9783662490969. DOI: 10.1007/978-3-662-49096-9_21.

[Qin15] W. QingLong. *Efficient k-out-of-n oblivious transfer protocol*. Cryptology ePrint Archive, Paper 2015/218. https://eprint.iacr.org/2015/218. 2015.

[ZRE15] S. Zahur, M. Rosulek, and D. Evans. "Two Halves Make a Whole: Reducing Data Transfer in Garbled Circuits Using Half Gates". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2015, pp. 220–250. ISBN: 9783662468036. DOI: 10.1007/978-3-662-46803-6_8.

[Ebe16] M. Eberl. *Fisher–Yates shuffle*. Formal proof development. Sept. 30, 2016. URL: https://isa-afp.org/entries/Fisher_Yates.html.

[BK17] N. Büscher and S. Katzenbeisser. *Compilation for Secure Multi-party Computation*. Springer International Publishing, 2017. ISBN: 9783319675220. DOI: 10.1007/978-3-319-67522-0.

[Din+18] W. Ding et al. "An Image Secret Sharing Method Based on Matrix Theory". In: *Symmetry* 10.10 (Oct. 2018), p. 530. DOI: 10.3390/sym10100530.

[EKR18] D. Evans, V. Kolesnikov, and M. Rosulek. "A Pragmatic Introduction to Secure Multi-Party Computation". In: *Foundations and Trends in Privacy and Security* 2.2–3 (2018), pp. 70–246. DOI: 10.1561/3300000019.

[Hsu+18] J.-C. Hsu et al. "Oblivious Transfer Protocols Based on Commutative Encryption". In: *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE, Feb. 2018. DOI: 10.1109/ntms.2018.8328707.

[CB19] J. I. Choi and K. R. B. Butler. "Secure Multiparty Computation and Trusted Hardware: Examining Adoption Challenges and Opportunities". In: *Security and Communication Networks* 2019 (Apr. 2019), pp. 1–28. DOI: 10.1155/2019/1368905.

[Mal19] D. Malkhi. *Concurrency the Works of Leslie Lamport. the Works of Leslie Lamport*. Morgan & Claypool Publishers, 2019. ISBN: 9781450372732.

Appendix A

OT implementation example

The following implementation [CO15] for the chosen one-out-of-two OT protocol is based on the Diffie-Hellman key-exchange protocol (detailed information available on [DH76]) which, in turn, assumes the existence of a symmetric cryptosystem scheme and the existence of a collision-free hash (digestion) function (section 5.1.2).

A.1 DH-based OT implementation

Implementation A.1 (DH-based $\binom{2}{1}$OT [CO15])**.**

Setting. Sender S determines its input tuple (m_0, m_1).

Receiver R determines its input bit c.

Step 1. Each participant samples a random number in $\mathbb{Z}_p$.

Step 2. S sends $A = g^a$ to R.

Step 3. R calculates B where:

If c is 0 then $B = g^b$

If c is 1 then $B = Ag^b$

Step 4. R sends B to S.

Step 5. S calculates the key pair k_0, k_1 where:

k_0 is the result of a hash function of B^a

k_1 is the result of a hash function of $(\frac{B}{A})^a$

Step 6. R calculates the key k_R as the result of a hash function of A^b.

Step 7. S encrypts both input messages m_0, m_1 with its respective key k_0, k_1, generating e_0, e_1, and sends them to R.

Step 8. R will decrypt both messages with k_R, one resulting in garbage and the other one in the output message m_c.

Sender S		Receiver R
input: (m_0, m_1)		input: c
output: $\emptyset$		output: m_c
S view	$S \xleftrightarrow[\text{channel}]{} R$	R view
$a \in \mathbb{Z}_p \leftarrow \$$		$b \in \mathbb{Z}_p \leftarrow \$$
	$\xrightarrow{A = g^a}$	
		if $(c = 0)$: $B = g^b$
		if $(c = 1)$: $B = Ag^b$
	$\xleftarrow{B}$	
$k_0 = H(B^a)$		$k_R = H(A^b)$
$k_1 = H((\frac{B}{A})^a)$		
	$e_0 \leftarrow E_{k_0}(m_0)$	
	$e_1 \leftarrow E_{k_1}(m_1)$	
	$\longrightarrow$	
		$m_c = D_{k_R}(e_c)$

Figure A.1: Chosen one-out-of-two Oblivious Transfer protocol implementation [CO15], based on the Diffie-Hellman key-exchange protocol.

The privacy properties and security guarantees for the previous OT protocol implementation are intrinsically dependent on the difficulty of solving the computational Diffie-Hellman (CDH) problem. As long as there is no efficient solution for the CDH problem, the previous protocol is ensured to hold the following conditions:

- **Sender's privacy.** The receiver cannot learn the message that does not correspond to its selection bit.
- **Receiver's privacy.** The sender cannot learn the receiver's selection bit.

Appendix B

Garbled Circuit construction

The following addendum details the Yao's Garbled Circuits protocol, a universal MPC solution in the presence of semi-honest adversaries. The protocol is divided in two operative phases:

- **GC generation phase.** Given a Boolean circuit $\mathcal{C}$ for a functionality $\mathcal{F}$, A transforms it into a garbled circuit version $\mathcal{C}_g$. This phase is entirely computed by A.
- **GC evaluation phase.** A sends the garbled circuit $\mathcal{C}_g$ to B. B proceeds to sequentially evaluate each gate (for gates where B has an input) by partaking in an OT protocol with A. At the end, the only significant new information B *learns* is the output bit for $\mathcal{C}_g$, evaluated on the private inputs of A and B.

For each phase, a step-by-step procedure and graphical examples are provided; based on [Yak17], [Rom17], [CKW18], and [CB19]. For simplicity, none of the various optimization proposals are used.

B.1 GC generation

Single-gate view

The following example develops the *garbling* process for an AND gate. Nonetheless, any gate may be implemented by replacing the values of the output column z.

Example B.1 (Standalone AND gate generation)**.**

Setting. The possible output values (keys) for the gate are considered: $\{k^0_{out}, k^1_{out}\}$.

Step 1. A samples a pair of one-time keys for each input wire in g; $\{k^0_A, k^1_A\}$ and $\{k^0_B, k^1_B\}$ respectively.

The wires are associated with the related input/output key (figure B.1).

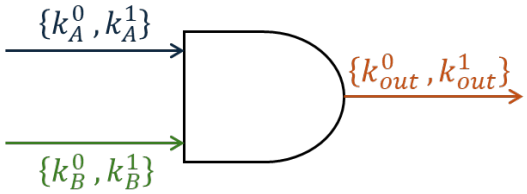

Figure B.1: Standalone AND gate. Each wire is labeled with the corresponding key pair.

Step 2. A transforms the truth table of g with the corresponding keys and encrypts each output with the respective inputs for that row.

x	y	z		x	y	z		Encrypted row
0	0	0		k^0_A	k^0_B	k^0_{out}		$E_{k^0_A}(E_{k^0_B}(k^0_{out}))$
0	1	0	$\Longrightarrow$	k^0_A	k^1_B	k^0_{out}	$\Longrightarrow$	$E_{k^0_A}(E_{k^1_B}(k^0_{out}))$
1	0	0		k^1_A	k^0_B	k^0_{out}		$E_{k^1_A}(E_{k^0_B}(k^0_{out}))$
1	1	1		k^1_A	k^1_B	k^1_{out}		$E_{k^1_A}(E_{k^1_B}(k^1_{out}))$

Step 3. A randomly permutes the encrypted rows.

The resulting 4 messages are the unit denominated *garbled gate*.

Circuit view

Whenever the output of a gate connects to the input of another gate, A does not sample a new key pair, instead, the output key pair becomes the input key pair for that wire.

The circuit-*garbling* protocol is a sequential generation of garbled gates.

The last gate to be evaluated may replace the output keys $\{k^0_{result}, k^1_{result}\}$ for the plain-text values $\{0, 1\}$ without risk of information leakage. Otherwise, the message pair $\{E_{k^0_{result}}(0), E_{k^1_{result}}(1)\}$ must be provided to B.

B.2 GC evaluation

Single-gate view

Protocol B.1 (Standalone garbled gate evaluation with A, B participation)**.**

Setting. A's input is $x \in \{0, 1\}$.

B's input is $y \in \{0, 1\}$.

The logic gate associated to the garbled gate computes the functionality $\mathcal{F}(x, y) = z$.

Step 1. A sends B the key that corresponds to its own input x, namely k_A^x.

Step 2. A and B engage in a $\binom{2}{1}$OT:

A acts as sender, with input messages $\{m_0 = k_B^0, m_1 = k_B^1\}$.

B acts as receiver, with input selection bit $c = y$.

B obtains k_B^y.

Step 3. B uses the pair of keys $\{k_A^x, k_B^y\}$ to *try to* decrypt all the rows in the garbled gate. One and only one row will produce a significant output, namely k_{out}^z.

Circuit view

The gates to be evaluated are categorized according to the origin of the data expected on each input wire: G labels a connection to a previously evaluated gate; A, B label which participant's input is utilized.

A*, *A* | *B*, *B. These gates can be evaluated without interaction. Given the Boolean circuit is of common knowledge to all parties, this results can be precomputed prior to execution.

A*, *B. This evaluation follows the standalone garbled gate evaluation protocol B.1 previously explored.

A*, *G. A's side is provided to B prior to evaluation; G's output key is directly used to decrypt. No OT protocol utilized.

B*, *G. B obtains the key corresponding to its input from an OT protocol execution; G's output key is directly used to decrypt.

G*, *G. Both keys are directly used to decrypt.

Appendix C

3-party Garbled Circuits

The following addendum provides a throughout exemplification of a complex multi-gated circuit.

C.1 3GC setting example

R_0's input is $x_a = 0, x_b = 1, x_d = 1$.

R_1's input is $y_a = 1, y_b = 1, y_c = 0$.

Key size n.

The logic circuit computes the functionality $\mathcal{F}(x_a, x_b, x_d, y_a, y_b, y_c) = C_{out}$.

$$C_{out} = ((x_a \wedge y_a) \vee y_c) \oplus ((x_b \vee y_b) \wedge x_d) \tag{C.1}$$

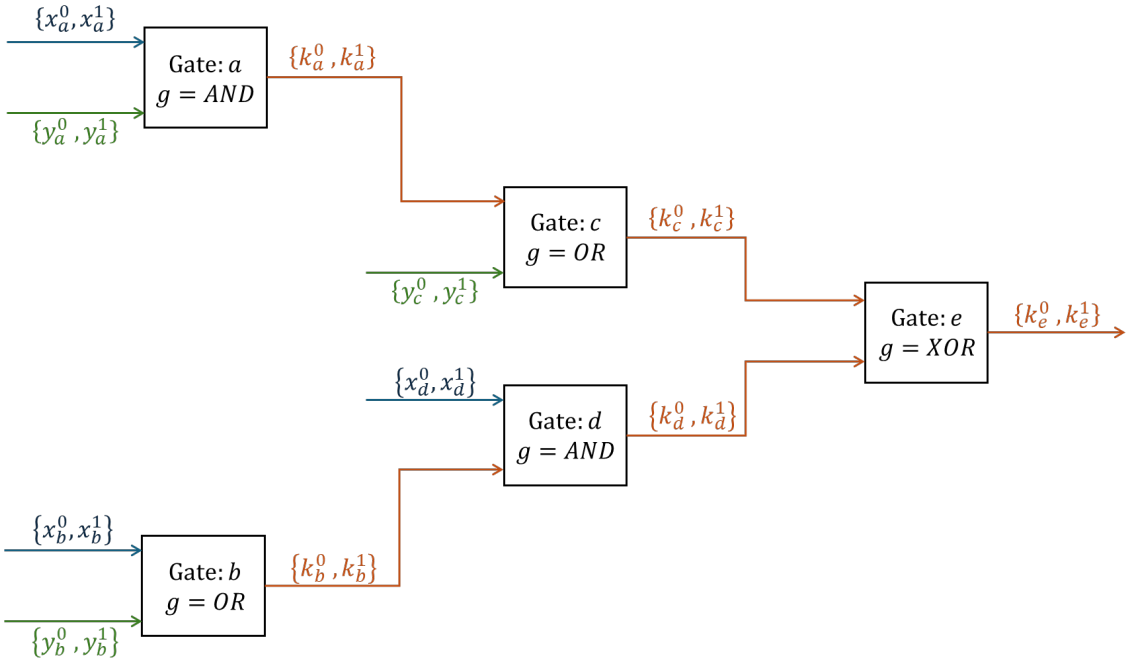

Figure C.1: Multi-gated circuit example.

C.2 3GC generation example

Example C.1 (3-party garbled multi-gated circuit generation)**.**

Pre-protocol phase

S sends the encrypted circuit output message pair $\{E_{k_e^0}(0), E_{k_e^1}(1)\}$ to the evaluator R_0

Gate *a*. (*No optimization*).

Step 1. S samples 2 key pairs: $\{k^0_{a,R_0}, k^1_{a,R_0}\}, \{k^0_{a,R_1}, k^1_{a,R_1}\}$.

S sends $\{k^0_{a,R_1}, k^1_{a,R_1}\}$ to R_1.

Step 2. S constructs the truth table for $(x \wedge y)$.

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

Step 3. S substitutes the output values in the table to the key-encrypted version.

x	y	z		Encrypted rows
k^0_{a,R_0}	k^0_{a,R_1}	k^0_a		$E_{k^0_{a,R_0}}(E_{k^0_{a,R_1}}(k^0_a))$
k^0_{a,R_0}	k^1_{a,R_1}	k^0_a	$\Longrightarrow$	$E_{k^0_{a,R_0}}(E_{k^1_{a,R_1}}(k^0_a))$
k^1_{a,R_0}	k^0_{a,R_1}	k^0_a		$E_{k^1_{a,R_0}}(E_{k^0_{a,R_1}}(k^0_a))$
k^1_{a,R_0}	k^1_{a,R_1}	k^1_a		$E_{k^1_{a,R_0}}(E_{k^1_{a,R_1}}(k^1_a))$

Step 4. S randomly permutes the encrypted rows. The resulting garbled gate is sent to R_0.

Gate *b*. (*Full participation message reduction*, section 4.9.2).

No previous garbled gate generation is needed.

Gate *c*.

Step 1. S samples 1 key pair: $\{k^0_{c,R_1}, k^1_{c,R_1}\}$.

S sends $\{k^0_{c,R_1}, k^1_{c,R_1}\}$ to R_1.

Step 2. S constructs the truth table for $x \vee y$.

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

Step 3. S substitutes the output values in the table to the key-encrypted version.

x	y	z		Encrypted rows
k^0_a	k^0_{c,R_1}	k^0_c		$E_{k^0_a}(E_{k^0_{c,R_1}}(k^0_c))$
k^0_a	k^1_{c,R_1}	k^1_c	$\Longrightarrow$	$E_{k^0_a}(E_{k^1_{c,R_1}}(k^1_c))$
k^1_a	k^0_{c,R_1}	k^1_c		$E_{k^1_a}(E_{k^0_{c,R_1}}(k^1_c))$
k^1_a	k^1_{c,R_1}	k^1_c		$E_{k^1_a}(E_{k^1_{c,R_1}}(k^1_c))$

Step 4. S randomly permutes the encrypted rows. The resulting garbled gate is sent to R_0.

Gate *d*.

Step 1. S samples 1 key pair $\{k^0_{d,R_0}, k^1_{d,R_0}\}$.

Step 2. S constructs the truth table for $x \wedge y$.

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

Step 3. S substitutes the output values in the table to the key-encrypted version.

x	y	z
k^0_{d,R_0}	k^0_b	k^0_d
k^0_{d,R_0}	k^1_b	k^0_d
k^1_{d,R_0}	k^0_b	k^0_d
k^1_{d,R_0}	k^1_b	k^1_d

$\Longrightarrow$

Encrypted rows
$E_{k^0_{d,R_0}}(E_{k^0_b}(k^0_d))$
$E_{k^0_{d,R_0}}(E_{k^1_b}(k^0_d))$
$E_{k^1_{d,R_0}}(E_{k^0_b}(k^0_d))$
$E_{k^1_{d,R_0}}(E_{k^1_b}(k^1_d))$

Step 4. S randomly permutes the encrypted rows. The resulting garbled gate is sent to R_0.

Gate *e*.

Step 1. No key sampling is required.

S constructs the truth table for $x \oplus y$.

x	y	z
0	0	0
0	1	1
1	0	1
1	1	0

Step 3. S substitutes the output values in the table to the key-encrypted version.

x	y	z
k^0_c	k^0_d	k^0_e
k^0_c	k^1_d	k^1_e
k^1_c	k^0_d	k^1_e
k^1_c	k^1_d	k^0_e

$\Longrightarrow$

Encrypted rows
$E_{k^0_c}(E_{k^0_d}(k^0_e))$
$E_{k^0_c}(E_{k^1_d}(k^1_e))$
$E_{k^1_c}(E_{k^0_d}(k^1_e))$
$E_{k^1_c}(E_{k^1_d}(k^0_e))$

Step 4. S randomly permutes the encrypted rows. The resulting garbled gate is sent to R_0.

C.3 3GC evaluation example

Example C.2 (3-party garbled multi-gated circuit evaluation)**.**

Gate *a*. (*No optimization*).

Step 1. The participants engage in a $(2,2)\text{OD}_{\text{XOR}}$ execution:

S inputs the messages: $\{m_0 = k^0_{a,R_0}, m_1 = k^1_{a,R_0}\}$.

Given that $x_a = 0$; R_0 inputs 0.

By protocol prescription; R_1 inputs 0.

$z = 0 \oplus 0$

R_0 receives $m_{z=0} = k^0_{a,R_0}$.

R_1 receives $m_{\overline{z}=1} = k^1_{a,R_0}$.

Step 2. Given that $y_a = 1$; R_1 sends k^1_{a,R_1} to R_0.

Step 3. R_0 uses k^0_{a,R_0} and k^1_{a,R_1} to decrypt the garbled circuit.

R_0 obtains k^0_a, from the only correctly decrypted row $E_{k^0_{a,R_0}}(E_{k^1_{a,R_1}}(k^0_a))$.

Gate *b*. (*Full participation message reduction*, section 4.9.2).

The participants engage in a $(2,2)\text{OD}_{\text{OR}}$ execution:

S inputs the messages: $\{m_0 = k^0_b, m_1 = k^1_b\}$.

Given that $x_b = 1$; R_0 inputs the bit 1.

Given that $y_b = 1$; R_1 inputs the bit 1.

$z = 1 \vee 1$, R_0 obtains k^1_b.

$\overline{z} = 0$, R_1 obtains k^0_b.

Gate *c*.

Evaluator R_0 has no input. No OD is conducted.

Step 1. Given that $y_c = 0$; R_1 sends k^0_{c,R_1} to R_0.

Step 2. R_0 uses k^0_a (Gate a's output) and k^0_{c,R_1} to decrypt the garbled circuit.

R_0 obtains k^0_c, from the only correctly decrypted row $E_{k^0_a}(E_{k^0_{c,R_1}}(k^0_c))$.

Gate *d*.

Participant R_1 has no input.

Step 1. The participants engage in a $(2,2)\text{OD}_{\text{XOR}}$ execution:

S inputs the messages: $\{m_0 = k^0_{d,R_0}, m_1 = k^1_{d,R_0}\}$.

Given that $x_d = 1$; R_0 inputs 1.

By protocol prescription; R_1 inputs 0.

$z = 1 \oplus 0$

R_0 receives $m_{z=1} = k^1_{d,R_0}$.

R_1 receives $m_{\overline{z}=0} = k^0_{a,R_0}$.

Step 2. R_0 uses k^1_{d,R_0} and k^1_b (Gate b's output) to decrypt the garbled circuit.

R_0 obtains k_d^1, from the only correctly decrypted row $E_{k^1_{d,R_0}}(E_{k_b^1}(k_d^1))$.

Gate *e*.

Participants R_0, R_1 have no input. No OD is conducted.

R_0 uses k_c^0 (Gate c's output), and k_d^1 (Gate d's output) to decrypt the garbled circuit.

R_0 obtains k_e^1, from the only correctly decrypted row $E_{k_c^1}(E_{k_d^0}(k_e^1))$.

Post-protocol phase

R_0 uses k_e^1 to decrypt the tuple $\{E_{k_e^0}(0), E_{k_e^1}(1)\}$, obtaining the circuit output bit 1.

R_0 publishes the output bit to all other participants.

References for appendices

[DH76] W. Diffie and M. Hellman. "New directions in cryptography". In: *IEEE Transactions on Information Theory* 22.6 (Nov. 1976), pp. 644–654. DOI: 10.1109/tit.1976.1055638.

[CO15] T. Chou and C. Orlandi. "The Simplest Protocol for Oblivious Transfer". In: *Progress in Cryptology – LATINCRYPT 2015*. Ed. by K. Lauter and F. Rodríguez-Henríquez. Cham: Springer International Publishing, 2015, pp. 40–58. ISBN: 978-3-319-22174-8.

[Rom17] Y. Romailler. *Yao's Garbled Circuits and how to construct those*. June 9, 2017. URL: https://romailler.ch/2017/06/09/crypto-garbling_circuits/ (visited on 08/01/2024).

[Yak17] S. Yakoubov. "A Gentle Introduction to Yao's Garbled Circuits". In: Boston Univeristy. Massachusetts Institute of Technology, 2017.

[CKW18] H. Corrigan-Gibbs, S. Kim, and D. J. Wu. "Lecture 6: Oblivious Transfer and Yao's Garbled Circuits". Apr. 2018.

[CB19] J. I. Choi and K. R. B. Butler. "Secure Multiparty Computation and Trusted Hardware: Examining Adoption Challenges and Opportunities". In: *Security and Communication Networks* 2019 (Apr. 2019), pp. 1–28. DOI: 10.1155/2019/1368905.