EFFICIENT AND SECURE EQUALITY-BASED TWO-PARTY COMPUTATION

by

Javad Darivandpour

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

Department of Computer Science

West Lafayette, Indiana

August 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

I dedicate this dissertation to my mother and father
for their endless love, support and encouragement.

ACKNOWLEDGMENTS

It gives me great pleasure to express my gratitude to many people who contributed, one way or another, to me achieving my goals.

First and foremost, I would like to thank my family, my lovely parents and my dear sister, for encouraging me every step of the way. I would not have been able to pursue my dreams if it was not for your full support, unconditional care, and everlasting encouragement. I would also like to thank my dear aunt, Ferial, for always being there for me.

My most sincere gratitude goes to my academic advisor, Professor Mikhail J. Atallah, for his invaluable guidance and genuine support. It is my absolute honor and pleasure to be given the opportunity of working with such an inspiring mentor who taught me how to improve myself in both professional and personal aspects of life. You taught me how to organize my ideas, pay attention to every detail, and treat people with the utmost respect. I thank you with all my heart for creating such a healthy and nurturing environment for me to grow and prosper.

Shortly after, I would like to thank my advisory committee members, Professor Greg N. Frederickson, Professor Samuel S. Wagstaff, and Professor Aniket Kate, for their valuable support and mentorship. Moreover, I am forever grateful and indebted to Professor Susanne E. Hambrusch, Professor Petros Drineas, and Professor Elena Grigorescu for their amazing mentorship and unconditional support. Furthermore, I thank my collaborator and dear friend Duc V. Le for his valuable help in Chapter 5.

Last but not least, I am greatly thankful to my friends, who never stopped energizing me. I would like to thank Poolad Imany, Babak Ravandi, Amir Sadeghi, Faezeh Ravazdezh, Samaneh Saadat, Mahsa Fardisi, Peyman Yousefi, Siamak Rabienia, Ali Hekmatfar, Mohsen Minaei, Hadi Shagerdi, Farzin Shamloo, Mohammad Chegini, Yasaman Taebi, Elham Mohammadrezaei, Hamed Saiedi, Nima Johari, and Ahmad Darki.

TABLE OF CONTENTS

LIST OF TABLES 8

LIST OF FIGURES 9

LIST OF PROTOCOLS 10

LIST OF SYMBOLS 11

ABBREVIATIONS 12

ABSTRACT 13

1 INTRODUCTION 14
- 1.1 Secure Two-party Computation 15
- 1.2 Functionality Formulation 15
- 1.3 Function Representation 16
- 1.4 Summary of Our Contributions 17
  - 1.4.1 Private Equality Testing 18
  - 1.4.2 Secure Wildcard Pattern Matching 18
  - 1.4.3 Secure Two-party Input-size Reduction 19
- 1.5 Organization of the Thesis 20

2 PRELIMINARIES 21
- 2.1 General MPC Threat Models 21
  - 2.1.1 Semi-honest Adversary 21
  - 2.1.2 Malicious Adversary 21
  - 2.1.3 Augmented Semi-honest Adversary 22
- 2.2 Additive Secret Sharing (Secret Splitting) 22
  - 2.2.1 Properties of Secret Splitting 24
  - 2.2.2 Basic Operations and Arithmetic Black-box Model 24
- 2.3 Pseudorandom Function (PRF) 27
- 2.4 Convolution 28

3 PRIVATE EQUALITY TESTING 32
- 3.1 Summary of Contributions 33
- 3.2 Related Work 34
- 3.3 Secure and Lightweight Protocol for Computing $\mathcal{F}_{\text{EQZ}}(\llbracket s \rrbracket^{\mathcal{A}}, \llbracket s \rrbracket^{\mathcal{B}})$ 35
  - Phase 1: Evaluating the Equality Predicate Using Arithmetic Over $\mathbb{F}_3$ 36
  - Phase 2: Modular Conversion in Split Form 37
- 3.4 Overall Performance 41

4 SECURE WILDCARD PATTERN MATCHING 42
- 4.1 Problem Definition 43
- 4.2 Related work 44
- 4.3 Summary of Contributions 45
  - 4.3.1 Wildcard Pattern Matching Algorithm 45

4.3.2 Secure Protocols . . . 46
4.3.3 Experimental Results . . . 46
4.3.4 Functionality Generalization . . . 46
4.4 Convolution-based Wildcard Pattern Matching Algorithm . . . 47
4.4.1 Alphabet Preprocessing . . . 48
4.4.2 Proposed Algorithm (Wildcards in Only **P**) . . . 52
4.4.3 Proposed Algorithm (Wildcards in Both **P** and **T**) . . . 54
4.5 Secure Two-party Protocols for Wildcard Pattern Matching . . . 56
4.5.1 Secure Convolution Computation . . . 57
4.5.2 Secure SWPM Protocols Using $\mathcal{F}_{\text{EQZ}}$ . . . 59
Secure Protocol for $(\bot, \Gamma) = \mathcal{F}^{S}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ . . . 59
Secure Protocol for $(\bot, \gamma) = \mathcal{F}^{C}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ . . . 61
Secure Protocol for $(\bot, (\gamma > 0)) = \mathcal{F}^{D}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ . . . 63
4.5.3 Secure SWPM Protocols Using Relaxed Equality Testing . . . 65
Secure Protocol for $(\bot, \Gamma) = \mathcal{F}^{S}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ . . . 66
Secure Protocol for $(\bot, \gamma) = \mathcal{F}^{C}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ . . . 68
Secure Protocol for $(\bot, (\gamma > 0)) = \mathcal{F}^{D}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ . . . 74
4.6 Experimental Results . . . 78
4.6.1 Performance: Execution Time & Transferred Data . . . 78
4.6.2 Comparison with Previous State of the Art . . . 80
4.7 Discussion: SWPM in Stronger Adversarial Models . . . 81
4.7.1 Bob Learns Only the Leftmost Match . . . 83
4.7.2 Bob Learns One Match at Random . . . 84

5 SECURE TWO-PARTY INPUT-SIZE REDUCTION . . . 86
5.1 Motivation and Overview . . . 86
5.2 Related Work . . . 88
5.3 Summary of Contributions . . . 89
5.3.1 Problem Formulation . . . 89
5.3.2 Proposed Constructions . . . 90
5.3.3 Use Cases and Implications . . . 91
5.4 Problem Definition . . . 91
5.5 Major Challenges . . . 93
5.6 Constructing a Minimal Perfect SPHF for $S = S^{\mathcal{A}} \cup S^{\mathcal{B}}$ . . . 93
5.6.1 FindSPHF: Randomized Label-then-Unify . . . 97
5.6.2 FindSPHF: Randomized Merge-then-Unify . . . 100
5.7 Distribution: Probabilistic Input Partitioning . . . 104
5.7.1 Challenge 1: Preventing Exploitation of Distributor Function's Structure 105
5.7.2 Challenge 2: Possible Failure of a Distributor Candidate and Its Consequences . . . 109
5.7.3 FindDistributor: Putting Pieces Together to Find a Valid Distributor 112
5.8 Overall Distribution-Resolution Scheme . . . 115
5.9 Discussion: Use Cases and Implications in 2PC . . . 118
5.9.1 Faster General Template Matching . . . 118
5.9.2 Search Queries in High-precision Scientific Data . . . 119

5.9.3 Implications of Secure Perfect Hashing for Private Set Intersection . . 120
6 SUMMARY 123
REFERENCES 126

LIST OF TABLES

3.1 Comparison of the secure realizations of $\mathcal{F}_{\text{EQZ}}$ functionality in the ABB model of computation and semi-honest threat model. $\ell = \lceil \log_2 q \rceil$ denotes the input's bit-length, and κ is a correctness parameter. 34

4.1 All times are measured in seconds, and all transferred data volumes are measured in megabytes. This table demonstrates LiLiP's performance for pattern queries of 64 characters, i.e., $|\mathbf{P}| = m = 64$. 79

4.2 All times are measured in seconds, and all transferred data volumes are measured in megabytes. This table demonstrates LiLiP's performance for pattern queries of 128 characters, i.e., $|\mathbf{P}| = m = 128$. 79

4.3 All times are measured in seconds, and all transferred data volumes are measured in megabytes. This table demonstrates LiLiP's performance for pattern queries of 512 characters, i.e., $|\mathbf{P}| = m = 512$. 80

5.1 Performance of proposed SPHF constructions: Let $m_{\mathcal{A}} = |S^{\mathcal{A}}|, m_{\mathcal{B}} = |S^{\mathcal{B}}|$, $m = m_{\mathcal{A}} + m_{\mathcal{B}}$ and the security-performance parameter κ be a small integer; as m grows larger, a smaller κ suffices for the same level of security (see Section 5.8 for choice of κ). 91

LIST OF FIGURES

3.1 $(\llbracket e \rrbracket^{\mathcal{A}}, \llbracket e \rrbracket^{\mathcal{B}}) = \mathcal{F}_{\text{PET}}(s_1, s_2)$ such that $\llbracket e \rrbracket = 1$ if and only if $s_1 = s_2$. 32

3.2 $(\llbracket e \rrbracket^{\mathcal{A}}, \llbracket e \rrbracket^{\mathcal{B}}) = \mathcal{F}_{\text{EQZ}}(\llbracket s \rrbracket^{\mathcal{A}}, \llbracket s \rrbracket^{\mathcal{B}})$ such that $\llbracket e \rrbracket = 1$ if and only if $s = 0$. 32

3.3 Modular conversion encodings and potential combinations: Alice and Bob each locally encodes its respective share of $\llbracket e^{(3)} \rrbracket$ into two bits a_1a_0 and b_1b_0 such that $a_1a_0 \oplus b_1b_0 = 00$ if and only if $\llbracket e^3 \rrbracket = 0$. 37

4.1 $(\perp, \Gamma) = \mathcal{F}^{S}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ 44

4.2 $(\perp, \gamma) = \mathcal{F}^{C}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ 44

4.3 $(\perp, (\gamma > 0)) = \mathcal{F}^{D}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ 44

4.4 Secure two-party circular convolution functionality 57

4.5 $(\llbracket e \rrbracket^{\mathcal{A}}, \llbracket e \rrbracket^{\mathcal{B}}) = \mathcal{F}_{\text{REQZ}}(\llbracket s \rrbracket^{\mathcal{A}}, \llbracket s \rrbracket^{\mathcal{B}})$ 65

4.6 Single-blind Permutation Functionality 69

4.7 $(\perp, \tau(\Gamma)) = \mathcal{F}^{\tau}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ 83

5.1 $(\varrho^{\mathcal{A}}, \varrho^{\mathcal{B}}) = \mathcal{F}_{\text{SPHF}}(S^{\mathcal{A}}, S^{\mathcal{B}})$ 92

5.2 High-level illustration of FindSPHF through an example ($\gamma = 4$). Sections 5.6.1 and 5.6.2 propose protocols that securely and efficiently apply this approach. . . . 94

5.3 The hypothetical function ϕ used in the proof of Lemma 5.1 95

5.4 $(\llbracket \mathbf{Y} \rrbracket^{\mathcal{A}}, \llbracket \mathbf{Y}^{\mathcal{B}} \rrbracket) = \mathcal{F}_{\text{DBP}}(\llbracket \mathbf{X} \rrbracket^{\mathcal{A}}, \llbracket \mathbf{X} \rrbracket^{\mathcal{B}})$ such that $\llbracket \mathbf{Y} \rrbracket = \llbracket \pi(\mathbf{X}) \rrbracket$ for a random permutation π that is unknown to both parties. 96

5.5 $(\llbracket \mathbf{Z} \rrbracket^{\mathcal{A}}, \llbracket \mathbf{Z} \rrbracket^{\mathcal{B}}) = \mathcal{F}_{\text{MERGE}}\Big((\llbracket \mathbf{X} \rrbracket^{\mathcal{A}}, \llbracket \mathbf{Y} \rrbracket^{\mathcal{A}}), (\llbracket \mathbf{X} \rrbracket^{\mathcal{B}}, \llbracket \mathbf{Y} \rrbracket^{\mathcal{B}})\Big)$, which takes sorted vectors $\llbracket \mathbf{X} \rrbracket$ and $\llbracket \mathbf{Y} \rrbracket$ as input, and outputs their sorted combination as vector $\llbracket \mathbf{Z} \rrbracket$ in split form. 101

5.6 Distribution functionality $(\psi^{\mathcal{A}}, \psi^{\mathcal{B}}) = \mathcal{F}_{\text{DIST}}(S^{\mathcal{A}}, S^{\mathcal{B}})$, where (i) Ψ is cooperatively and securely computable; and, (ii) the maximum load bins for $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$ under Ψ have at most γ items, i.e., $|S_i^{\mathcal{A}}|, |S_i^{\mathcal{B}}| \leq \gamma$ for $0 \leq i < m$. 106

5.7 "Pick-any-One" functionality $(t, t) = \mathcal{F}_{\text{PA1}}(\llbracket \mathbf{D} \rrbracket^{\mathcal{A}}, \llbracket \mathbf{D} \rrbracket^{\mathcal{B}})$: The input $\llbracket \mathbf{D} \rrbracket$ is a length-κ vector of 0s and 1s. If $\{i \mid d_i = 1\} = \emptyset$, then $t = 0$; or else, $t \xleftarrow{\$} \{i \mid d_i = 1\}$. 110

5.8 Upper bound $p_{\mathsf{fail}} \leq m \cdot (e/\gamma)^{\gamma}$ for various choices of γ as a function of m 116

5.9 Overview of the Distribution-Resolution approach for the SPHF construction. Black-box reductions in the same level denote independent secure computations in parallel. 117

5.10 Hamming similarity of $\mathbf{X}_{i,j}$ and $\mathbf{T}$ 119

LIST OF PROTOCOLS

3.1 Secure realization of $(\llbracket e \rrbracket^{\mathcal{A}}, \llbracket e \rrbracket^{\mathcal{B}}) = \mathcal{F}_{\mathrm{EQZ}}(\llbracket s \rrbracket^{\mathcal{A}}, \llbracket s \rrbracket^{\mathcal{B}})$ using modular conversion . 40

4.1 Secure realization of $(\llbracket \mathbf{Z}^{\mathcal{A}} \rrbracket, \llbracket \mathbf{Z} \rrbracket^{\mathcal{B}}) = \mathcal{F}_{\mathrm{CONV}}(\mathbf{X}, \mathbf{Y})$ 58

4.2 Secure realization of $(\perp, \Gamma) = \mathcal{F}^{S}_{\mathrm{SWPM}}(\mathbf{T}, \mathbf{P})$ based on $\mathcal{F}_{\mathrm{EQZ}}$ 60

4.3 Secure realization of $(\perp, \gamma) = \mathcal{F}^{C}_{\mathrm{SWPM}}(\mathbf{T}, \mathbf{P})$ based on $\mathcal{F}_{\mathrm{EQZ}}$ 62

4.4 Secure realization of $(\perp, (\gamma > 0)) = \mathcal{F}^{D}_{\mathrm{SWPM}}(\mathbf{T}, \mathbf{P})$ based on $\mathcal{F}_{\mathrm{EQZ}}$ 64

4.5 Secure realization of $(\llbracket e \rrbracket^{\mathcal{A}}, \llbracket e \rrbracket^{\mathcal{B}}) = \mathcal{F}_{\mathrm{REQZ}}(\llbracket s \rrbracket^{\mathcal{A}}, \llbracket s \rrbracket^{\mathcal{B}})$ 65

4.6 Secure realization of $(\perp, \Gamma) = \mathcal{F}^{S}_{\mathrm{SWPM}}(\mathbf{T}, \mathbf{P})$ based on $\mathcal{F}_{\mathrm{REQZ}}$ 67

4.7 Secure realization of $(\llbracket \mathbf{Z} \rrbracket^{\mathcal{A}}, \llbracket \mathbf{Z} \rrbracket^{\mathcal{A}}) = \mathcal{F}_{\mathrm{SBP}}(\pi, \mathbf{X})$ using precomputed auxiliary inputs $\mathbf{R}^{\mathcal{A}}$, $\mathbf{R}^{\mathcal{B}}$ and $\mathbf{U}^{\mathcal{B}} = \pi(\mathbf{R}^{\mathcal{A}} - \mathbf{R}^{\mathcal{B}})$ 70

4.8 Secure realization of $(\perp, \gamma) = \mathcal{F}^{C}_{\mathrm{SWPM}}(\mathbf{T}, \mathbf{P})$ based on $\mathcal{F}_{\mathrm{REQZ}}$ 73

4.9 Secure construction of the augmented vector $\mathbf{E}'$ by obliviously injecting chaff and clone entries 76

4.10 Secure approximation of $(\perp, (\gamma > 0)) = \mathcal{F}^{D}_{\mathrm{SWPM}}(\mathbf{T}, \mathbf{P})$ based on $\mathcal{F}_{\mathrm{REQZ}}$ 77

4.11 Secure realization of $(\perp, \tau_{\min}(\Gamma)) = \mathcal{F}^{\tau_{\min}}_{\mathrm{SWPM}}(\mathbf{T}, \mathbf{P})$ based on $\mathcal{F}_{\mathrm{EQZ}}$ 85

5.1 Secure realization of FindSPHF approach in constant rounds 99

5.2 Construction of ODPRF^{2}_{2} 107

5.3 Secure realization of $(t, t) = \mathcal{F}_{\mathrm{PA1}}(\llbracket \mathbf{D} \rrbracket^{\mathcal{A}}, \llbracket \mathbf{D} \rrbracket^{\mathcal{B}})$ 111

5.4 Finding a valid distributor function. Note that Step 1 is equivalent to κ invocations of $\textsc{Gen}(1^{\tau})$ from Protocol 5.2. 114

5.5 Private Set Intersection built on Secure Perfect Hashing 121

LIST OF SYMBOLS

$\cdot$	Scalar product
$*$	Convolution operator
$\star$	Wildcard symbol
$\odot$	Pointwise product of two vectors
$\oplus$	Bitwise exclusive-OR
$\perp$	Null
$[\![s]\!]$	An additively shared secret s
Superscript $\mathcal{A}$	Alice's private ownership
Superscript $\mathcal{B}$	Bob's private ownership
$\mathbf{X}$	A vector
x_i	The i^{th} entry in vector $\mathbf{X}$
$\mathbf{X}^{rev}$	The reverse of vector $\mathbf{X}$
$\mathbf{X}\|\|\mathbf{Y}$	The concatenation of vectors $\mathbf{X}$ and $\mathbf{Y}$
$r \overset{\$}{\leftarrow} S$	Choosing an item r uniformly at random from the set S

ABBREVIATIONS

Aux.	Auxiliary
a.k.a	also know as
w.p.	with probability
w.h.p.	with high probability
w.l.o.g.	without lost of generality
Inj.	Injective
Mult.	Multiplication
MPC	Secure Multiparty Computation
2PC	Secure Two-party Computation
PET	Private Equality Testing
SWPM	Secure Wildcard Pattern Matching
PHF	Perfect Hash Function
SPHF	Secure Perfect Hash Function
PRF	Pseudorandom Function
OPRF	Oblivious Pseudorandom Function
DPRF	Distributed Pseudorandom Function
ODPRF	Oblivious Distributed Pseudorandom Function

ABSTRACT

Multiparty computation refers to a scenario in which multiple distinct yet connected parties aim to jointly compute a functionality. Over recent decades, with the rapid spread of the internet and digital technologies, multiparty computation has become an increasingly important topic. In addition to the integrity of computation in such scenarios, it is essential to ensure that the privacy of sensitive information is not violated. Thus, secure multiparty computation aims to provide sound approaches for the joint computation of desired functionalities in a secure manner: Not only must the integrity of computation be guaranteed, but also each party must not learn anything about the other parties' private data. In other words, each party learns no more than what can be inferred from its own input and its prescribed output.

This thesis considers secure two-party computation over arithmetic circuits based on additive secret sharing. In particular, we focus on efficient and secure solutions for fundamental functionalities that depend on the equality of private comparands. The first direction we take is providing efficient protocols for two major problems of interest. Specifically, we give novel and efficient solutions for *private equality testing* and multiple variants of *secure wildcard pattern matching* over any arbitrary finite alphabet. These problems are of vital importance: Private equality testing is a basic building block in many secure multiparty protocols; and, secure pattern matching is frequently used in various data-sensitive domains, including (but not limited to) private information retrieval and healthcare-related data analysis. The second direction we take towards a performance improvement in equality-based secure two-party computation is via introducing a generic functionality-independent secure preprocessing that results in an overall computation and communication cost reduction for any subsequent protocol. We achieve this by providing the first precise functionality formulation and secure protocols for replacing original inputs with much smaller inputs such that this replacement neither changes the outcome of subsequent computations nor violates the privacy of sensitive inputs. Moreover, our input-size reduction opens the door to a new approach for efficiently solving Private Set Intersection. The protocols we give in this thesis are typically secure in the semi-honest adversarial threat model.

1. INTRODUCTION

Cryptography (rooted in Greek for "hidden writing") emerged thousands of years ago as a means for communicating secret messages. Its boundaries have expanded vastly over time alongside the emergence of more complex communication and computation needs in human societies. In recent decades, with the rapid development of computing devices, the modern state of computation and communication calls for extremely sophisticated tools to address safety and privacy demands in various domains. Hence, cryptography has become a well-structured ever-developing mathematical science that provides for these needs.

One of the (relatively) recent subfields of cryptography is Secure Multiparty Computation (MPC) that was formally introduced in 1982 by Yao [1]. In general, MPC refers to a setting in which n parties $P_1, P_2, \cdots, P_n$ with respective private inputs $x_1, x_2, \cdots, x_n$ wish to compute a function $f(x_1, x_2, \cdots, x_n)$ while preserving the privacy of their input data as much as possible. Due to its strong security guarantees, MPC has a wide range of applications such as secure general data analysis and data mining [2]–[9], biomedical computations [10]–[14], financial computations [15]–[18], and many more task-specific use cases.

General-purpose MPC protocols that are capable of securely evaluating any function have been known since the 1980s. The first protocol to realize MPC was presented by Yao [1] through the introduction of garbled circuits. This seminal work was followed by other general-purpose MPC solutions for secure evaluation of any function [19]–[21]. Although such protocols were mainly of theoretical interest due to their inefficiencies in computation and communication complexities, they led the cryptographic community into the area of general-purpose MPC compilers (e.g., [22], [23]). The idea behind such compilers is to allow programming in (fairly) high-level languages, then converting them to a circuit format that will be executed using secure computation protocols. This would reduce the burden of designing protocols and allow non-experts to use MPC for secure computations of interest. Building efficient and easy-to-use general-purpose MPC compilers is an ongoing effort [24]. However, the general-purpose MPC solutions tend to be rather inefficient; their generality does not allow exploiting all optimization opportunities specific to a problem of interest. Another course of research towards practically applicable MPC is designing highly-optimized special purpose MPC protocols for specific functions of interest (e.g., [25]–[27]). In this dissertation, we follow this latter paradigm for achieving efficient two-party protocols for problems whose answers depend on equality of input symbols (e.g., pattern matching and private set intersection).

1.1 Secure Two-party Computation

In this dissertation, we focus on Secure Two-party Computation (2PC) as a special case of MPC [28]. In 2PC, two distrusting parties (say, Alice and Bob) wish to jointly compute a function of their private inputs while satisfying two main requirements: i) Each party must receive its correct intended output (*correctness*); and, ii) neither party learns anything other than what can be deduced from its own input and its prescribed output (*privacy*). The most basic 2PC problems are:

1. *Yao's millionaires problem* [1] in which two millionaires intend to determine who is richer without revealing their net worth to each other.

2. *Socialist millionaires problem* in which two millionaires aim to determine whether or not they are as rich as each other. This problem is often referred to as private equality testing.

Despite their basic nature, these 2PC problems have proven to be of vital importance in many computations in the digital world, including applications in online auctions, voting systems, sensitive signal processing, and private search queries, to mention a few.

Extensive research has been dedicated to efficient 2PC solutions for a wide variety of problems (e.g., [27], [29]–[37]). As a result, 2PC has recently become a more practically applicable cryptographic technology [38].

1.2 Functionality Formulation

Privacy in 2PC requires that neither party can deduce any information other than what can be deduced from its own input and prescribed output. Unlike a function, a functionality specifies not only what is computed but also who provides each input and who learns which output. Formally, consider a functionality $\mathcal{F}$ that takes Alice's (resp. Bob's) private input $\mathsf{in}^{\mathcal{A}}$ (resp. $\mathsf{in}^{\mathcal{B}}$), and in return provides her (resp. him) with the output $\mathsf{out}^{\mathcal{A}}$ (resp. $\mathsf{out}^{\mathcal{B}}$); such an ideal functionality is denoted as follows:

$$(\mathsf{out}^{\mathcal{A}}, \mathsf{out}^{\mathcal{B}}) = \mathcal{F}(\mathsf{in}^{\mathcal{A}}, \mathsf{in}^{\mathcal{B}})$$

To clarify, we discuss two distinct 2PC functionality definitions for private equality testing, where private inputs $\mathsf{in}^{\mathcal{A}}$ and $\mathsf{in}^{\mathcal{B}}$ are integers:

1. First, we consider a case in which the functionality discloses the test outcome to Bob but not to Alice. In other words, Bob learns the predicate value $e = (\mathsf{in}^{\mathcal{A}} == \mathsf{in}^{\mathcal{B}})$ while Alice learns nothing; i.e., $\mathsf{out}^{\mathcal{A}} = \perp$ and $\mathsf{out}^{\mathcal{B}} = e$. Note that the output $e = 1$ inherently reveals the integer $\mathsf{in}^{\mathcal{A}}$ to Bob, whereas the output $e = 0$ reveals nothing about $\mathsf{in}^{\mathcal{A}}$ to Bob except the fact that $\mathsf{in}^{\mathcal{A}} \neq \mathsf{in}^{\mathcal{B}}$.

2. Now, we consider another case in which the functionality does not disclose the test output to either party, but returns the predicate value $e = (\mathsf{in}^{\mathcal{A}} == \mathsf{in}^{\mathcal{B}})$ as a shared secret; i.e., neither party's private output alone gives any information about e, but e can be reconstructed using both shares of the output. In this scenario, neither party learns *anything* about the other party's input.

In the remainder of this thesis, we typically require the outputs to be shared secrets, unless we explicitly state otherwise. This allows us to maintain a maximum level of privacy when a protocol itself serves as a building block in another solution, without sacrificing generality because the output can be reconstructed if either party is supposed to learn it in its entirety.

1.3 Function Representation

After choosing a proper formulation based on the desired goals, a natural question is how to mathematically represent functions; there are two primary choices:

- *Boolean circuits* in which a function's inputs and outputs are encoded as bits.

- *Arithmetic circuit* over a prime field in which a function's inputs and outputs are elements in the finite field $\mathbb{F}_q$ of integers modulo a prime q.

The trade-off between these two choices boils down to the impact of the underlying encoding on the efficiency of computing basic building blocks. On the one hand, boolean functions such as equality and inequality tests are easy to compute in the boolean circuit, whereas they are challenging in the arithmetic circuit. On the other hand, the addition and multiplication of secrets can be performed efficiently in the arithmetic circuit but not in the boolean circuit. This thesis aims to provide efficient approaches in the arithmetic circuit when the desired functionality depends on equality tests that involve private comparands. In order to obtain guaranteed privacy while maintaining efficiency, we use additive secret sharing (see Section 2.2).

In 2006, the seminal results of [39] resolved the gap between arithmetic circuits and boolean circuits. Damgård *et al.* [39] proposed, for the first time, a bit-decomposition technique that converts a shared secret integer into the sharings of its bits. This technique was the first bridge between boolean and arithmetic circuits in MPC. Moreover, Schoenmakers *et al.* [40] proposed similar results based on homomorphic cryptosystems that obtain encrypted bits of an integer from the encrypted integer itself.

1.4 Summary of Our Contributions

We take two distinct approaches to improve the performance of equality-based 2PC protocols. First, we present efficient solutions for two problems of interest, namely, Private Equality Testing (Chapter 3) and Secure Wildcard Pattern Matching (Chapter 4). Both of these problems play fundamental roles as building blocks in a wide variety of more complex 2PC problems. Our second approach presented in Chapter 5 aims to improve the performance of equality-based 2PC protocols through a generic secure preprocessing that reduces the inputs' sizes. Most of the results of this thesis have already been published in scholarly venues: Parts of Chapter 4 in [41]; and, Chapter 5 in [42]. Below, we briefly review our contributions.

1.4.1 Private Equality Testing

Both instances of secure comparison of private integers (Yao's millionaires and socialist millionaires problems) are among the most fundamental 2PC problems. These functionalities, alongside the basic addition and multiplication operations, are widely used as building blocks for solving other problems. Here, we concentrate on private equality testing, which is itself vastly studied [39], [43]–[55] (see Section 3.2 for a detailed review).

In private equality testing, Alice's and Bob's respective private inputs are integers a and b, and they would like to securely evaluate the predicate "$(a == b)$". Due to its non-arithmetic nature, private equality testing is a major efficiency bottleneck for many 2PC protocols in the arithmetic circuit model. Although bit-decomposition is sufficient for solving this problem [39], [43], protocols that involve bit-decomposition are rather expensive and should be considered as a last resort. In Chapter 3, we present a novel lightweight protocol that securely evaluates the equality predicate over the arithmetic field $\mathbb{F}_q$. In order to obtain efficiency, all internal states of our protocol are over $\mathbb{F}_3$ rather than $\mathbb{F}_q$; at the end, our protocol uses our novel modular conversion technique, that uses only three secure multiplications over $\mathbb{F}_q$, to convert the secret-shared test result from $\mathbb{F}_3$ to $\mathbb{F}_q$.

1.4.2 Secure Wildcard Pattern Matching

Pattern matching, in its various forms, is one of the primary problems when it comes to data processing and maintenance. Its countless applications include bioinformatics and DNA sequencing, digital forensics, intrusion detection, spelling checking, plagiarism detection, spam filtering, search engines, and content search in large databases. Pattern matching in a 2PC setting has comparatively many applications and is also well-studied [4], [56]–[59]. In Chapter 4, we address the Secure Wildcard Pattern Matching (SWPM) problem, which serves as a building block in numerous 2PC problems.

Let $\Sigma \subseteq \mathbb{F}_q$ be the underlying alphabet. In secure pattern matching, a party (say, Alice) has a private text string $\mathbf{T} \in \Sigma^n$ and the other party (say, Bob) has a private pattern string $\mathbf{P} \in \Sigma^m$, where $m \leq n$; the goal is for Bob to learn only about the occurrences of $\mathbf{P}$ in $\mathbf{T}$ without learning any additional information about $\mathbf{T}$. Meanwhile, Alice must learn nothing about $\mathbf{P}$. Note that secure pattern matching can be interpreted as a generalization of private equality testing (with output disclosure to Bob), which corresponds to the special case of $n = m = 1$. SWPM also allows the use of a wildcard (or, "*don't care*") symbol $\star \notin \Sigma$ that matches any alphabet symbol. There are three major variants of SWPM:

- *Search version*: Bob learns all positions in $\mathbf{T}$ at which $\mathbf{P}$ occurs.
- *Counting version*: Bob learns only the count of occurrences of $\mathbf{P}$ in $\mathbf{T}$.
- *Decision version*: Bob learns only whether or not $\mathbf{P}$ occurs in $\mathbf{T}$ at least once.

Throughout, by *occurrence* of $\mathbf{P}$ in $\mathbf{T}$ we mean as a substring of $\mathbf{T}$. In spite of the extensive work done on secure pattern matching, the bulk of the existing work in the literature is dedicated to the search version. In Chapter 4, we first propose a linearithmic wildcard pattern matching algorithm which is particularly well suited for easy and efficient secure instantiation. Then, we build on it to give provably secure protocols that solve all three variants of SWPM.

Additionally, we point out that the traditional search version of SWPM has a major definitional drawback in the stronger security models where Bob can lie about his input: An adversarial Bob who may use a judiciously crafted input string $\tilde{\mathbf{P}}$ (rather than his prescribed input $\mathbf{P}$) can learn Alice's input $\mathbf{T}$, partially or even in its entirety. This drawback is inherent to the functionality definition, and it defeats the purpose of using *any* secure solution. We address this issue through formulating a generalized SWPM functionality with respect to a *filtering function* that restricts what Bob learns. Moreover, we show that our solutions to the three main variants of the problem fit in this more general definition. Finally, we give two additional instances of filtered SWPM that relate to the search version but do not allow Bob to learn Alice's input text.

1.4.3 Secure Two-party Input-size Reduction

In Chapter 5, we consider a novel approach for improving the performance of equality-based 2PC protocols. In particular, we focus on the obvious fact that the performance of any protocol depends on its inputs' sizes. As a result, a secure preprocessing of inputs that replaces each input symbol (an element in $\mathbb{F}_q$) with a much smaller integer, in a manner that does not change the outcome of subsequent protocols, would reduce the computation and communication costs. In other words, we intend to transfer all computations from $\mathbb{F}_q$ to $\mathbb{F}_{\hat{q}}$ for $\hat{q} \ll q$. Such a preprocessing is especially advantageous when its cost can be amortized over subsequent computations that all benefit from the smaller inputs. There are some task-specific [31], [60] secure size reduction approaches. However, our work is the first attempt towards a generic solution in this domain; e.g., it does not matter if our size-reduction scheme is used before invoking a protocol for secure pattern matching or for private set intersection.

We formalize the 2PC input-size reduction problem, propose efficient and secure solutions to it, and discuss its use cases. In a nutshell, Alice's and Bob's inputs are their respective private sets $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$ of large integers, and their private outputs are images of their sets under a function ρ that injectively maps $S^{\mathcal{A}} \cup S^{\mathcal{B}}$ into $\{0, 1, \cdots, N-1\}$ for a small $N \geq |S^{\mathcal{A}}| + |S^{\mathcal{B}}|$; this allows executing subsequent protocols over $\mathbb{F}_{\hat{q}}$ instead of $\mathbb{F}_q$, where $\hat{q}$ is the smallest prime larger than $N-1$. Alice's (resp. Bob's) knowledge of this mapping on $S^{\mathcal{A}}$ (resp. $S^{\mathcal{B}}$) must reveal nothing about $S^{\mathcal{B}}$ (resp. $S^{\mathcal{A}}$). Thus, neither party should be able to learn $\rho(x)$ for any x that is not in its private set; otherwise, either party could exploit the small codomain of ρ to learn about the other party's set.

1.5 Organization of the Thesis

This dissertation is organized as follows. Chapter 2 reviews the underlying concepts that are used in this document. Chapters 3 and 4 present the first direction that we take for improving equality-based 2PC. In particular, Chapter 3 reviews and gives a new solution to the private equality testing problem, and Chapter 4 is dedicated to the secure wildcard pattern matching problem. Chapter 5 presents our second direction that addresses the need for a generic secure preprocessing of inputs that results in a performance improvement for any subsequent 2PC equality-based computation. Finally, we summarize this dissertation in Chapter 6.

2. PRELIMINARIES

This chapter reviews the existing concepts and primitives used in the remainder of this thesis.

2.1 General MPC Threat Models

Adversarial behavior is one of the major factors in designing secure two-party protocols. Often, 2PC protocols are investigated in the presence of either a semi-honest or a malicious adversary. In a nutshell, the former model considers a corrupted party that behaves honestly but curiously tries to learn about the other party's private data by probing its transcript of the protocol steps. There is no restriction on a malicious adversary's behavior, and it may deviate from the protocol as it desires. In what follows, we discuss these models in detail and review a third model that serves as a bridge between them. In this document, we assume a *static* corruption model in which honest and corrupted parties are fixed in the beginning and remain so during the course of protocol execution.

2.1.1 Semi-honest Adversary

A semi-honest (a.k.a *honest-but-curious* or *passive*) adversary follows the protocol specifications correctly. However, it maintains the transcript of all the messages received, and attempts to use it for learning unauthorized information about the other party's private data. Although this adversarial model may seem insufficient for practical purposes, it is useful in many cases. First, it perfectly models inadvertent leakage of information in protocols' design. Moreover, designing secure protocols in the semi-honest model is often the first step for obtaining secure protocols in the stronger malicious model.

2.1.2 Malicious Adversary

A malicious (a.k.a *active*) adversary can arbitrarily deviate from the protocol specification. In this model, the goal is to catch a cheating adversary with overwhelming probability. Providing security in the presence of malicious adversaries ensures that no adversarial attack can succeed. However, protocols that are secure in this model are usually much less efficient.

One of the common approaches in obtaining security in the malicious model is i) designing a secure protocol in the semi-honest model, and ii) using some cryptographic primitives to enforce a semi-honest behavior on both parties.

2.1.3 Augmented Semi-honest Adversary

By definition, a semi-honest adversary always provides its prescribed input. However, it is only natural to allow a corrupted party to modify its input before the protocol execution begins. This is because of the following two reasons [28]:

- Subjectively, it can be argued that choosing a different input is not improper behavior. Hence, this free choice of input is in the spirit of semi-honest behavior.
- When protocols secure in the semi-honest model are used as a stepping stone for obtaining secure protocols in the malicious model, it is crucial to allow the semi-honest adversary to modify its input. In fact, Goldreich introduces the augmented semi-honest model as above when describing how to obtain security in the malicious model from protocols that are secure only against a semi-honest adversary [61]. On another note, it is only natural that any protocol that is secure in the malicious model must also be secure in the weaker semi-honest model. Hazay *et al.* [28] discuss that this is guaranteed to hold only when the semi-honest adversary can also change its input.

2.2 Additive Secret Sharing (Secret Splitting)

Secret sharing schemes distribute a secret s among n parties such that any subset of the participants that satisfy a certain criterion are able to reconstruct s. In particular, a secret sharing scheme is an (n, t)-threshold scheme if the shared secret s among n parties can be reconstructed by any subset of $t \leq n$ or more participants, and fewer than t parties cannot learn anything about the secret [62]. Such a secret sharing scheme must be able to uniquely determine the secret s (*correctness*); furthermore, any group of less than t parties must not be able to learn anything about s (*privacy*).

Additive secret sharing (a.k.a *secret splitting*) over a finite field $\mathbb{F}_q$ is the simplest form of secret sharing, and it is an (n, n)-threshold scheme. In the two-party case ($n = 2$), a split secret s among two parties, Alice and Bob, consists of two uniformly distributed shares $[\![s]\!]^{\mathcal{A}} \in \mathbb{F}_q$ (Alice's share) and $[\![s]\!]^{\mathcal{B}} \in \mathbb{F}_q$ (Bob's share) such that $s = [\![s]\!]^{\mathcal{A}} + [\![s]\!]^{\mathcal{B}}$. Secret splitting is a valid $(2, 2)$-threshold scheme:

- *Correctness*: Given both shares, s can be reconstructed uniquely through one addition.
- *Privacy*: Each party's share is statistically independent of the secret s; hence, one share alone does not reveal any information about s.

Moreover, a secret s that is private to one party (say, Alice) can be shared among parties as follows:

1. Alice generates a uniform random $r \xleftarrow{\$} \mathbb{F}_q$.
2. Alice sends r to Bob as his share of the secret s; she also computes $s - r$ as her own share of the secret. In summary,

$$[\![s]\!]^{\mathcal{A}} = s - r \qquad \text{and} \qquad [\![s]\!]^{\mathcal{B}} = r \tag{2.1}$$

Clearly, $s = [\![s]\!]^{\mathcal{A}} + [\![s]\!]^{\mathcal{B}}$, and each party's share is a uniform random that is independent of the secret s; moreover, Bob learns nothing about Alice's private secret s because he learns only the uniform random r. Note that in the above special case when one party owns s (hence, knows it), it is not necessary to make the shares uniform randoms that are both independent of s, but it suffices to ensure that the second party's share does not carry any information about the secret. Therefore in such a case, the parties can split s non-interactively as follows:

$$[\![s]\!]^{\mathcal{A}} = s \qquad \text{and} \qquad [\![s]\!]^{\mathcal{B}} = 0 \tag{2.2}$$

Obviously, $s = [\![s]\!]^{\mathcal{A}} + [\![s]\!]^{\mathcal{B}}$, and Bob learns nothing about Alice's secret s because his share $[\![s]\!]^{\mathcal{B}} = 0$ is independent of s and carries no information about it.

Notation 2.1. In the remainder of this document, we use $[\![s]\!]$ to denote a secret s that is additively split among Alice and Bob.

2.2.1 Properties of Secret Splitting

Secret splitting has the following two well-known properties:

1. *Linearity*: For two split secrets $[\![s_1]\!]$ and $[\![s_2]\!]$ and public integers c and d, Alice and Bob can compute $[\![s_3]\!] = [\![c \cdot s_1 + d \cdot s_2]\!]$ without any interaction as follows:

$$[\![s_3]\!]^{\mathcal{A}} = c \cdot [\![s_1]\!]^{\mathcal{A}} + d \cdot [\![s_2]\!]^{\mathcal{A}} \qquad \text{and} \qquad [\![s_3]\!]^{\mathcal{B}} = c \cdot [\![s_1]\!]^{\mathcal{B}} + d \cdot [\![s_2]\!]^{\mathcal{B}} \tag{2.3}$$

   The correctness of this property follows immediately from the definition of secret splitting.

2. *Efficient secure multiplication*: For two split secrets $[\![s_1]\!]$ and $[\![s_2]\!]$, there is a one-round multiplication protocol that allows Alice and Bob to securely compute $[\![s_3]\!] = [\![s_1 \cdot s_2]\!]$. More details about this property will be reviewed below in Section 2.2.2.

The two properties above play an important role in the efficiency of protocols built on additive secret sharing.

2.2.2 Basic Operations and Arithmetic Black-box Model

The performance of our protocols depends on lightweight and secure computations of basic operations in split form, including addition and multiplication of split secrets as well as generating split randoms. Here, we give a high-level overview of these operations and how they are modeled in the remainder of this thesis.

Addition in Split Form

Addition of two secret inputs (private or shared) does not need any interaction among parties; it requires only one local addition by each party:

$$[\![s_1 + s_2]\!]^{\mathcal{A}} = [\![s_1]\!]^{\mathcal{A}} + [\![s_2]\!]^{\mathcal{A}} \qquad \text{and} \qquad [\![s_1 + s_2]\!]^{\mathcal{B}} = [\![s_1]\!]^{\mathcal{B}} + [\![s_2]\!]^{\mathcal{B}} \tag{2.4}$$

Multiplication in Split Form

It is well-known that secure multiplication of two secrets, unlike addition, requires Alice and Bob to interact. One common practice for secure multiplication of split secrets over a finite field is through Beaver multiplication triples [63]:

1. *Input-independent offline phase*: Generating a triple of correlated randoms $\langle [\![a]\!], [\![b]\!], [\![c]\!] \rangle$ such that $[\![c]\!] = [\![a \cdot b]\!]$. Since this offline phase is input-independent, it can be carried out ahead of time before the actual inputs are available.

2. *Online phase*: For split secrets $[\![s_1]\!]$ and $[\![s_2]\!]$, Alice and Bob collaborate in a lightweight derandomization step that uses $\langle [\![a]\!], [\![b]\!], [\![c]\!] \rangle$ to obtain $[\![s_1 \cdot s_2]\!]$. For that, Alice and Bob first compute $[\![s_1 - a]\!]$, $[\![s_2 - b]\!]$, then reveal $s_1 - a$ and $s_2 - b$ to each other by exchanging their respective shares of these values; afterwards, they compute:

$$[\![s_1 \cdot s_2]\!] = [\![c]\!] + [\![s_1]\!](s_2 - b) + [\![s_2]\!](s_1 - a) - (s_1 - a)(s_2 - b) \tag{2.5}$$

   Note that this online phase does not use any cryptographic primitives. Moreover, it requires exactly one round of communication for exchanging shares of $[\![s_1 - a]\!]$ and $[\![s_2 - b]\!]$. Since Equation 2.5 involves only addition in split form and multiplication by publicly known integers, it can be computed non-interactively due to the linearity of secret splitting.

Generating Beaver triples has been extensively studied based on various assumptions and frameworks (e.g., [64]–[66]).

Split Random Generation

To generate a uniform random field element $[\![r]\!] \xleftarrow{\$} \mathbb{F}_q$ in split form, it suffices for Alice and Bob to choose uniform random shares $[\![r]\!]^{\mathcal{A}} \xleftarrow{\$} \mathbb{F}_q$ and $[\![r]\!]^{\mathcal{B}} \xleftarrow{\$} \mathbb{F}_q$, which inherently gives the uniform random $[\![r]\!]$. Sometimes, it is required to guarantee that $[\![r]\!] \neq 0$. Note that generating $[\![r]\!]$ as described above results in $[\![r]\!] = 0$ with probability $|\mathbb{F}_q|^{-1}$ which is negligible if the finite field $\mathbb{F}_q$ is large enough. However, if $[\![r]\!] = 0$ with probability $|\mathbb{F}_q|^{-1}$ is not acceptable, then a simple approach for Alice and Bob to make sure $[\![r]\!] \xleftarrow{\$} \mathbb{F}_q \setminus \{0\}$ is as follows:

1. Alice and Bob generate two uniform randoms $[\![r_1]\!], [\![r_2]\!] \xleftarrow{\$} \mathbb{F}_q$ as described above.

2. Alice and Bob securely compute $[\![r_1 \cdot r_2]\!]$ and reveal $r_1 \cdot r_2$ to each other by exchanging their respective shares of $[\![r_1 \cdot r_2]\!]$.

3. If $r_1 \cdot r_2 \neq 0$, then it is true that $r_1 \neq 0$ and $r_2 \neq 0$. As a result, they keep $[\![r_1]\!]$ as the desired non-zero split random $[\![r]\!]$ and dispose of $[\![r_2]\!]$. In the other case (of $r_1 \cdot r_2 = 0$), they discard the randoms and try again.

Arithmetic Black-box Model.

Our protocols use secret splitting to securely carry out the desired computations on private inputs and internal values that must not be revealed to either party. After obtaining the desired output(s) in split form, it is straightforward to reveal the result to either party (if needed) through reconstructing the desired shared output. In order to focus on the task at hand rather than the detailed specifications and security guarantees of the underlying protocols for basic operations discussed earlier, we use the Arithmetic Black-box (ABB) model of computation as in [67]. In the ABB model, each basic operation is provided through access to an ideal functionality. These ideal functionalities can be interpreted as a (hypothetical) trusted third party, who provides generation and storage of elements of $\mathbb{F}_q$ as well as arithmetic computations on field elements.

Below, we provide an intuitive presentation of how this model works for the non-trivial operation of secure multiplication. For computing the product $[\![s_1 \cdot s_2]\!]$, the trusted third party securely collects the inputs from both parties, carries out the computation, splits the result, and provides each party with a share of the computation result; i.e.,

1. Alice sends her shares $[\![s_1]\!]^{\mathcal{A}}$ and $[\![s_2]\!]^{\mathcal{A}}$ to the trusted server. Similarly, Bob sends his $[\![s_1]\!]^{\mathcal{B}}$ and $[\![s_2]\!]^{\mathcal{B}}$ to the trusted server.

2. The server reconstructs secrets s_1 and s_2. Then, it computes the product $s_3 = s_1 \cdot s_2$.

3. The server splits the multiplication result s_3 and distributes its two shares. Specifically, the server chooses a uniform random $r \xleftarrow{\$} \mathbb{F}_q$; then, it sends $[\![s_3]\!]^{\mathcal{A}} = s_3 - r$ and $[\![s_3]\!]^{\mathcal{B}} = r$ to, respectively, Alice and Bob.

One needs to make sure that in practice such an ideal functionality can be replaced with (existing) two-party protocols without compromising the overall security. This is possible because there are protocols [66] that securely realize these ideal functionalities in the Universal Composability (UC) model of [68]. The UC framework, introduced by Canetti [68], is a general-purpose method for the security analysis of cryptographic protocols. Protocols that are UC-secure remain secure even if executed sequentially or in parallel in composition with other protocols. Henceforth, we avoid the details of these basic operations and use them in a black-box manner.

2.3 Pseudorandom Function (PRF)

Let τ be a security parameter, and $F : \{0,1\}^{\tau} \times \{0,1\}^{\ell_{in}} \rightarrow \{0,1\}^{\ell_{out}}$ be a keyed function (the first input being the key denoted by $k \in \{0,1\}^{\tau}$) such that $F(k,x)$ is efficiently computable for all k and x. Typically, one chooses k uniformly at random and obtains a single-input function $F_k : \{0,1\}^{\ell_{in}} \rightarrow \{0,1\}^{\ell_{out}}$ defined by $F_k(\cdot) = F(k,\cdot)$. Function F is a PRF if no probabilistic polynomial-time (PPT) distinguisher can distinguish whether it is interacting with $F_k(\cdot)$ or a truly random function $f : \{0,1\}^{\ell_{in}} \rightarrow \{0,1\}^{\ell_{out}}$ [69].

Definition 2.1 (Pseudorandom Function [69])**.** *Let F be an efficiently computable keyed function and τ be the security parameter. Function F is a Pseudorandom Function if for all PPT distinguishers D, there is a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$|\Pr[D^{F_k(\cdot)}(1^{\tau}) = 1] - \Pr[D^{f(\cdot)}(1^{\tau}) = 1]| \leq \mathsf{negl}(\tau),$$

where the superscript of D denotes distinguisher's oracle access to the corresponding function. Moreover, the first probability is over the choice of key k and randomness of D, and the second probability is over the choice of truly random function f and randomness of D.

Note that the security parameter τ (key length) must be chosen for the desired security level, while the input and output lengths, ℓ_{in} and ℓ_{out}, are determined by the application in which the PRF is being used; ℓ_{in} and ℓ_{out} must be polynomially bounded in τ [69].

Oblivious PRF (OPRF)

An OPRF is a tuple $\langle F, \mathrm{Prot}_F^{\mathrm{OPRF}} \rangle$, where F is a PRF, and $\mathrm{Prot}_F^{\mathrm{OPRF}}$ is a two-party protocol that enables a party who has the key k to allow a querier to compute $F_k(x)$ for a private query x. The querier learns nothing other than $F_k(x)$, and the key holder learns nothing about x and $F_k(x)$ [28], [70], [71].

Distributed PRF (DPRF)

A t-out-of-n DPRF refers to a threshold evaluation of $F_k(\cdot)$; in a system of n parties, any subset of t parties can compute $F_k(x)$; but, any subset of fewer than t parties should not be able to obtain any information about $F_k(x)$ [72].

2.4 Convolution

In its most general form, linear convolution (denoted by $*$) is a mathematical binary operation that operates on two functions f and g to produce a third function h. It is well-known that this operation is linear and commutative. Below, we briefly review its definition as well as one of its special cases that will be used in the remainder of this document.

The convolution function $h = f * g$ is defined as the integral of the product of f and g after one is reversed and shifted; this integral is assessed for all possible values of shift. Formally,

$$h(x) = f(x) * g(x) = \int_{-\infty}^{\infty} f(y) \cdot g(x-y)\, dy \tag{2.6}$$

or, equivalently

$$h(x) = f(x) * g(x) = \int_{-\infty}^{\infty} f(x-y) \cdot g(y)\, dy \tag{2.7}$$

Discrete convolution refers to the case in which functions f and g are defined on the set $\mathbb{Z}$ of integers. Accordingly, the convolution of two sequences $\mathbf{F} = [f_0, f_1, \cdots, f_{\lambda_1 - 1}] \in \mathbb{C}^{\lambda_1}$

and $\mathbf{G} = [g_0, g_1, \cdots, g_{\lambda_2-1}] \in \mathbb{C}^{\lambda_2}$ is a sequence $\mathbf{H}$ of length $\lambda = \lambda_1 + \lambda_2 - 1$ such that for $0 \leq i \leq \lambda$:

$$h_i = \sum_{j=-\infty}^{\infty} f_j \cdot g_{i-j} \tag{2.8}$$

or, equivalently

$$h_i = \sum_{j=-\infty}^{\infty} f_{i-j} \cdot g_j \tag{2.9}$$

where any entry with an out-of-range index is considered to be equal to zero, i.e.,

$$\begin{aligned} f_k &= 0 \qquad \text{for} \quad k < 0 \;\; \text{and} \;\; k \geq \lambda_1 \\ g_k &= 0 \qquad \text{for} \quad k < 0 \;\; \text{and} \;\; k \geq \lambda_2 \end{aligned} \tag{2.10}$$

In what follows, we focus on discrete convolution of two sequences.

Circular Convolution

Circular convolution (denoted by $\circledast$) of two same-length sequences $\mathbf{F}, \mathbf{G} \in \mathbb{C}^{\lambda}$ is defined as a length-λ sequence $\mathbf{H}$ such that for $0 \leq k < \lambda$:

$$h_k = \sum_{i+j=k \pmod{\lambda}} f_i \cdot g_j = \sum_{i=0}^{\lambda-1} f_i \cdot g_{(k-i \mod \lambda)} \tag{2.11}$$

or, equivalently

$$h_k = \sum_{i+j=k \pmod{\lambda}} f_j \cdot g_i = \sum_{i=0}^{\lambda-1} f_{(k-i \mod \lambda)} \cdot g_i \tag{2.12}$$

Note that circular convolution is defined for same-length sequences; in the case that $\mathbf{F}$ and $\mathbf{G}$ have different length, circular convolution is possible only after padding the shorter sequence with enough zeros to equalize the lengths of both sequences. Although linear convolution and circular convolution are fundamentally different operations, an equivalency can be established among them under certain conditions. In case of convolving two sequences, circular convolution can be viewed as linear convolution when the sequences represent periodic functions; thus, shifting a sequence can be interpreted as its rotation because the values repeat due to the periodicity.

Fast Convolution Computation

It is well-known that fast transforms with a convolution property can be used for efficient computation of circular convolution [73]–[75]; i.e., the circular convolution of two sequences $\mathbf{F}, \mathbf{G} \in \mathbb{C}^{\lambda}$ can be computed with $\mathcal{O}(\lambda \log \lambda)$ computation complexity due to the convolution theorem, which states that the Discrete Fourier Transform (DFT) of $\mathbf{F} \circledast \mathbf{G}$ is equal to the pointwise product of the DFT of $\mathbf{F}$ and the DFT of $\mathbf{G}$ [73]–[75], i.e.,

$$\mathrm{DFT}(\mathbf{F} \circledast \mathbf{G}) = \mathrm{DFT}(\mathbf{F}) \odot \mathrm{DFT}(\mathbf{G}) \tag{2.13}$$

Hence, given any Fast Fourier Transform (FFT) algorithm that requires $\mathcal{O}(\lambda \log \lambda)$ work to compute the DFT (and its inverse DFT^{-1}) of a sequence, the circular convolution of two leqngth-λ sequences can also be computed with $\mathcal{O}(\lambda \log \lambda)$ work as follows:

$$\mathbf{F} \circledast \mathbf{G} = \mathrm{DFT}^{-1}\Big(\mathrm{DFT}(\mathbf{F}) \odot \mathrm{DFT}(\mathbf{G})\Big) \tag{2.14}$$

In Chapter 4 of this thesis, we will use the Number Theoretic Transform (NTT) for fast convolution computation. NTT is a generalization of DFT over the finite field $\mathbb{F}_q$ of integers modulo a prime q. NTT allows performing convolution computation on integer sequences as it satisfies the convolution property [74]. Using NTT (rather than DFT) for convolving integer sequences is beneficial as it is somewhat faster and also avoids floating-point arithmetic that could have resulted in roundoff errors.

The linear convolution of two sequences $\hat{\mathbf{F}} \in \mathbb{C}^{\lambda_1}$ and $\hat{\mathbf{G}} \in \mathbb{C}^{\lambda_2}$ can be obtained using any algorithm for computing circular convolution as follows [73]:

1. Augment both sequences $\hat{\mathbf{F}}$ and $\hat{\mathbf{G}}$ by padding them with enough zeros to make them both of length $\lambda = \lambda_1 + \lambda_2 - 1$; i.e.,

$$\mathbf{F} = \hat{\mathbf{F}}||\overbrace{[0, 0, \cdots, 0]}^{\text{length: } \lambda_2 - 1} \qquad \text{and} \qquad \mathbf{G} = \hat{\mathbf{G}}||\overbrace{[0, 0, \cdots, 0]}^{\text{length: } \lambda_1 - 1}$$

2. Compute $\mathbf{H}$, a sequence of length $\lambda = \lambda_1 + \lambda_2 - 1$, that is the circular convolution of the augmented sequences. More formally,

$$\mathbf{H} = \mathbf{F} \circledast \mathbf{G} = \hat{\mathbf{F}} * \hat{\mathbf{G}}$$

Although the above approach through zero padding of both sequences $\hat{\mathbf{F}}$ and $\hat{\mathbf{G}}$ is sufficient on its own, it is not advisable when one sequence is much longer than the other for two reasons [73] (w.l.o.g., $\lambda_1 \gg \lambda_2$):

- The shorter sequence $\hat{\mathbf{G}}$ must be padded with too many zeros, which results in an unacceptable amount of unnecessary computations.
- The DFT (or NTT) must be performed on very long sequences which may be impractical or inconvenient.

There exist well-known segmenting techniques such as *overlap-add* and *overlap-save* that are often used to avoid the drawbacks discussed above [73]. The idea in these techniques is to divide the (longer) sequence of length λ_1 into shorter blocks with a length that is proportional to λ_2; then, each block is separately convolved with the shorter sequence and the resulting convolution vectors are combined to construct $\mathbf{H} = \hat{\mathbf{F}} * \hat{\mathbf{G}}$ [73]. Using these segmenting approaches further improves the cost of convolution computation to $\mathcal{O}(\lambda_1 \log \lambda_2)$.

3. PRIVATE EQUALITY TESTING

Private equality testing (PET) is a central primitive in 2PC as many protocols require to test if two private or encrypted values are equal. In this chapter, we propose a lightweight and provably secure solution to this problem. To maintain generality, we emphasize that the test output must not be disclosed to either party (unless they agree to do so). Hence, our protocol returns the test result as a split secret.

In the original PET formulation (socialist millionaires problem), Alice has a private integer $s_1 \in \mathbb{F}_q$ and Bob has a private integer $s_2 \in \mathbb{F}_q$, and they intend to securely evaluate the predicate value "$(s_1 == s_2)$". Formally, the PET functionality (denoted by $\mathcal{F}_{\text{PET}}$) is defined as in Figure 3.1.

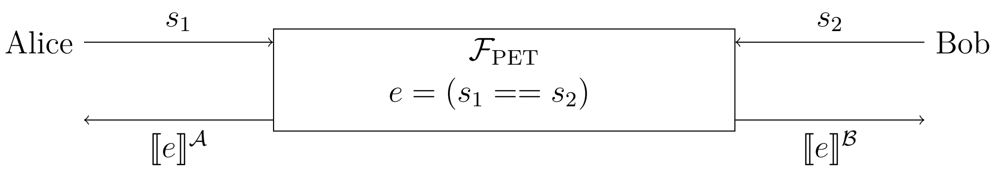

Alice
s_1
$\mathcal{F}_{\text{PET}}$
$e = (s_1 == s_2)$
s_2
Bob
$[\![e]\!]^{\mathcal{A}}$
$[\![e]\!]^{\mathcal{B}}$

Figure 3.1. $([\![e]\!]^{\mathcal{A}}, [\![e]\!]^{\mathcal{B}}) = \mathcal{F}_{\text{PET}}(s_1, s_2)$ such that $[\![e]\!] = 1$ if and only if $s_1 = s_2$.

Recall that an integer that is private to a party can be split in a non-interactive manner by setting the other party's share to be 0. Moreover, in many 2PC scenarios, s_1 and s_2 may not be known by either party (e.g., they are internal values of another protocol). Hence, we consider the more general case of PET in which inputs are additively split secrets, and (perhaps) unknown to either party. On the other hand, for any two split secrets it is true that $[\![s_1]\!] = [\![s_2]\!]$ if and only if $[\![s_1 - s_2]\!] = 0$. Thus, it suffices to give a secure protocol that computes the "*equal to zero*" functionality (denoted by $\mathcal{F}_{\text{EQZ}}$) as illustrated in Figure 3.2. In the remainder of this document, private equality testing refers to $\mathcal{F}_{\text{EQZ}}$.

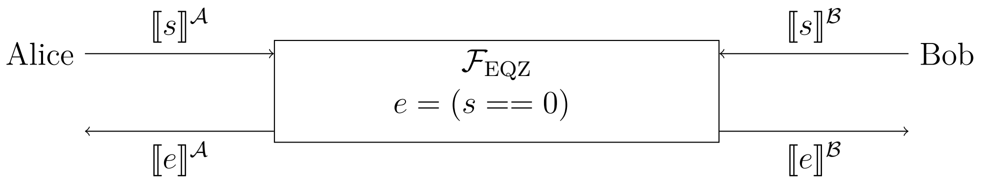

Alice
$[\![s]\!]^{\mathcal{A}}$
$\mathcal{F}_{\text{EQZ}}$
$e = (s == 0)$
$[\![s]\!]^{\mathcal{B}}$
Bob
$[\![e]\!]^{\mathcal{A}}$
$[\![e]\!]^{\mathcal{B}}$

Figure 3.2. $([\![e]\!]^{\mathcal{A}}, [\![e]\!]^{\mathcal{B}}) = \mathcal{F}_{\text{EQZ}}([\![s]\!]^{\mathcal{A}}, [\![s]\!]^{\mathcal{B}})$ such that $[\![e]\!] = 1$ if and only if $s = 0$.

3.1 Summary of Contributions

Our equality testing protocol, ZeroTest (Protocol 3.1), is built based on the two following observations:

- $[\![s]\!] = 0$ if and only if $[\![s]\!]^{\mathcal{A}} = -[\![s]\!]^{\mathcal{B}}$
- If q is an ℓ-bit prime (i.e., $\ell = \lceil \log_2 q \rceil$), then any integer $x \in \mathbb{F}_q$ has a unique binary representation $Bits(x) = x_{\ell-1}x_{\ell-2} \cdots x_0$ that does not result in a modular wraparound in $\mathbb{F}_q$ when computing $x = \sum_{i=0}^{\ell-1} x_i \cdot 2^i \mod q$; i.e,

$$\sum_{i=0}^{\ell-1} x_i \cdot 2^i = \Big(\sum_{i=0}^{\ell-1} x_i \cdot 2^i\Big) \mod q \tag{3.1}$$

We use the above-mentioned unique binary representation to resolve the gap caused by the boolean nature of $\mathcal{F}_{\text{EQZ}}$ while working in the arithmetic circuit model. Even a naive implementation of the solution we give would require only $2\ell - 1$ invocations to a secure multiplication protocol over $\mathbb{F}_q$, which itself results in a much more computationally efficient protocol compared to those based on bit-decomposition [39], [43] that require $\mathcal{O}(\ell \log \ell)$ secure multiplications (with large hidden constant factors; see Table 3.1). But, our final protocol does even better: It obtains a more lightweight equality testing by using a new *modular conversion* technique that allows all internal computations to be over $\mathbb{F}_3$ rather than $\mathbb{F}_q$, while the final output is still additively split over $\mathbb{F}_q$. In particular, our ZeroTest protocol carries out the above-mentioned $2\ell-1$ secure multiplications over $\mathbb{F}_3$; this avoids the additional overhead of arithmetic over $\mathbb{F}_q$. Note that the result of a number of multiplications over $\mathbb{F}_3$ will be also in $\mathbb{F}_3$. However, since $\mathcal{F}_{\text{EQZ}}$ is defined over $\mathbb{F}_q$, its output must be in $\mathbb{F}_q$ as well. Our modular conversion technique uses *only three* secure multiplications over $\mathbb{F}_q$ to convert the test result $e \in \{0, 1\} \subset \mathbb{F}_3$ to its counterpart in $\mathbb{F}_q$. In summary, ZeroTest carries out all internal arithmetic over $\mathbb{F}_3$ and uses exactly three secure multiplications to generate the final output in $\mathbb{F}_q$.

3.2 Related Work

Private equality testing is one of the most well-studied problems in the 2PC setting as it has proven to be a core primitive in many secure two-party applications [52]–[55]. Solutions in various models and frameworks have been proposed based on secret sharing [39], [43]–[46], garbled circuits [47], [48], and homomorphic encryption [49]–[51]. Below, we focus on the existing work based on secret sharing as those are closest to our approach. In what follows, ℓ denotes the bit-length of the input and κ is a correctness parameter. Moreover, since among basic operations over $\mathbb{F}_q$ the secure multiplication is the dominant cost factor in the arithmetic model, the performance of protocols is measured based on the count of required invocations to a secure multiplication protocol and the total number of communication rounds.

Damgård *et al.* [39] proposed, for the first time, a bit-decomposition technique that converts a shared secret $[\![s]\!]$ into the sharings of the bits of s in a constant number of rounds. This tool was the first bridge between arithmetic and boolean circuit models in 2PC. Building on bit-decomposition, [39] gives a secure protocol for computing $\mathcal{F}_{\mathrm{EQZ}}([\![s]\!]^{\mathcal{A}}, [\![s]\!]^{\mathcal{B}})$; this protocol requires $98\ell + 94\ell \log_2 \ell$ invocations in a total of 39 rounds. Nishide *et al.* [43] proposes a simplified bit-decomposition scheme that can be used to securely compute $\mathcal{F}_{\mathrm{EQZ}}([\![s]\!]^{\mathcal{A}}, [\![s]\!]^{\mathcal{B}})$ through $98\ell + 47\ell \log_2 \ell$ invocations in 26 rounds of communication. Moreover, [43] gives another equality testing protocol (without bit-decomposition) that requires 81ℓ invocations in a total of 8 rounds. Another course of research [45], [51] obtains sublinear (in ℓ) number

Table 3.1.
Comparison of the secure realizations of $\mathcal{F}_{\mathrm{EQZ}}$ functionality in the ABB model of computation and semi-honest threat model. $\ell = \lceil \log_2 q \rceil$ denotes the input's bit-length, and κ is a correctness parameter.

Solution	Secure Mult. Over $\mathbb{F}_q$	Rounds	Error Rate
[39] (Bit-decomposition)	$98\ell + 94\ell \log_2 \ell$	39	0
[43] (Bit-decomposition)	$98\ell + 47\ell \log_2 \ell$	26	0
[43] (Bit-oriented)	81ℓ	8	0
[43] (Legendre symbol)	$12k$	4	2^{-k}
[44] (Prob. ModCNV$_{q\to 2}$)	k	8	2^{-k}
ZeroTest (Bit-oriented)	3	$3 + \log_2 \ell$	0

of invocations in the online phase of the protocols; however, these latter protocols are based on more intensive offline computations and give results in a total of $\mathcal{O}(\ell)$ invocations.

For some probabilistic equality testing protocols [43], [44] the number of invocations is independent of the input bit-length ℓ; in these solutions, the count of invocations merely depends on a correctness parameter κ that results in an error probability $2^{-\Omega(\kappa)}$. Note that in spite of this, the overall computation and communication complexities inherently depend on ℓ because of arithmetic over $\mathbb{F}_q$. The probabilistic solution in [43] computes the Legendre symbol multiple times: The key idea is that for a uniform random $r \in \mathbb{F}_q$, $s = 0$ always results in $\left(\frac{r}{q}\right) = \left(\frac{s+r}{q}\right)$; on the other hand, if $s \neq 0$, then $\left(\frac{s}{q}\right) = \left(\frac{s+r}{q}\right)$ with a non-negligible probability. This approach requires 12κ invocations in 4 rounds of communication to obtain the desired failure probability. Yu *et al.* [44] adapts the solution above and uses a probabilistic and lightweight modular conversion instead of the Legendre symbol; this gives a performance of κ invocations in 8 rounds. Note that our deterministic modular conversion differs from the probabilistic PET solution based on randomized modular conversion in [44].

3.3 Secure and Lightweight Protocol for Computing $\mathcal{F}_{\mathbf{EQZ}}([\![s]\!]^{\mathcal{A}}, [\![s]\!]^{\mathcal{B}})$

In this section, we present our provably secure two-party protocol, ZeroTest (Protocol 3.1), for securely computing $([\![e]\!]^{\mathcal{A}}, [\![e]\!]^{\mathcal{B}}) = \mathcal{F}_{\mathrm{EQZ}}([\![s]\!]^{\mathcal{A}}, [\![s]\!]^{\mathcal{B}})$ as illustrated in Figure 3.2. ZeroTest consists of two phases:

- Phase 1 computes the predicate $([\![s]\!] == 0)$ using arithmetic over $\mathbb{F}_3$, and obtains the test result $[\![e]\!] \in \{0, 1\} \subset \mathbb{F}_3$.

- Phase 2 converts the output of Phase 1 to its counterpart in $\mathbb{F}_q$.

Notation 3.1. Hereafter, superscripts (3) and (q) for an integer e denote the integer in, respectively, $\mathbb{F}_3$ and $\mathbb{F}_q$. For instance, $[\![e^{(3)}]\!] = 1$ means that there are private shares $[\![e^{(3)}]\!]^{\mathcal{A}}, [\![e^{(3)}]\!]^{\mathcal{B}} \in \mathbb{F}_3$ such that $[\![e^{(3)}]\!]^{\mathcal{A}} + [\![e^{(3)}]\!]^{\mathcal{B}} \mod 3 = 1$.

Moreover, if the superscripts are used for a bit α (rather than an integer), then $\alpha^{(3)}$ and $\alpha^{(q)}$ denote the corresponding integers in, respectively, $\mathbb{F}_3$ and $\mathbb{F}_q$:

- If $\alpha = 0$, then $\alpha^{(3)} = 0 \in \mathbb{F}_3$ and $\alpha^{(q)} = 0 \in \mathbb{F}_q$

- If $\alpha = 1$, then $\alpha^{(3)} = 1 \in \mathbb{F}_3$ and $\alpha^{(q)} = 1 \in \mathbb{F}_q$

Notation 3.2. For bit operations, this document follows the established notations used in the literature, namely, if α and β are two bits then

- $\bar{\alpha}$ denotes the complement of α, i.e., $\bar{\alpha} = 1 - \alpha$
- $\alpha \oplus \beta$ denotes the exclusive-OR of bits α and β.

Phase 1: Evaluating the Equality Predicate Using Arithmetic Over $\mathbb{F}_3$

Let $\mathit{Bits}([\![s]\!]^{\mathcal{A}}) = \alpha_{\ell-1}\alpha_{\ell-2}\cdots\alpha_2\alpha_1\alpha_0$ and $\mathit{Bits}(-[\![s]\!]^{\mathcal{B}}) = \beta_{\ell-1}\beta_{\ell-2}\cdots\beta_2\beta_1\beta_0$ be the unique binary representations of, respectively, $[\![s]\!]^{\mathcal{A}}$ and $-[\![s]\!]^{\mathcal{B}}$ with the property summarized in Equation 3.1. Clearly, $[\![s]\!] = 0$ if and only if $\mathit{Bits}([\![s]\!]^{\mathcal{A}}) = \mathit{Bits}(-[\![s]\!]^{\mathcal{B}})$. In other words, $[\![s]\!] = 0$ if and only if $\bar{\alpha}_i \oplus \beta_i = 1$ for all $0 \leq i \leq \ell - 1$. Moreover, it can easily be verified that

$$\begin{aligned} (\bar{\alpha}_i \oplus \beta_i)^{(3)} &= \bar{\alpha}_i^{(3)} + \beta_i^{(3)} - 2(\bar{\alpha}_i^{(3)} \cdot \beta_i^{(3)}) \\ (\bar{\alpha}_i \oplus \beta_i)^{(q)} &= \bar{\alpha}_i^{(q)} + \beta_i^{(q)} - 2(\bar{\alpha}_i^{(q)} \cdot \beta_i^{(q)}) \end{aligned} \tag{3.2}$$

This enables securely computing $[\![z_i^{(3)}]\!] = [\![(\bar{\alpha}_i \oplus \beta_i)^{(3)}]\!]$ through one secure multiplication invocation over $\mathbb{F}_3$, as follows:

1. Alice and Bob compute $[\![\bar{\alpha}_i^{(3)} \cdot \beta_i^{(3)}]\!]$
2. Alice computes $[\![z_i^{(3)}]\!]^{\mathcal{A}} = \bar{\alpha}_i^{(3)} - 2[\![\bar{\alpha}_i^{(3)} \cdot \beta_i^{(3)}]\!]^{\mathcal{A}}$
3. Bob computes $[\![z_i^{(3)}]\!]^{\mathcal{B}} = \beta_i^{(3)} - 2[\![\bar{\alpha}_i^{(3)} \cdot \beta_i^{(3)}]\!]^{\mathcal{B}}$

Hence, Alice and Bob obtain $[\![z_i^{(3)}]\!]$ for $0 \leq i \leq \ell - 1$ in one round via ℓ independent secure multiplications carried out in parallel. Obviously, $[\![s]\!] = 0$ if and only if $[\![z_i^{(3)}]\!] = 1$ for all $0 \leq i \leq \ell - 1$. Hence, for obtaining the desired output $[\![e^{(3)}]\!]$, it suffices to compute

$$[\![e^{(3)}]\!] = \prod_{i=0}^{\ell-1} [\![z_i^{(3)}]\!] \tag{3.3}$$

Computing the product of Equation 3.3 in a binary tree fashion needs $\ell - 1$ secure multiplications in a total of $\log_2 \ell = \log_2 \log_2 q$ rounds.

Alice encodes $[\![e^{(3)}]\!]^{\mathcal{A}}$ as follows:

$$\begin{aligned} 0 \in \mathbb{F}_3 &\to a_1a_0 = 00 \\ 1 \in \mathbb{F}_3 &\to a_1a_0 = 01 \\ 2 \in \mathbb{F}_3 &\to a_1a_0 = 10 \end{aligned}$$

Bob encodes $[\![e^{(3)}]\!]^{\mathcal{B}}$ as follows:

$$\begin{aligned} 0 \in \mathbb{F}_3 &\to b_1b_0 = 00 \\ 1 \in \mathbb{F}_3 &\to b_1b_0 = 10 \\ 2 \in \mathbb{F}_3 &\to b_1b_0 = 01 \end{aligned}$$

(a) Local encodings: Alice and Bob encode their shares of $[\![e^{(3)}]\!]$ into two pairs of bits (resp.) a_1a_0 and b_1b_0 such that $a_1a_0 \oplus b_1b_0 = 00$ if and only if $[\![e^{(3)}]\!] = 0$.

$$\begin{aligned}
&\text{Combination 1:} && ([\![e^{(3)}]\!]^{\mathcal{A}}, [\![e^{(3)}]\!]^{\mathcal{B}}) = (0,0) \longrightarrow a_1a_0 \oplus b_1b_0 = 00 \\
&\text{Combination 2:} && ([\![e^{(3)}]\!]^{\mathcal{A}}, [\![e^{(3)}]\!]^{\mathcal{B}}) = (1,2) \longrightarrow a_1a_0 \oplus b_1b_0 = 00 \\
&\text{Combination 3:} && ([\![e^{(3)}]\!]^{\mathcal{A}}, [\![e^{(3)}]\!]^{\mathcal{B}}) = (2,1) \longrightarrow a_1a_0 \oplus b_1b_0 = 00 \\
&\text{Combination 4:} && ([\![e^{(3)}]\!]^{\mathcal{A}}, [\![e^{(3)}]\!]^{\mathcal{B}}) = (0,1) \longrightarrow a_1a_0 \oplus b_1b_0 = 10 \\
&\text{Combination 5:} && ([\![e^{(3)}]\!]^{\mathcal{A}}, [\![e^{(3)}]\!]^{\mathcal{B}}) = (1,0) \longrightarrow a_1a_0 \oplus b_1b_0 = 01 \\
&\text{Combination 6:} && ([\![e^{(3)}]\!]^{\mathcal{A}}, [\![e^{(3)}]\!]^{\mathcal{B}}) = (2,2) \longrightarrow a_1a_0 \oplus b_1b_0 = 11
\end{aligned}$$

(b) All possible combinations of $([\![e^{(3)}]\!]^{\mathcal{A}}$, $[\![e^{(3)}]\!]^{\mathcal{B}})$, and the resulting pair of bits $a_1a_0 \oplus b_1b_0$.

Figure 3.3. Modular conversion encodings and potential combinations: Alice and Bob each locally encodes its respective share of $[\![e^{(3)}]\!]$ into two bits a_1a_0 and b_1b_0 such that $a_1a_0 \oplus b_1b_0 = 00$ if and only if $[\![e^3]\!] = 0$.

Phase 2: Modular Conversion in Split Form

Since the input $[\![s]\!]$ is an element in $\mathbb{F}_q$, we have to make sure that so is the output. Although Phase 1 correctly computes the equality testing output, it returns $[\![e^{(3)}]\!] \in \{0,1\} \subset \mathbb{F}_3$. Phase 2 of ZeroTest maps $[\![e^{(3)}]\!]$ to its counterpart $[\![e^{(q)}]\!] \in \{0,1\} \subset \mathbb{F}_q$. In order to do so, we introduce a modular conversion technique that uses only three secure multiplications over $\mathbb{F}_q$ in a total of two rounds. Below, we first describe a high-level overview of this technique, then we give step-by-step instructions for it.

Procedure overview: Alice and Bob locally encode their respective shares $[\![e^{(3)}]\!]^{\mathcal{A}}$ and $[\![e^{(3)}]\!]^{\mathcal{B}}$ into two pairs of bits, respectively, a_1a_0 and b_1b_0 as in Figure 3.3a; by construction, this encoding results in:

$$\begin{aligned} c_1c_0 &= a_1a_0 \oplus b_1b_0 = 00 && \text{if } [\![e^{(3)}]\!] = 0 \\ c_1c_0 &= a_1a_0 \oplus b_1b_0 \in \{01, 10, 11\} && \text{if } [\![e^{(3)}]\!] = 1 \end{aligned} \tag{3.4}$$

Then, both parties cooperate to obliviously compute $[\![\delta^{(q)}]\!]$ that is the count of non-zero bits in c_1c_0. Equation 3.4 ensures that

$$\begin{aligned} [\![\delta^{(q)}]\!] &= 0 && \iff && [\![e^{(3)}]\!] = 0 \\ [\![\delta^{(q)}]\!] &\in \{1, 2\} && \iff && [\![e^{(3)}]\!] = 1 \end{aligned} \tag{3.5}$$

Hence, Alice and Bob must securely map $[\![\delta^{(q)}]\!]$ to $[\![e^{(q)}]\!]$ as follows:

$$[\![e^{(q)}]\!] = \begin{cases} 0 & \text{if } [\![\delta^{(q)}]\!] = 0 \\ 1 & \text{if } [\![\delta^{(q)}]\!] \in \{1, 2\} \end{cases} \tag{3.6}$$

In summary, the overall modular conversion works as illustrated below:

$$[\![e^{(3)}]\!]^{\mathcal{A}}, [\![e^{(3)}]\!]^{\mathcal{B}} \xrightarrow[\text{as in Figure 3.3a}]{\text{local encodings}} a_1a_0,\ b_1b_0 \xrightarrow[\text{2 secure mult.}]{\text{first round}} [\![\delta^{(q)}]\!] \xrightarrow[\text{1 secure mult.}]{\text{second round}} [\![e^{(q)}]\!] \tag{3.7}$$

Step-by-step instructions:

1. Alice and Bob use Figure 3.3a to obtain their respective private two-bit encodings a_1a_0 and b_1b_0 according to their private shares $[\![e^{(3)}]\!]^{\mathcal{A}}$ and $[\![e^{(3)}]\!]^{\mathcal{B}}$.

2. In order to securely compute the two-bit exclusive-OR $c_1c_0 = a_1a_0 \oplus b_1b_0$ with outputs in $\mathbb{F}_q$, Alice and Bob simply use two instances of Equation 3.2 for $i \in \{0, 1\}$ as follows:

$$[\![c_i^{(q)}]\!] = [\![(a_i \oplus b_i)^{(q)}]\!] = [\![a_i^{(q)} + b_i^{(q)} - 2(a_i^{(q)} \cdot b_i^{(q)})]\!] \tag{3.8}$$

Then, they compute (non-interactively)

$$[\![\delta^{(q)}]\!] = [\![c_0^{(q)}]\!] + [\![c_1^{(q)}]\!] \tag{3.9}$$

3. At this point, Alice and Bob must apply the mapping of Equation 3.6 in a secure manner. This can be done through one secure multiplication because of the following observation: The choices of private encodings a_1a_0 and b_1b_0 (see Figure 3.3b) guarantee that

$$\begin{aligned} [\![e^{(3)}]\!]^{\mathcal{A}} = 2 &\longrightarrow [\![\delta^{(q)}]\!] \in \{0,2\} \subset \mathbb{F}_q \\ [\![e^{(3)}]\!]^{\mathcal{A}} \neq 2 &\longrightarrow [\![\delta^{(q)}]\!] \in \{0,1\} \subset \mathbb{F}_q, \end{aligned} \tag{3.10}$$

As a result, in order to obliviously obtain $[\![e^{(q)}]\!]$, it is sufficient that Alice chooses an auxiliary private integer $\tau^{(q)}$ such that

$$\tau^{(q)} = \begin{cases} 2^{-1} \mod q & \text{if } [\![e^{(3)}]\!]^{\mathcal{A}} = 2 \\ 1 & \text{if } [\![e^{(3)}]\!]^{\mathcal{A}} \neq 2 \end{cases} \tag{3.11}$$

Afterwards, Alice and Bob engage in a single secure multiplication over $\mathbb{F}_q$ to compute

$$[\![e^{(q)}]\!] = [\![\tau^{(q)} \cdot \delta^{(q)}]\!] \tag{3.12}$$

Remark 3.1. The property summarized in Equation 3.10 symmetrically holds for Bob; i.e.,

$$\begin{aligned} [\![e^{(3)}]\!]^{\mathcal{B}} = 2 &\longrightarrow [\![\delta^{(q)}]\!] \in \{0,2\} \subset \mathbb{F}_q \\ [\![e^{(3)}]\!]^{\mathcal{B}} \neq 2 &\longrightarrow [\![\delta^{(q)}]\!] \in \{0,1\} \subset \mathbb{F}_q \end{aligned} \tag{3.13}$$

As a result, the roles of Alice and Bob can be exchanged in Step 3 of the above instructions.

Lemma 3.1 (Correctness and Security of Protocol 3.1)**.** *Protocol* ZeroTest*, on Alice's and Bob's respective private inputs $[\![s]\!]^{\mathcal{A}}$ and $[\![s]\!]^{\mathcal{B}}$, securely computes the "equal to zero" functionality $([\![e]\!]^{\mathcal{A}}, [\![e]\!]^{\mathcal{B}}) = \mathcal{F}_{\text{EQZ}}([\![s]\!]^{\mathcal{A}}, [\![s]\!]^{\mathcal{B}})$ in the semi-honest threat model. Moreover, the protocol is unconditionally secure in the ABB model of computation.*

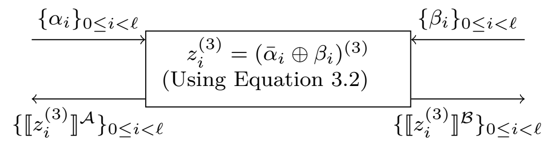
ZEROTEST

Alice | **Bob**

On input $[\![s]\!]^{\mathcal{A}} \in \mathbb{F}_q$ | On input $[\![s]\!]^{\mathcal{B}} \in \mathbb{F}_q$

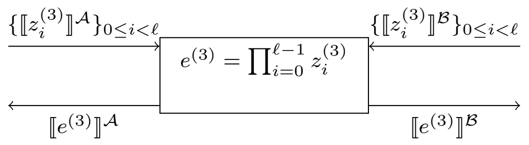
.... Phase 1: Computing $[\![e^{(3)}]\!] = ([\![s]\!] == 0)$ ....

1 : For $\alpha = [\![s]\!]^{\mathcal{A}}$, $\mathit{Bits}(\alpha) = \alpha_{\ell-1}\alpha_{\ell-2}\cdots\alpha_1\alpha_0$ | For $\beta = -[\![s]\!]^{\mathcal{B}}$, $\mathit{Bits}(\beta) = \beta_{\ell-1}\beta_{\ell-2}\cdots\beta_1\beta_0$



$\{\alpha_i\}_{0\le i<\ell}$
$\{\beta_i\}_{0\le i<\ell}$
$z_i^{(3)} = (\bar{\alpha}_i \oplus \beta_i)^{(3)}$
(Using Equation 3.2)
$\{[\![z_i^{(3)}]\!]^{\mathcal{A}}\}_{0\le i<\ell}$
$\{[\![z_i^{(3)}]\!]^{\mathcal{B}}\}_{0\le i<\ell}$

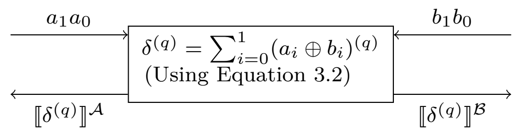
2 : $\{[\![z_i^{(3)}]\!]^{\mathcal{A}}\}_{0\le i<\ell}$ | $\{[\![z_i^{(3)}]\!]^{\mathcal{B}}\}_{0\le i<\ell}$



$\{[\![z_i^{(3)}]\!]^{\mathcal{A}}\}_{0\le i<\ell}$
$\{[\![z_i^{(3)}]\!]^{\mathcal{B}}\}_{0\le i<\ell}$
$e^{(3)} = \prod_{i=0}^{\ell-1} z_i^{(3)}$
$[\![e^{(3)}]\!]^{\mathcal{A}}$
$[\![e^{(3)}]\!]^{\mathcal{B}}$

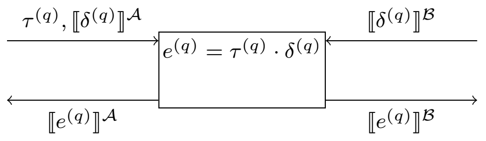
.... Phase 2: Modular Conversion - Convert $[\![e^{(3)}]\!]$ to $[\![e^{(q)}]\!]$ ....

3 : $a_1a_0 = \text{encode}([\![e^{(3)}]\!]^{\mathcal{A}})$ as in Figure 3.3(a) | $b_1b_0 = \text{encode}([\![e^{(3)}]\!]^{\mathcal{B}})$ as in Figure 3.3(a)



a_1a_0
b_1b_0
$\delta^{(q)} = \sum_{i=0}^{1}(a_i \oplus b_i)^{(q)}$
(Using Equation 3.2)
$[\![\delta^{(q)}]\!]^{\mathcal{A}}$
$[\![\delta^{(q)}]\!]^{\mathcal{B}}$

4 : Choose $\tau^{(q)} \in \{1, 2^{-1}\}$ as in Equation 3.11



$\tau^{(q)}, [\![\delta^{(q)}]\!]^{\mathcal{A}}$
$[\![\delta^{(q)}]\!]^{\mathcal{B}}$
$e^{(q)} = \tau^{(q)} \cdot \delta^{(q)}$
$[\![e^{(q)}]\!]^{\mathcal{A}}$
$[\![e^{(q)}]\!]^{\mathcal{B}}$

5 : **return** $[\![e^{(q)}]\!]^{\mathcal{A}}$ | **return** $[\![e^{(q)}]\!]^{\mathcal{B}}$

Protocol 3.1. Secure realization of $([\![e]\!]^{\mathcal{A}}, [\![e]\!]^{\mathcal{B}}) = \mathcal{F}_{\text{EQZ}}([\![s]\!]^{\mathcal{A}}, [\![s]\!]^{\mathcal{B}})$ using modular conversion

Proof. Proof of correctness for Protocol 3.1 is straightforward by construction. For its security, note that i) all steps in both phases merely use local computations and/or secure multiplications in split form, and ii) none of the shared or private secrets are ever revealed to either party. Moreover, the property summarized in Equation 3.10 (symmetrically, Equation 3.13) does not provide any information to Alice (symmetrically, Bob) because it only depends on her own private share $[\![e^{(3)}]\!]^{\mathcal{A}}$ and the fact that in secret splitting, each share alone is independent of the secret. $\square$

3.4 Overall Performance

Note that for a secure multiplication over an ℓ-bit field parties need to exchange $\mathcal{O}(\ell)$ bits, and each party has to carry out $\mathcal{O}(\ell^2)$ local bit operations (see Section 2.2 for Beaver's multiplication technique). On the other hand, protocol ZeroTest uses $\mathcal{O}(\ell)$ secure multiplications over $\mathbb{F}_3$ and three secure multiplications over $\mathbb{F}_q$. This results in a total of $\mathcal{O}(\ell^2)$ bit operations by each party and communicating $\mathcal{O}(\ell)$ bits. Note that without modular conversion, all $\mathcal{O}(\ell)$ multiplications of Phase 1 would have been over $\mathbb{F}_q$, which would have resulted in $\mathcal{O}(\ell^3)$ bit operations with a communication complexity of $\mathcal{O}(\ell^2)$. Finally, ZeroTest needs $3+\log_2 \ell$ rounds of communication: $1+\log_2 \ell$ rounds in Phase 1 and 2 rounds in Phase 2.

ZeroTest requires only three invocations of secure multiplication over $\mathbb{F}_q$, which is by far the best result among the existing equality testing protocols. This improvement comes at the cost of $\mathcal{O}(\log \ell)$ round complexity. We argue that this trade-off is not a drawback at all. Note that for a primitive as fundamental as private equality testing, the actual cost of the protocol is prominent compared to its asymptotic behavior. For all practical purposes

- the exact number of rounds for ZeroTest (i.e., $3+\log_2 \ell$ rounds) is much smaller than the large constant number of required rounds by solutions based on bit-decomposition. Recall that the protocols of [39] and [43] that need, respectively, 39 and 26 rounds; and,

- the exact number of rounds for ZeroTest is comparable to the number of required rounds for other solutions such as the bit-oriented protocol of [43] and the probabilistic solution of [44] (both require 8 rounds).

4. SECURE WILDCARD PATTERN MATCHING

In its various forms, pattern matching is a fundamental problem in computer science, and has been studied extensively. The problem has also been considered by the security and privacy community; in the 2PC framework, Alice has a length-n private text string $\mathbf{T}$ and Bob has a length-m private pattern $\mathbf{P}$; they wish to learn about the occurrences of $\mathbf{P}$ in $\mathbf{T}$ without revealing any additional information about their private inputs to each other. Very credible arguments were given in the literature about why participants would want to not reveal their inputs to each other, arguments that we refrain from repeating (for motivational details see [4], [56]–[59]). There are many protocols that enable Alice and Bob to securely carry out such computations; however, the existing protocols for the problem mentioned above have drawbacks that detract from practical deployment. For some, the drawback is their quadratic complexity $\mathcal{O}(mn)$; even those protocols with quasilinear complexity [76] make use of expensive cryptographic primitives such as homomorphic encryption and impose limitations on input and alphabet sizes.

In this chapter, we consider Secure Wildcard Pattern Matching (SWPM), in which the wildcard (or, "don't care") symbol is a non-alphabet symbol that matches any alphabet symbol. We address three variants of SWPM, namely, *search*, *counting* and *decision* versions. We give two 2PC protocols for each version: First, we solve all variants assuming a black-box access to the ideal functionality $\mathcal{F}_{\text{EQZ}}$ that was discussed in Chapter 3. Our second approach uses a *relaxed* "equal to zero" functionality, $\mathcal{F}_{\text{REQZ}}$, which can be securely realized much more efficiently compared to $\mathcal{F}_{\text{EQZ}}$; however, using $\mathcal{F}_{\text{REQZ}}$ requires additional steps to achieve correctness and the desired level of security. All our protocols avoid the previously discussed drawbacks of existing solutions for these functionalities: The protocols we propose have linearithmic computation complexity and linear communication complexity in a constant number of rounds. Furthermore, our schemes use only lightweight computational primitives, modular addition and multiplication, and avoid expensive cryptographic primitives such homomorphic encryption and public-key operations.

4.1 Problem Definition

Let $\Sigma = \{1, 2, \ldots, \sigma\}$ be a fixed finite alphabet and $\star \notin \Sigma$ be the wildcard symbol that matches any symbol in $\Sigma \cup \{\star\}$. In this document, for the sake of simplicity, we mainly focus on solving SWPM with wildcards in the pattern string; however, Section 4.4.3 discusses how our solutions can be easily modified to support wildcards in both text and pattern strings. In SWPM, Alice has a text string $\mathbf{T} \in \Sigma^n$ and Bob has a pattern string $\mathbf{P} \in (\Sigma \cup \{\star\})^m$, where $m \leq n$. Bob wants to search in $\mathbf{T}$ for (possibly overlapping) occurrences of $\mathbf{P}$; neither party is willing to reveal anything about its private input to the other party, other than Bob learning his prescribed output.

Notation 4.1. In the remainder of this chapter, we use l to denote the number of wildcards in the pattern string $\mathbf{P}$; moreover, m' denotes the count of occurrences of alphabet symbols in $\mathbf{P}$, i.e., $m' = m - l$.

Notation 4.2. In this work, $\mathbf{T}_i = t_i t_{i+1} \cdots t_{i+m-1}$ for $1 \leq i \leq N$; we consider pattern matching without wraparounds, i.e., $N = n - m + 1$. Moreover, $\mathbf{T}_i \overset{\star}{=} \mathbf{P}$ denotes that $\mathbf{P}$ matches the substring $\mathbf{T}_i$, considering that the wildcard symbol $\star$ matches all symbols. For example, $312 \overset{\star}{=} 3 \star 2$ (whereas, $312 \neq 3 \star 2$).

Remark 4.3 will discuss how our scheme can be modified to also detect matchings with wraparounds, i.e., when $N = n$ and all indices in $\mathbf{T}_i = t_i t_{i+1} \cdots t_{i+m-1}$ are $\pmod{n}$.

In general, the goal is that Bob learns only *about* the occurrences of $\mathbf{P}$ in $\mathbf{T}$, while Alice learns nothing. The three variants of SWPM are:

1. **Search version $(\perp, \Gamma) = \mathcal{F}^{S}_{\mathbf{SWPM}}(\mathbf{T}, \mathbf{P})$:** As illustrated in Figure 4.1, Bob learns all positions in $\mathbf{T}$ at which $\mathbf{P}$ occurs; i.e., he learns the set

$$\Gamma = \{i \mid \mathbf{T}_i \overset{\star}{=} \mathbf{P}\} \tag{4.1}$$

2. **Counting version $(\perp, \gamma) = \mathcal{F}^{C}_{\mathbf{SWPM}}(\mathbf{T}, \mathbf{P})$:** As illustrated in Figure 4.2, Bob learns only the count $\gamma = |\Gamma|$.

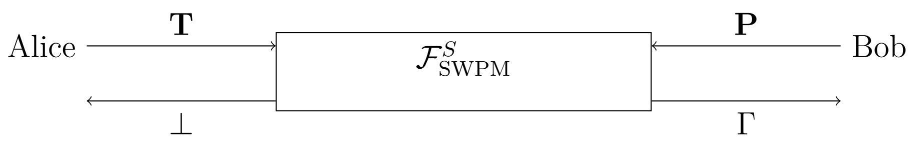


Figure 4.1. $(\perp, \Gamma) = \mathcal{F}^S_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$

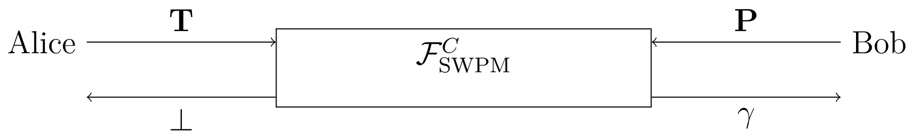


Figure 4.2. $(\perp, \gamma) = \mathcal{F}^C_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$

3. **Decision version $(\perp, (\gamma > 0)) = \mathcal{F}^D_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$:** As illustrated in Figure 4.3, Bob learns only the predicate value $(\gamma > 0)$.

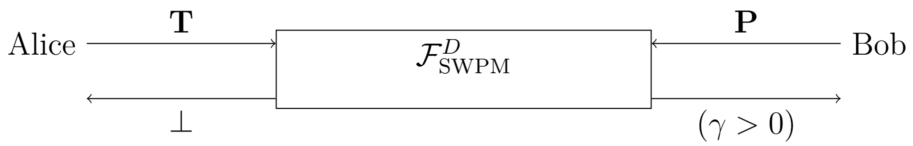


Figure 4.3. $(\perp, (\gamma > 0)) = \mathcal{F}^D_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$

4.2 Related work

Due to its importance, several studies have investigated secure two-party pattern matching; some address the problem under the stronger malicious adversarial model [56], [76]–[78], while others assume the less powerful semi-honest adversarial model (or they can be optimized for the semi-honest model) [56], [59], [79]. A group of studies, including [78], [80], [81] concentrate on binary-alphabet pattern matching. In this document, we focus on the more general version of secure wildcard pattern matching over any arbitrary finite alphabet.

Recently, Riazi *et al.* [59] provided a solution based on Yao's Garbled Circuit where parties require $\mathcal{O}(1)$ communication rounds at the price of $\mathcal{O}(nm \log |\Sigma|)$ computational complexity. Their scheme has been built based on the provably secure Yao's garbled circuit protocol, which uses expensive primitives in the oblivious transfer parts of the protocol.

Baron *et al.* [56] propose a scheme (appropriate for both malicious and semi-honest models) that reduces pattern matching to a linear algebra formulation that allows for generic solutions based on any Additively Homomorphic Encryption. In the semi-honest model, their approach requires $\mathcal{O}(1)$ rounds of communication, $\mathcal{O}(n+m)$ encryptions and exponentiations, $\mathcal{O}(nm)$ multiplications as well as $\mathcal{O}((n+m)\kappa)$ communication complexity, where κ is a security parameter.

In [76], Vergnaud builds an approach on top of the string matching algorithms in [82], [83]. The scheme has $\mathcal{O}(n \log m)$ computational complexity and requires constant rounds of communication; however, it uses homomorphic encryption and imposes restrictions on inputs and alphabet size.

A different approach to address the SWPM problem is constructing an automaton based on the pattern string and obliviously evaluating it on the text string. Works in [84], [85] suggest such schemes; these solutions require $\mathcal{O}(nm)$ and $\mathcal{O}(nm|\Sigma|)$ computational complexity for, respectively, Bob and Alice. An elaborate comparison for the solutions of [84] and [85] can be found in [85].

4.3 Summary of Contributions

Below, we briefly review the contributions of this chapter.

4.3.1 Wildcard Pattern Matching Algorithm

In Section 4.4, we propose a novel convolution-based algorithm for wildcard pattern matching without any security and privacy considerations. The algorithm's computational bottleneck is the use of (just one) convolution operator that results in an overall complexity of $\mathcal{O}(n \log m)$. The scheme consists of three steps: Preprocessing the alphabet, convolution computation, and processing the convolution output to find all occurrence positions. Our scheme is a template algorithm in the sense that it has multiple possible instantiations based on various alphabet preprocessing approaches.

4.3.2 Secure Protocols

In Section 4.5, we give protocols that build on our wildcard pattern matching algorithm to securely realize the three variants of SWPM in the 2PC setting. In particular, we propose

1. A lightweight protocol for secure convolution computation. The protocol requires $\mathcal{O}(n \log m)$ local computation by each party and only $\mathcal{O}(n)$ secure multiplication invocations in one round of communication.

2. Lightweight protocols for each of the *search*, *counting* and *decision* versions of SWPM in the semi-honest model. We give two secure solutions for each version, one based on the $\mathcal{F}_{\text{EQZ}}$ functionality and the other based on a relaxed private equality testing, namely, $\mathcal{F}_{\text{REQZ}}$. All our protocols require constant rounds of communication between Alice and Bob. Moreover, the computational bottleneck in all these protocols is the secure convolution computation; they all have a linearithmic computation complexity and a linear communication complexity.

4.3.3 Experimental Results

As a proof of concept, we have implemented our solutions based on $\mathcal{F}_{\text{REQZ}}$, that we call **LiLiP** (**Li**ghtweight, **Li**nearithmic & **P**rivate), and it performs very well in practice. Our experimental results presented in Section 4.6 demonstrate the practicality of our solutions even for very large input sizes. Our experiments show that our scheme is 1.4× to 21.5× faster than PriSearch [59], which is itself superior to other approaches.

4.3.4 Functionality Generalization

In Section 4.7, we go beyond the semi-honest threat model and point out that the search version of SWPM , as the most commonly used variant of SWPM has a definitional drawback in the stronger adversarial models that makes the use of any secure protocol for computing it unavailing. In particular, we show that if Bob uses a judiciously crafted input pattern $\hat{\mathbf{P}}$ rather than his prescribed input $\mathbf{P}$, he can learn Alice's input $\mathbf{T}$. Such an attack by Bob is possible due to the power of the wildcard symbol, and it has nothing to do with the choice of the secure two-party protocol used for computing $(\perp, \Gamma) = \mathcal{F}^{S}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$. In order to address this issue, we propose the use of a filtering function τ that restricts what Bob may learn. Accordingly, we suggest two additional variants of SWPM, in each of which Bob learns the position of exactly one occurrence of $\mathbf{P}$ in $\mathbf{T}$ (if any).

4.4 Convolution-based Wildcard Pattern Matching Algorithm

The connection of pattern matching and convolution has been long studied [76], [82], [83], [86]–[88]. This section presents a novel algorithm based on convolution, Algorithm 1, that we propose for solving the traditional wildcard pattern matching without any security and privacy concerns. Algorithm 1 consists of three phases:

1. Input strings $\mathbf{T}$ and $\mathbf{P}$ are mapped to their respective unique counterparts $\hat{\mathbf{T}}$ and $\hat{\mathbf{P}}$ in a new alphabet domain where certain properties hold. For this, Alice and Bob require a (one-time) preprocessing of $\Sigma \cup \{\star\}$ to map each symbol in it to a unique value in the new alphabet domain; later, the above mapping is used to obtain $\hat{\mathbf{T}}$ and $\hat{\mathbf{P}}$ for any pair of input strings $\mathbf{T} \in \Sigma^n$ and $\mathbf{P} \in (\Sigma \cup \{\star\})^m$.

2. Obtaining a score vector $\mathbf{C} = [c_1, c_2, \cdots, c_N]$, where the i^{th} entry $c_i = \sum_{j=0}^{m-1} \hat{t}_{i+j} \cdot \hat{p}_j$ is the matching score for $\mathbf{P}$ and the subtrsing $\mathbf{T}_i$. Specifically, for $1 \leq i \leq N$

$$c_i = m' \iff \mathbf{T}_i \overset{\star}{=} \mathbf{P} \tag{4.2}$$

   As will become apparent, exactly one convolution computation is sufficient for obtaining the score vector $\mathbf{C}$.

3. Processing the score vector $\mathbf{C}$ to obtain the set Γ containing all positions in $\mathbf{T}$ at which $\mathbf{P}$ occurs, i.e., $\Gamma = \{i \mid \mathbf{T}_i \overset{\star}{=} \mathbf{P}\} = \{i \mid c_i = m'\}$.

Below, we first describe the properties required for preprocessing the symbols in $\Sigma \cup \{\star\}$, and provide two preprocessing approaches that are sufficient for solving wildcard pattern matching. Then, we present the overall algorithm.

4.4.1 Alphabet Preprocessing

As the first step in our pattern matching algorithm, we need to translate the alphabet $\Sigma \cup \{\star\}$ to a new domain with certain properties described below. This preprocessing is based on a pair of correlated mappings $f_{\mathbf{T}}$ and $f_{\mathbf{P}}$ that are used to map the symbols appearing in, respectively, $\mathbf{T}$ and $\mathbf{P}$ into the new alphabet domain.

Notation 4.3. For the mapping function $f_{\mathbf{T}} : \Sigma \to \Sigma'$, where Σ' is the new alphabet domain, we use $\hat{\mathbf{T}} = f_{\mathbf{T}}(\mathbf{T})$ to denote $\hat{\mathbf{T}} = f_{\mathbf{T}}(t_1) f_{\mathbf{T}}(t_2) \cdots f_{\mathbf{T}}(t_n)$. Similarly, for $f_{\mathbf{P}} : \Sigma \cup \{\star\} \to \Sigma'$, we use $\hat{\mathbf{P}} = f_{\mathbf{P}}(\mathbf{P})$ to denote $\hat{\mathbf{P}} = f_{\mathbf{P}}(p_1) f_{\mathbf{T}}(p_2) \cdots f_{\mathbf{T}}(p_m)$.

The desired properties for mappings $f_{\mathbf{T}}$ and $f_{\mathbf{P}}$ are:

1. Both $f_{\mathbf{T}}$ and $f_{\mathbf{P}}$ must be injections to avoid any collisions among alphabet symbols.

2. The codomain of $f_{\mathbf{T}}$ and $f_{\mathbf{P}}$ (i.e., Σ') must be algebraically suitable for fast convolution computations.

3. For any substring $\mathbf{T}_i = t_i t_{i+1} \cdots t_{i+m-1}$, the scalar product $f_{\mathbf{T}}(\mathbf{T}_i) \cdot f_{\mathbf{P}}(\mathbf{P}) = m'$ if and only if $\mathbf{T}_i \overset{\star}{=} \mathbf{P}$.

4. (Optional) The codomain of mappings $f_{\mathbf{T}}$ and $f_{\mathbf{P}}$ (i.e., Σ') supports modular arithmetic over the finite field $\mathbb{F}_q$.

Although Property 4 is not necessary for efficient wildcard pattern matching, it is included to avoid floating-point arithmetic and obtain a higher level of security in protocols of Section 4.5. The latter is because modular addition and multiplication of a secret and a uniform random provide perfect secrecy over, respectively, $\mathbb{F}_q$ and $\mathbb{F}_q \setminus \{0\}$ (a.k.a additive one-time pad and multiplicative one-time pad). On the other hand, non-modular arithmetic would provide only statistical security and the randoms used to hide a secret must be much greater (in magnitude) than the secret itself.

Example 4.1 (Complex Mappings $f_{\mathbf{T}} : \Sigma \to \mathbb{C}$ and $f_{\mathbf{P}} : \Sigma \cup \{\star\} \to \mathbb{C}$)**.** Here, we describe the first pair of mappings with the complex field $\mathbb{C}$ used as the new alphabet domain. This pair of mappings fulfills properties 1,2 and 3, but not property 4. With $\mathbb{C}$ as the new arithmetic space, one needs to use FFT to apply convolution efficiently. The mapping functions $f_{\mathbf{T}} : \Sigma \to \mathbb{C}$ and $f_{\mathbf{P}} : \Sigma \cup \{\star\} \to \mathbb{C}$ are defined as follows:

$$f_{\mathbf{T}}(k) = \exp(\frac{2\pi\sqrt{-1}}{\sigma}k), \quad \text{for } k \in \Sigma \tag{4.3}$$

$$f_{\mathbf{P}}(k) = \begin{cases} \exp(-\frac{2\pi\sqrt{-1}}{\sigma}k) & \text{if } k \in \Sigma \\ 0 & \text{if } k = \star \end{cases} \tag{4.4}$$

Lemma 4.1. *Mappings of Example 4.1 satisfy properties 1, 2 and 3.*

Proof. Property 4 does not hold since mappings deal with powers of σ^{th} root of unity in $\mathbb{C}$. Below, we prove that all other properties hold:

Property 1: $f_{\mathbf{T}}$ (resp. $f_{\mathbf{P}}$) is injective on Σ because it maps any alphabet symbols $k \in \Sigma = \{1, \ldots, \sigma\}$ to the k^{th} (resp. $-k^{th}$) power of σ^{th} primitive root of unity. Moreover,

$$0 = f_{\mathbf{P}}(\star) \neq f_{\mathbf{P}}(k) = \exp(-\frac{2\pi\sqrt{-1}}{\sigma}k) \ \text{ for } k \in \Sigma$$

Property 2: Since the selected codomain is $\mathbb{C}$, the classic Fast Fourier Transform can be used for efficient convolution computation [73]–[75].

Property 3: We need to show that

$$f_{\mathbf{T}}(\mathbf{T}_i) \cdot f_{\mathbf{P}}(\mathbf{P}) = \hat{\mathbf{T}}_i \cdot \hat{\mathbf{P}} = m' \Longleftrightarrow \mathbf{T}_i \overset{\star}{=} \mathbf{P}$$

Let $M_\star = \{j \mid (1 \leq j \leq m) \wedge (p_j = \star)\}$, i.e., the set of all indices corresponding to wildcards in $\mathbf{P}$. Note that

$$\begin{aligned} f_{\mathbf{T}}(\mathbf{T}_i) \cdot f_{\mathbf{P}}(\mathbf{P}) &= \sum_{j=1}^{m} f_{\mathbf{T}}(t_{i+j-1}) f_{\mathbf{P}}(p_j) \\ &= \sum_{j \notin M_\star} f_{\mathbf{T}}(t_{i+j-1}) f_{\mathbf{P}}(p_j) + \sum_{j \in M_\star} f_{\mathbf{T}}(t_{i+j-1}) f_{\mathbf{P}}(p_j) \\ &= \sum_{j \notin M_\star} \exp(\frac{2\pi\sqrt{-1}}{\sigma} t_{i+j-1}) \exp(-\frac{2\pi\sqrt{-1}}{\sigma} p_j) + \sum_{j \in M_\star} 0 \\ &= \sum_{j \notin M_\star} \exp(\frac{2\pi\sqrt{-1}}{\sigma} t_{i+j-1}) \exp(-\frac{2\pi\sqrt{-1}}{\sigma} p_j) \\ &= \sum_{j \notin M_\star} \exp(\frac{2\pi\sqrt{-1}}{\sigma} (t_{i+j-1} - p_j)) \end{aligned}$$

It remains to show that

$$\sum_{j \notin M_\star} \exp(\frac{2\pi\sqrt{-1}}{\sigma} (t_{i+j-1} - p_j)) = m' \Longleftrightarrow t_{i+j-1} = p_j \text{ for all } j \notin M_\star$$

The "$\Leftarrow$" direction is straightforward because if $t_{i+j-1} = p_j$, then $t_{i+j-1} - p_j = 0$; thus,

$$\sum_{j \notin M_\star} \exp(\frac{2\pi\sqrt{-1}}{\sigma} (t_{i+j-1} - p_j)) = \sum_{j \notin M_\star} \exp(0) = \sum_{j \notin M_\star} 1 = m'$$

The "$\Rightarrow$" direction holds because $1 \leq p_j, t_{i+j-1} \leq \sigma$, and that any mismatch $t_{i+j-1} \neq p_j$ results in

$$1 - \sigma \leq t_{i+j-1} - p_j \neq 0 \leq \sigma - 1$$

Hence, the real part of $\exp(\frac{2\pi\sqrt{-1}}{\sigma}(t_{i+j-1} - p_j))$ for a character mismatch is guaranteed to be smaller than 1. Hence, the existence of at least one such symbol mismatch gives a summation result with a real part that is smaller than m', and this implies that the scalar product $f_{\mathbf{T}}(\mathbf{T}_i) \cdot f_{\mathbf{P}}(\mathbf{P})$ cannot be equal to m'.

$\square$

As mentioned earlier, mapping alphabet symbols to the complex space as in Example 4.1 is sufficient for solving wildcard pattern matching efficiently. However, because of dealing with the floating-point arithmetic, the method suffers from being limited with machine-precision in numerical computations. Moreover, using it in the two-party protocols of Section 4.5 cannot provide perfect secrecy as modular arithmetic is not present. Example 4.2 extends this mapping to its counterpart over the finite field $\mathbb{F}_q$. The resulting pair of mappings satisfies all four above-mentioned properties.

Example 4.2 (Modular Mappings $f_{\mathbf{T}} : \Sigma \to \mathbb{F}_q$ and $f_{\mathbf{P}} : \Sigma \cup \{\star\} \to \mathbb{F}_q$)**.** In order to also satisfy the optional property discussed earlier, we use the finite field $\mathbb{F}_q$ as the new alphabet domain. With $\mathbb{F}_q$ as the new arithmetic space, Number Theoretic Transform (NTT) can be used for efficient convolution computation. The prime q must be chosen based on the input size n and alphabet size σ: Fix a prime $q = \alpha n + 1$ such that $q > \sigma$ and α is a positive integer; the existence of such a prime is guaranteed due to Dirichlet's theorem [89].

The mapping functions $f_{\mathbf{T}} : \Sigma \to \mathbb{F}_q$ and $f_{\mathbf{P}} : \Sigma \cup \{\star\} \to \mathbb{F}_q$ will be defined analogous to their definition in Example 4.1. Particularly, let ω_σ be a σ^{th} principal root of unity modulo q; then,

$$f_{\mathbf{T}}(k) = \omega_\sigma^k \mod q, \quad \text{for } k \in \Sigma \tag{4.5}$$

$$f_{\mathbf{P}}(k) = \begin{cases} \omega_\sigma^{-k} \mod q & \text{if } k \in \Sigma \\ 0 & \text{if } k = \star \end{cases} \tag{4.6}$$

where $\omega_\sigma^{-k} \mod q = (\omega_\sigma^{-1})^k \mod q$, and $\omega_\sigma^{-1} \equiv \omega_\sigma^{\sigma-1} \mod q$. Note that the σ^{th} principal root of unity ω_σ must not be confused with the n^{th} principal root of unity ω_n used in NTT where n is the length of vectors to be convolved.

Lemma 4.2. *The mappings of Example 4.2 satisfy all desired properties 1,2,3 and 4.*

Proof. Clearly, Property 4 holds since the codomain of $f_{\mathbf{T}}$ and $f_{\mathbf{P}}$ is $\mathbb{F}_q$. Moreover, the validity of properties 1, 2, and 3 follow similar arguments to their proofs in Lemma 4.1 due to i) the similar functions' definitions for the symbol translations using powers of σ^{th} primitive root of unity, and ii) the fact that NTT is a generalization of the classic Discrete Fourier Transform to the finite field $\mathbb{F}_q$. $\square$

Although the above transformations involve (modular) exponentiation computations for alphabet symbols, these are for a one-time preprocessing step that is carried out for the alphabet Σ well before input strings $\mathbf{T}$ and $\mathbf{P}$ are known. This preprocessing can be done ahead of time for selected values of n close to the ones expected in practice, e.g., powers of two, in which case padding with zeros and partitioning $\mathbf{T}$ can be used to make the encountered n have one such length. After such a preprocessing, any desired number of instances of the wildcard pattern matching over Σ can be solved without these modular exponentiations.

Remark 4.1. One might consider using the mappings $f_{\mathbf{T}}$ and $f_{\mathbf{P}}$ such that

$$f_{\mathbf{T}}(k) = k \mod q, \quad \text{for } k \in \Sigma$$

$$f_{\mathbf{P}}(k) = \begin{cases} k^{-1} \mod q & \text{if } k \in \Sigma \\ 0 & \text{if } k = \star \end{cases}$$

to avoid roots of unity and the $|\Sigma|$ modular exponentiations in the preprocessing. However, with a closer look it becomes apparent that the use of such mapping functions would violate Property 3; i.e., these mappings could result in $\hat{\mathbf{T}}_i \cdot \hat{\mathbf{P}} = m'$ even if $\hat{\mathbf{T}}_i \not\overset{\star}{=} \hat{\mathbf{P}}$.

4.4.2 Proposed Algorithm (Wildcards in Only P)

Given a pair of mappings $f_{\mathbf{T}}$ and $f_{\mathbf{P}}$ as described in Section 4.4.1, one can solve the wildcard pattern matching problem through exactly one convolution as in Algorithm 1. Henceforth, we assume the pair of mappings $f_{\mathbf{T}}$ and $f_{\mathbf{P}}$ as in equations, respectively, 4.5 and 4.6 from Example 4.2. After applying the preprocessing and obtaining $\hat{\mathbf{T}} = f_{\mathbf{T}}(\mathbf{T})$ and $\hat{\mathbf{P}} = f_{\mathbf{P}}(\mathbf{P})$, a score vector $\mathbf{C}$ is computed such that

$$c_i = \hat{\mathbf{T}}_i \cdot \hat{\mathbf{P}} \qquad \text{for } \ 1 \leq i \leq N \tag{4.7}$$

Notation 4.4. Hereafter, any use of the convolution operator refers to discrete linear convolution, unless stated otherwise. Moreover, when we use $\mathbf{C} = \hat{\mathbf{T}} * \hat{\mathbf{P}}^{rev}$, we assume that $\mathbf{C}$ contains only the "$\max(|\hat{\mathbf{T}}|, |\hat{\mathbf{P}}^{rev}|) - \min(|\hat{\mathbf{T}}|, |\hat{\mathbf{P}}^{rev}|) + 1$" *valid entries* of the actual convolution vector $\hat{\mathbf{C}} = \hat{\mathbf{T}} * \hat{\mathbf{P}}^{rev}$; i.e., $\mathbf{C}$ contains only those entries in $\hat{\mathbf{C}}$ that correspond to a complete overlap of the shorter input vector $\hat{\mathbf{P}}^{rev}$ with a $\hat{\mathbf{T}}_i$ for $1 \leq i \leq N$.

In order to obtain the score vector $\mathbf{C}$ with the desired property summarized in Equation 4.7, it suffices to convolve $\hat{\mathbf{T}}$ and $\hat{\mathbf{P}}^{rev}$; i.e., $\mathbf{C} = \hat{\mathbf{T}} * \hat{\mathbf{P}}^{rev}$.

Algorithm 1 Pattern Matching with Wildcards Only in $\mathbf{P}$

Inputs: Text string $\mathbf{T} \in \Sigma^n$ and pattern string $\mathbf{P} \in (\Sigma \cup \{\star\})^m$, where $m \leq n$

Outputs: Set $\Gamma = \{i \mid \mathbf{T}_i \overset{\star}{=} \mathbf{P}\}$

```
1:  procedure WILDCARD-PATTERN-MATCHING($\mathbf{T}, \mathbf{P}$)
2:      $\Gamma \leftarrow \emptyset$
3:      $m' \leftarrow |\{p_i \mid p_i = \star\}|$
4:      $\hat{\mathbf{T}} \leftarrow f_{\mathbf{T}}(\mathbf{T})$                ▷ See Equation 4.5
5:      $\hat{\mathbf{P}} \leftarrow f_{\mathbf{P}}(\mathbf{P})$                ▷ See Equation 4.6
6:      $\mathbf{C} \leftarrow \hat{\mathbf{T}} * \hat{\mathbf{P}}^{rev}$       ▷ Using NTT in this case
7:      $N \leftarrow n - m + 1$
8:      for $i \leftarrow 1$ to $N$ do
9:          if $c_i = m'$ then
10:             $\Gamma \leftarrow \Gamma \cup \{i\}$
11:         end if
12:     end for
13:     return $\Gamma$
14: end procedure
```

Theorem 4.4.1 (Correctness of Algorithm 1)**.** *Algorithm 1, on inputs $\mathbf{T} \in \Sigma^n$ and $\mathbf{P} \in (\Sigma \cup \{\star\})^m$, returns the set of all positions in the text string $\mathbf{T}$ at which $\mathbf{P}$ occurs; i.e., $\Gamma = \{i \mid \mathbf{T}_i \overset{\star}{=} \mathbf{P}\}$.*

Proof. This follows directly from Property 3 of alphabet mappings $f_{\mathbf{T}}$ and $f_{\mathbf{P}}$, and the fact that the convolutions $\mathbf{C} = \hat{\mathbf{T}} * \hat{\mathbf{P}}^{rev}$ results in $c_i = \hat{\mathbf{T}}_i \cdot \hat{\mathbf{P}}$ for $1 \leq i \leq N$ by the definition of convolution. □

Remark 4.2. Algorithm 1 can easily be extended to also detect matchings with wraparounds. In order to do so, it suffices to use $N = n$ (rather than $N = n - m + 1$) and modify Step 6 1 as follows:

$$\mathbf{C} = (\hat{\mathbf{T}}||[\hat{t}_1, \hat{t}_2, \cdots, \hat{t}_{m-1}]) * \hat{\mathbf{P}}^{rev}$$

4.4.3 Proposed Algorithm (Wildcards in Both P and T)

Algorithm 1 presented above supports wildcard symbols only in the pattern string $\mathbf{P}$. Here, we discuss how Algorithm 2 simply extends Algorithm 1 to also support wildcards in the text string $\mathbf{T}$. The first step towards supporting wildcards in $\mathbf{T}$ is to augment Equation 4.5 for $f_{\mathbf{T}}$ to also include the mapping $f_{\mathbf{T}}(\star) = 0$; i.e., the augmented mapping $f'_{\mathbf{T}} : \Sigma \cup \{\star\} \to \mathbb{F}_q$ is defined as below:

$$f'_{\mathbf{T}}(k) = \begin{cases} \omega_\sigma^k \mod q & \text{if } k \in \Sigma \\ 0 & \text{if } k = \star \end{cases} \tag{4.8}$$

Using the augmented mapping $f'_{\mathbf{T}}$, the convolution $\hat{\mathbf{T}} * \hat{\mathbf{P}}^{rev}$ results in a contribution of $+0$ to all corresponding score values in $\mathbf{C}$ for alignments of $\mathbf{T}$ and $\mathbf{P}$ that involve a wildcard in $\mathbf{T}$. However, this convolution does not account for the $+1$ contributions to the score values due to matches between alphabet symbols in $\mathbf{P}$ and wildcards in $\mathbf{T}$. Algorithm 2 addresses this shortcoming by computing another convolution $\bar{\mathbf{T}} * \bar{\mathbf{P}}^{rev}$ where the binary vectors $\bar{\mathbf{T}} = g_{\mathbf{T}}(\mathbf{T})$ and $\bar{\mathbf{P}} = g_{\mathbf{P}}(\mathbf{P})$ are obtained using the additional pair of mappings $g_{\mathbf{T}}, g_{\mathbf{P}} : \Sigma \cup \{\star\} \to \{0, 1\} \subset \mathbb{F}_q$ defined below:

- $g_{\mathbf{T}}$ maps all alphabet symbols to 0, and it maps $\star$ to 1; i.e.,

$$g_{\mathbf{T}}(k) = \begin{cases} 0 & \text{if } k \in \Sigma \\ 1 & \text{if } k = \star \end{cases} \tag{4.9}$$

- $g_{\mathbf{P}}$ maps all alphabet symbols to 1, and it maps $\star$ to 0; i.e.,

$$g_{\mathbf{P}}(k) = \begin{cases} 1 & \text{if } k \in \Sigma \\ 0 & \text{if } k = \star \end{cases} \tag{4.10}$$

Hence, it is sufficient to modify Step 6 in Algorithm 1 as follows:

$$\mathbf{C} \leftarrow \hat{\mathbf{T}} * \hat{\mathbf{P}}^{rev} + \bar{\mathbf{T}} * \bar{\mathbf{P}}^{rev} \tag{4.11}$$

Theorem 4.4.2 (Correctness of Algorithm 2)**.** *Algorithm 2, on inputs* $\mathbf{T}(\Sigma \cup \{\star\})^n$ *and* $\mathbf{P} \in (\Sigma \cup \{\star\})^m$*, returns the set of all positions in text* $\mathbf{T}$ *at which* $\mathbf{P}$ *occurs when both may contain wildcards; i.e.,* $\Gamma = \{i \mid \mathbf{T}_i \overset{\star}{=} \mathbf{P}\}$*.*

Proof. Any entry in the score vector $\mathbf{C}$ is a sum of two terms:

$$c_i = \hat{\mathbf{T}}_i \cdot \hat{\mathbf{P}} + \bar{\mathbf{T}}_i \cdot \bar{\mathbf{P}}$$

These terms together account for all possible types of character matching, and each match is accounted for exactly once. In particular, the structures of mappings $f'_{\mathbf{T}}, f_{\mathbf{P}} : \Sigma \cup \{\star\} \rightarrow \mathbb{F}_q$ and $g_{\mathbf{T}}, g_{\mathbf{P}} : \Sigma \cup \{\star\} \rightarrow \{0, 1\} \subset \mathbb{F}_q$ ensure that all three match types discussed below are accounted for:

- **Match type $\boldsymbol{t_{i+j-1} \neq \star}$, $\boldsymbol{p_j \neq \star}$ and $\boldsymbol{t_{i+j-1} = p_j}$**: In this case $\hat{t}_{i+j-1}\hat{p}_j = 1$ and $\bar{t}_{i+j-1}\bar{p}_j = 0$ resulting in an overall contribution of $+1$ to c_i.
- **Match type $\boldsymbol{t_{i+j-1} = \star}$ and $\boldsymbol{p_j \neq \star}$**: In this case $\hat{t}_{i+j-1}\hat{p}_j = 0$ and $\bar{t}_{i+j-1}\bar{p}_j = 1$ resulting in a contribution of $+1$ to c_i.
- **Match type $\boldsymbol{t_{i+j-1} \in \Sigma \cup \{\star\}}$ and $\boldsymbol{p_j = \star}$**: In this case $\hat{t}_{i+j-1}\hat{p}_j = 0$ and $\bar{t}_{i+j-1}\bar{p}_j = 0$ resulting in an overall contribution of $+0$ to c_i. Note that (similar to Algorithm 1), the comparison of Step 11 is $c_i = m'$ rather than $c_i = m$. Recall that $m' = m - l$, where l is the count of wildcards in $\mathbf{P}$. Hence, contribution of a character matching that corresponds to $p_j = \star$ is accounted for via reducing the required match count (-1 per wildcard in $\mathbf{P}$) rather than contributing a $+1$ to the score value c_i.

□

Algorithm 2 Pattern Matching with Wildcards in both **T** and **P**

```
    Inputs:  Text string $\mathbf{T} \in (\Sigma \cup \{\star\})^n$ and pattern string $\mathbf{P} \in (\Sigma \cup \{\star\})^m$, where $m \leq n$
    Outputs: Set $\Gamma = \{i \mid \mathbf{T}_i \overset{\star}{=} \mathbf{P}\}$
 1: procedure WILDCARD-PATTERN-MATCHING($\mathbf{T}, \mathbf{P}$)
 2:     $\Gamma \leftarrow \emptyset$
 3:     $m' \leftarrow |\{p_i \mid p_i = \star\}|$
 4:     $\hat{\mathbf{T}} \leftarrow f'_{\mathbf{T}}(\mathbf{T})$                                ▷ See Equation 4.8
 5:     $\hat{\mathbf{P}} \leftarrow f_{\mathbf{P}}(\mathbf{P})$                                ▷ See Equation 4.6
 6:     $\bar{\mathbf{T}} \leftarrow g_{\mathbf{T}}(\mathbf{T})$                                ▷ See Equation 4.9
 7:     $\bar{\mathbf{P}} \leftarrow g_{\mathbf{P}}(\mathbf{P})$                                ▷ See Equation 4.10
 8:     $\mathbf{C} \leftarrow \hat{\mathbf{T}} * \hat{\mathbf{P}}^{rev} + \bar{\mathbf{T}} * \bar{\mathbf{P}}^{rev}$    ▷ $*$ denotes convolution [by NTT in this case]
 9:     $N \leftarrow n - m + 1$
10:     for $i \leftarrow 1$ to $N$ do
11:         if $c_i = m'$ then
12:             $\Gamma \leftarrow \Gamma \cup \{i\}$
13:         end if
14:     end for
15:     return $\Gamma$
16: end procedure
```

Remark 4.3. Algorithm 2 can easily be extended to also detect matchings with wraparounds. In order to do so, it suffices to use $N = n$ (rather than $N = n - m + 1$) and modify Step 8 as follows:

$$\mathbf{C} = (\hat{\mathbf{T}}||[\hat{t}_1, \hat{t}_2, \cdots, \hat{t}_{m-1}]) * (\hat{\mathbf{P}}^{rev} + \bar{\mathbf{T}}||[\bar{t}_1, \bar{t}_2, \cdots, \bar{t}_{m-1}]) * \bar{\mathbf{P}}^{rev}$$

4.5 Secure Two-party Protocols for Wildcard Pattern Matching

In this section, we investigate secure wildcard pattern matching (SWPM) and give protocols that deploy Algorithm 1 to securely compute the desired functionalities $\mathcal{F}^S_{\text{SWPM}}, \mathcal{F}^C_{\text{SWPM}}$ and $\mathcal{F}^D_{\text{SWPM}}$ as defined in Section 4.1. In what follows, all arithmetic is modular; this is possible because the underlying alphabet preprocessing (from Example 4.2) has $\mathbb{F}_q$ for a prime q as the new alphabet domain. Moreover, note that Alice and Bob locally compute, respectively, $\hat{\mathbf{T}} = f_{\mathbf{T}}(\mathbf{T})$ and $\hat{\mathbf{P}} = f_{\mathbf{P}}(\mathbf{P})$ without any privacy concerns; this is because the underlying alphabet is known by both parties and the alphabet preprocessing is independent of the private inputs $\mathbf{T}$ and $\mathbf{P}$. Hence, there is only need for secure convolution computation and secure protocols that process the resulting (split) score vector to securely realize each SWPM version $\mathcal{F}^S_{\text{SWPM}}, \mathcal{F}^C_{\text{SWPM}}$ and $\mathcal{F}^D_{\text{SWPM}}$.

4.5.1 Secure Convolution Computation

Recall from Section 2.4 that efficient computation of discrete linear convolution of two vectors is possible through efficient discrete circular convolution. Hence, it suffices to give a secure protocol that computes circular convolution of private vectors. Formally, the two-party convolution functionality in split form (illustrated in Figure 4.4) takes as inputs, Alice's and Bob's length-λ private input vectors $\mathbf{X}$ and $\mathbf{Y}$, and outputs $[\![\mathbf{X} \circledast \mathbf{Y}]\!]$; i.e., the functionality is defined as $([\![\mathbf{Z}]\!]^{\mathcal{A}}, [\![\mathbf{Z}]\!]^{\mathcal{B}}) = \mathcal{F}_{\text{CONV}}(\mathbf{X}, \mathbf{Y})$ such that $[\![\mathbf{Z}]\!] = [\![\mathbf{X} \circledast \mathbf{Y}]\!]$. The following Protocol 4.1, Convolution, securely computes this functionality via only λ secure multiplications.

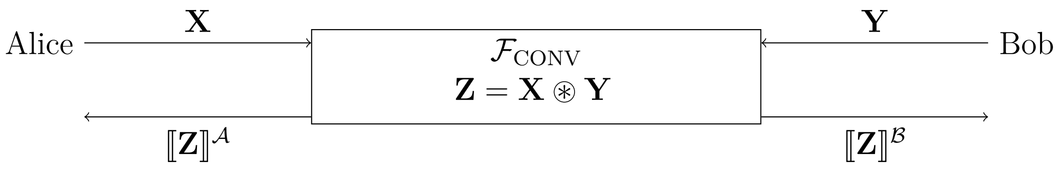
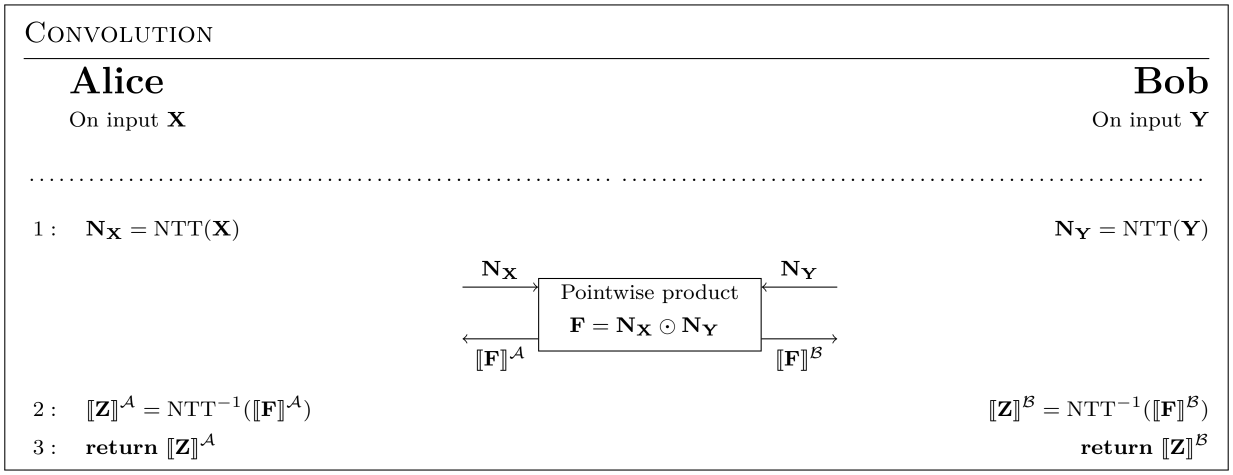

Alice
$\mathbf{X}$
$\mathcal{F}_{\text{CONV}}$
$\mathbf{Z} = \mathbf{X} \circledast \mathbf{Y}$
$\mathbf{Y}$
Bob
$[\![\mathbf{Z}]\!]^{\mathcal{A}}$
$[\![\mathbf{Z}]\!]^{\mathcal{B}}$

Figure 4.4. Secure two-party circular convolution functionality

Lemma 4.3 (Correctness and Security of Protocol 4.1)**.** *Protocol Convolution, on Alice's and Bob's respective private inputs* $\mathbf{X}$ *and* $\mathbf{Y}$*, securely computes the functionality* $([\![\mathbf{Z}]\!]^{\mathcal{A}}, [\![\mathbf{Z}]\!]^{\mathcal{B}}) = \mathcal{F}_{\text{CONV}}(\mathbf{X}, \mathbf{Y})$ *in the semi-honest threat model. Moreover, the protocol is unconditionally secure in the ABB model of computation.*


CONVOLUTION
Alice
On input $\mathbf{X}$
Bob
On input $\mathbf{Y}$
1 : $\mathbf{N_X} = \mathrm{NTT}(\mathbf{X})$
$\mathbf{N_Y} = \mathrm{NTT}(\mathbf{Y})$
$\mathbf{N_X}$
$\mathbf{N_Y}$
Pointwise product
$\mathbf{F} = \mathbf{N_X} \odot \mathbf{N_Y}$
$[\![\mathbf{F}]\!]^{\mathcal{A}}$
$[\![\mathbf{F}]\!]^{\mathcal{B}}$
2 : $[\![\mathbf{Z}]\!]^{\mathcal{A}} = \mathrm{NTT}^{-1}([\![\mathbf{F}]\!]^{\mathcal{A}})$
$[\![\mathbf{Z}]\!]^{\mathcal{B}} = \mathrm{NTT}^{-1}([\![\mathbf{F}]\!]^{\mathcal{B}})$
3 : **return** $[\![\mathbf{Z}]\!]^{\mathcal{A}}$
return $[\![\mathbf{Z}]\!]^{\mathcal{B}}$

Protocol 4.1. Secure realization of $([\![\mathbf{Z}^{\mathcal{A}}]\!], [\![\mathbf{Z}]\!]^{\mathcal{B}}) = \mathcal{F}_{\mathrm{CONV}}(\mathbf{X}, \mathbf{Y})$

Proof. The correctness follows immediately from the convolution theorem together with the linearity of Number Theoretic Transform; in particular,

$$\begin{aligned}
\mathbf{Z} &= [\![\mathbf{Z}]\!]^{\mathcal{A}} + [\![\mathbf{Z}]\!]^{\mathcal{B}} \\
&= \mathrm{NTT}^{-1}([\![\mathbf{F}]\!]^{\mathcal{A}}) + \mathrm{NTT}^{-1}([\![\mathbf{F}]\!]^{\mathcal{B}}) \\
&= \mathrm{NTT}^{-1}([\![\mathbf{F}]\!]^{\mathcal{A}} + [\![\mathbf{F}]\!]^{\mathcal{B}}) && \text{(Linearity of NTT)} \\
&= \mathrm{NTT}^{-1}(\mathbf{F}) \\
&= \mathrm{NTT}^{-1}(\mathbf{N_X} \odot \mathbf{N_X}) \\
&= \mathrm{NTT}^{-1}(\mathrm{NTT}(\mathbf{X}) \odot \mathrm{NTT}(\mathbf{Y})) \\
&= \mathbf{X} \circledast \mathbf{Y} && \text{(Convolution Theorem)}
\end{aligned}$$

Security of Protocol 4.1 follows from the fact that the only step with joint computations involves λ invocations of secure multiplication on (distinct) pairs of inputs, with outputs in split form; hence, each party receives only a uniform vector over $\mathbb{F}_q^{\lambda}$ that, in the absence of the other party's output, is statistically independent of $\mathbf{Z}$. Moreover, note that the output $[\![\mathbf{Z}]\!]$ being additively split is a crucial factor in maintaining the privacy of the input vectors. That is because if a party learns the vector $\mathbf{Z}$, combined with their input, she/he can use the convolution theorem to reconstruct the other party's input. □

Complexity. Protocol 4.1 requires one round of communication (in Step 1); moreover, its computation and communication complexities are, respectively, the $\mathcal{O}(\lambda \log \lambda)$ and $\mathcal{O}(\lambda)$.

Notation 4.5. Henceforth, any use of a unary functionality (or protocol) with a vector as its input denotes computing the intended functionality for each entry of the vector independently. For example, if $\mathbf{X}$ is length-λ vector, then $\mathcal{F}_{\text{EQZ}}([\![\mathbf{X}]\!]^{\mathcal{A}}, [\![\mathbf{X}]\!]^{\mathcal{B}})$ denotes $\mathcal{F}_{\text{EQZ}}([\![x_i]\!]^{\mathcal{A}}, [\![x_i]\!]^{\mathcal{B}})$ for all $1 \leq i \leq \lambda$.

4.5.2 Secure SWPM Protocols Using $\mathcal{F}_{\text{EQZ}}$

In this section, we present our protocols for securely computing all three variants of SWPM that securely deploy Algorithm 1 with a black-box access to the functionality $\mathcal{F}_{\text{EQZ}}$ (see Figure 3.2). The mutual first step in all protocols that follow is computing $[\![\mathbf{C}]\!] = [\![\hat{\mathbf{T}} * \hat{\mathbf{P}}^{rev}]\!]$ using Protocol 4.1.

Secure Protocol for $(\perp, \Gamma) = \mathcal{F}^{S}_{\mathbf{SWPM}}(\mathbf{T}, \mathbf{P})$

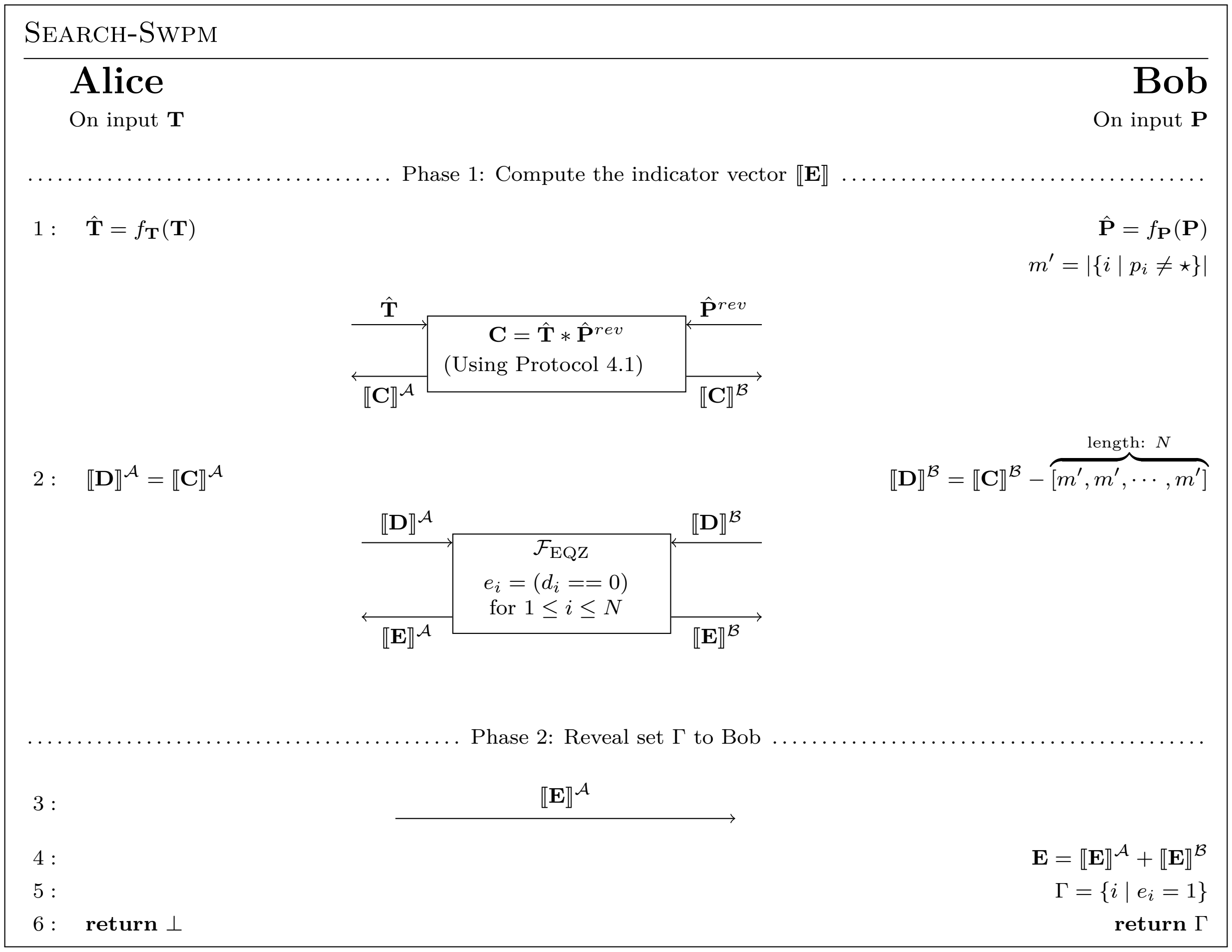
The following Protocol 4.2, Search-Swpm, allows Bob to learn only the set Γ containing all indices in the range $1 \leq i \leq N$ for which $\mathbf{T}_i \overset{\star}{=} \mathbf{P}$. The construction of $[\![\mathbf{C}]\!]$ ensures that $i \in \Gamma$ if and only if the score value $c_i = m'$, where m' is the count of non-wildcard characters in $\mathbf{P}$. m' is private to Bob and should not be revealed to Alice. Since subtraction in split form does not need any interaction among parties, Alice and Bob can simply obtain a modified score vector $[\![\mathbf{D}]\!]$ as follows:

$$[\![\mathbf{D}]\!]^{\mathcal{A}} = [\![\mathbf{C}]\!]^{\mathcal{A}} \qquad \text{and} \qquad [\![\mathbf{D}]\!]^{\mathcal{B}} = [\![\mathbf{C}]\!]^{\mathcal{B}} - \overbrace{[m', m', \cdots, m']}^{\text{length: } N} \tag{4.12}$$

Clearly,

$$[\![d_i]\!] = 0 \qquad \Longleftrightarrow \qquad [\![c_i]\!] = m' \tag{4.13}$$

Hence, it suffices for Alice and Bob to obliviously obtain an indicator vector $[\![\mathbf{E}]\!]$ such that $[\![e_i]\!] = [\![(d_i == 0)]\!]$ for $1 \leq i \leq N$. They achieve this through N independent (but, parallel)



SEARCH-SWPM
Alice
On input $\mathbf{T}$
Bob
On input $\mathbf{P}$
Phase 1: Compute the indicator vector $[\![\mathbf{E}]\!]$
1 : $\hat{\mathbf{T}} = f_{\mathbf{T}}(\mathbf{T})$
$\hat{\mathbf{P}} = f_{\mathbf{P}}(\mathbf{P})$
$m' = |\{i \mid p_i \neq \star\}|$
$\hat{\mathbf{T}}$
$\hat{\mathbf{P}}^{rev}$
$\mathbf{C} = \hat{\mathbf{T}} * \hat{\mathbf{P}}^{rev}$
(Using Protocol 4.1)
$[\![\mathbf{C}]\!]^{\mathcal{A}}$
$[\![\mathbf{C}]\!]^{\mathcal{B}}$
2 : $[\![\mathbf{D}]\!]^{\mathcal{A}} = [\![\mathbf{C}]\!]^{\mathcal{A}}$
$[\![\mathbf{D}]\!]^{\mathcal{B}} = [\![\mathbf{C}]\!]^{\mathcal{B}} - \overbrace{[m', m', \cdots, m']}^{\text{length: } N}$
$[\![\mathbf{D}]\!]^{\mathcal{A}}$
$[\![\mathbf{D}]\!]^{\mathcal{B}}$
$\mathcal{F}_{\mathrm{EQZ}}$
$e_i = (d_i == 0)$
for $1 \leq i \leq N$
$[\![\mathbf{E}]\!]^{\mathcal{A}}$
$[\![\mathbf{E}]\!]^{\mathcal{B}}$
Phase 2: Reveal set Γ to Bob
3 : $[\![\mathbf{E}]\!]^{\mathcal{A}}$
4 : $\mathbf{E} = [\![\mathbf{E}]\!]^{\mathcal{A}} + [\![\mathbf{E}]\!]^{\mathcal{B}}$
5 : $\Gamma = \{i \mid e_i = 1\}$
6 : **return** $\perp$
return Γ

Protocol 4.2. Secure realization of $(\perp, \Gamma) = \mathcal{F}^{S}_{\mathrm{SWPM}}(\mathbf{T}, \mathbf{P})$ based on $\mathcal{F}_{\mathrm{EQZ}}$

invocations of "equal to zero" functionality, i.e., $([\![e_i]\!]^{\mathcal{A}}, [\![e_i]\!]^{\mathcal{B}}) = \mathcal{F}_{\mathrm{EQZ}}([\![d_i]\!]^{\mathcal{A}}, [\![d_i]\!]^{\mathcal{B}})$. At this point, Alice sends $[\![\mathbf{E}]\!]^{\mathcal{A}}$ to Bob, and he construct the set

$$\Gamma = \{i \mid e_i = 1\} = \{i \mid d_i = 0\} \tag{4.14}$$

as desired.

Lemma 4.4 (Correctness and Security of Protocol 4.2)**.** *Protocol* SEARCH-SWPM*, on Alice's and Bob's respective private inputs* $\mathbf{T}$ *and* $\mathbf{P}$*, securely computes the functionality* $(\perp, \Gamma) = \mathcal{F}^{S}_{\mathrm{SWPM}}(\mathbf{T}, \mathbf{P})$ *in the semi-honest threat model. Moreover, given a black-box access to the* $\mathcal{F}_{\mathrm{EQZ}}$ *functionally, the protocol is unconditionally secure in the ABB model of computation.*

Proof. The correctness of Protocol 4.2 follows from the previously discussed correctness of Algorithm 1 based on the mapping functions $f_{\mathbf{T}}(\cdot)$ and $f_{\mathbf{P}}(\cdot)$ together with the construction of vector $[\![\mathbf{D}]\!]$ as summarized in Equation 4.12 that ensures $[\![d_i]\!] = 0$ for $1 \le i \le N$ if and only if $\mathbf{T}_i \overset{\star}{=} \mathbf{P}$.

The security of Phase 1 follows from the unconditional security of protocol Convolution in Step 1 and the black-box access to the ideal functionality $\mathcal{F}_{\text{EQZ}}$ used in the second step of this phase. Hence, to prove the overall security of Protocol 4.2, it is enough to argue that revealing the indicator vector $\mathbf{E}$ to Bob in Phase 2 does not provide him with any information about $\mathbf{T}$ that he would not be able to learn from his prescribed output $\Gamma = \{i \mid \mathbf{T}_i \overset{\star}{=} \mathbf{P}\}$. For that, it suffices to show that given Γ, Bob can construct the indicator vector $\mathbf{E}$ without any additional information or any help from Alice; Bob can obtain $\mathbf{E}$ from Γ by computing e_i for $1 \le i \le N$ as follows:

$$e_i = \begin{cases} 1 & \text{if } i \in \Gamma \\ 0 & \text{if } i \notin \Gamma \end{cases}$$

In other words, the vector $\mathbf{E}$ is just a representation of Bob's prescribed output Γ, and this completes the proof. $\square$

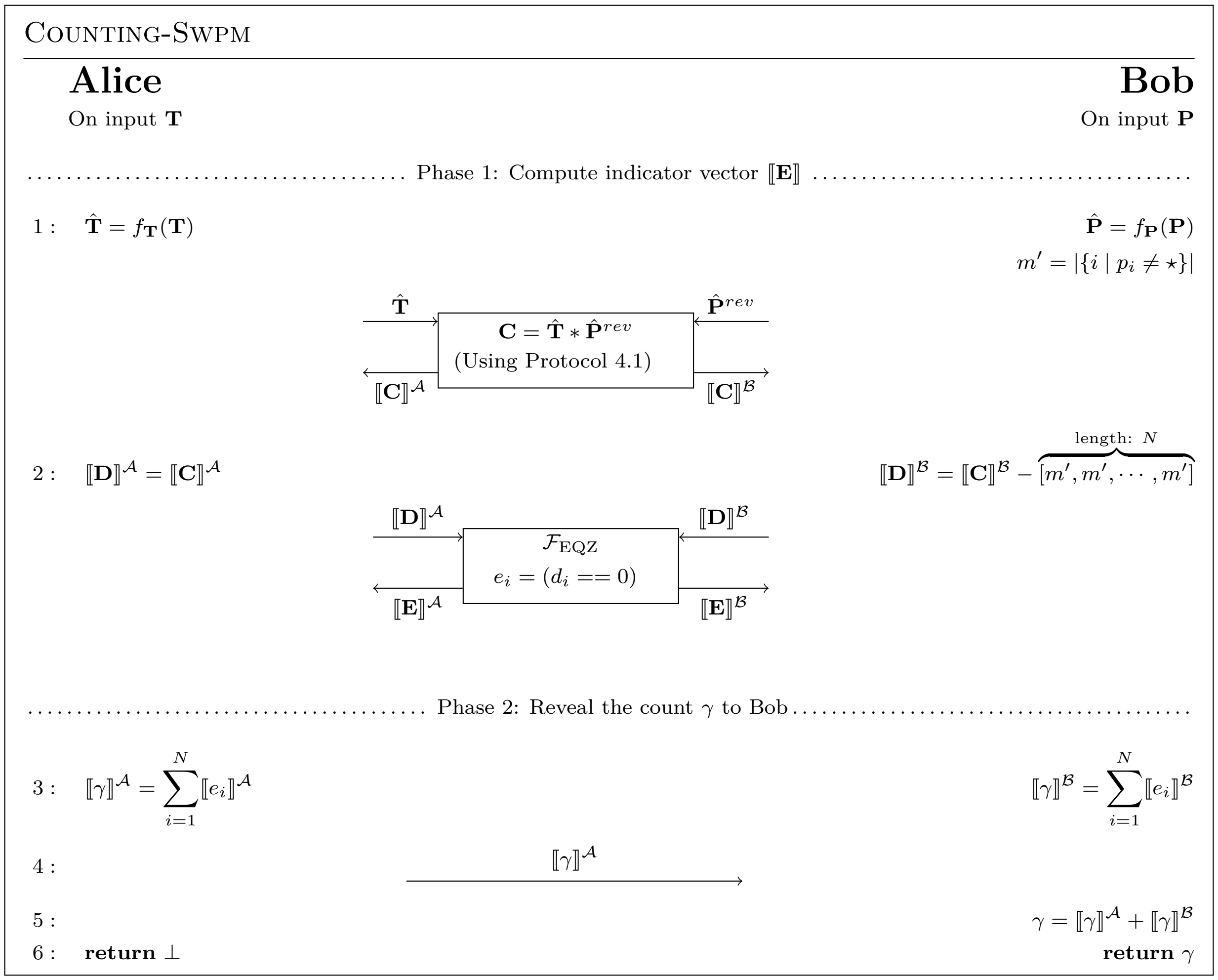
Complexity. Protocol 4.2 requires three rounds of communication; moreover, its computation and communication complexities are, respectively, $\mathcal{O}(n \log m)$ and $\mathcal{O}(n)$.

Secure Protocol for $(\perp, \gamma) = \mathcal{F}^{C}_{\mathbf{SWPM}}(\mathbf{T}, \mathbf{P})$

In order for Bob to learn the count $\gamma = |\Gamma|$, Alice and Bob first securely compute the indicator vector $[\![\mathbf{E}]\!]$ as discussed earlier. However, it is unacceptable if Alice sends her share $[\![\mathbf{E}]\!]^{\mathcal{A}}$ to Bob; doing so would reveal the set Γ to Bob while the functionality $\mathcal{F}^{C}_{\text{SWPM}}$ requires Bob to learn nothing about Γ other than its cardinality. Since $[\![e_i]\!] \in \{0, 1\}$, Alice and Bob can simply use the non-interactive property of addition in split form to efficiently address this issue. In particular, Alice and Bob compute $[\![\gamma]\!] = \sum_{i=1}^{n} [\![e_i]\!]$ as follows:

$$[\![\gamma]\!]^{\mathcal{A}} = \sum_{i=1}^{N} [\![e_i]\!]^{\mathcal{A}} \qquad \text{and} \qquad [\![\gamma]\!]^{\mathcal{B}} = \sum_{i=1}^{N} [\![e_i]\!]^{\mathcal{B}} \tag{4.15}$$

At this point, Alice sends her share $[\![\gamma]\!]^{\mathcal{A}}$ to Bob, and he reconstructs the desired value $\gamma = |\Gamma|$.



Counting-Swpm

```
Alice                                                   Bob
On input $\mathbf{T}$                                   On input $\mathbf{P}$

........ Phase 1: Compute indicator vector $[\![\mathbf{E}]\!]$ ........

1:  $\hat{\mathbf{T}} = f_{\mathbf{T}}(\mathbf{T})$      $\hat{\mathbf{P}} = f_{\mathbf{P}}(\mathbf{P})$
                                                        $m' = |\{i \mid p_i \neq \star\}|$
    $\hat{\mathbf{T}}$ -> [ $\mathbf{C} = \hat{\mathbf{T}} * \hat{\mathbf{P}}^{rev}$ (Using Protocol 4.1) ] <- $\hat{\mathbf{P}}^{rev}$
    $[\![\mathbf{C}]\!]^{\mathcal{A}}$ <-                                 -> $[\![\mathbf{C}]\!]^{\mathcal{B}}$
2:  $[\![\mathbf{D}]\!]^{\mathcal{A}} = [\![\mathbf{C}]\!]^{\mathcal{A}}$      $[\![\mathbf{D}]\!]^{\mathcal{B}} = [\![\mathbf{C}]\!]^{\mathcal{B}} - \overbrace{[m', m', \cdots, m']}^{\text{length: } N}$
    $[\![\mathbf{D}]\!]^{\mathcal{A}}$ -> [ $\mathcal{F}_{\mathrm{EQZ}}$ $e_i = (d_i == 0)$ ] <- $[\![\mathbf{D}]\!]^{\mathcal{B}}$
    $[\![\mathbf{E}]\!]^{\mathcal{A}}$ <-                                 -> $[\![\mathbf{E}]\!]^{\mathcal{B}}$

........ Phase 2: Reveal the count $\gamma$ to Bob ........

3:  $[\![\gamma]\!]^{\mathcal{A}} = \sum_{i=1}^{N} [\![e_i]\!]^{\mathcal{A}}$      $[\![\gamma]\!]^{\mathcal{B}} = \sum_{i=1}^{N} [\![e_i]\!]^{\mathcal{B}}$
4:                 $[\![\gamma]\!]^{\mathcal{A}}$ ->
5:                                                      $\gamma = [\![\gamma]\!]^{\mathcal{A}} + [\![\gamma]\!]^{\mathcal{B}}$
6:  return $\perp$                                      return $\gamma$
```

Protocol 4.3. Secure realization of $(\perp, \gamma) = \mathcal{F}^{C}_{\mathrm{SWPM}}(\mathbf{T}, \mathbf{P})$ based on $\mathcal{F}_{\mathrm{EQZ}}$

Lemma 4.5 (Correctness and Security of Protocol 4.3). *Protocol* Counting-Swpm, *on Alice's and Bob's respective private inputs* $\mathbf{T}$ *and* $\mathbf{P}$, *securely computes the functionality* $(\perp, \gamma) = \mathcal{F}^{C}_{\mathrm{SWPM}}(\mathbf{T}, \mathbf{P})$ *in the semi-honest threat model. Moreover, given a black-box access to the* $\mathcal{F}_{\mathrm{EQZ}}$ *functionally, the protocol is unconditionally secure in the ABB model of computation.*

Proof. For correctness, note that the first phase for computing the indicator vector $[\![\mathbf{E}]\!]$ is similar to that of Protocol 4.2. Thus, it suffices to argue that at the end of Phase 2, Bob learns $\gamma = |\Gamma|$. Recall from the construction of functions $f_{\mathbf{T}}(\cdot)$ and $f_{\mathbf{P}}(\cdot)$ in Example 4.2 that the prime modulo q is larger than $\max(n, \sigma)$ and hence larger than N. This choice of prime q together with the property that $[\![e_i]\!] = 1$ if and only if $\mathbf{T}_i \overset{\star}{=} \mathbf{P}$ ensures that the summation $\sum_{i=1}^{N} [\![\gamma]\!]$ in Step 3 does not result in a modular wraparound; this immediately implies that the result of this summation is guaranteed to be the count of entries in $[\![\mathbf{E}]\!]$ that are equal to 1; this completes the proof that Bob's output γ is indeed equal to $|\Gamma|$.

The security of Phase 1 follows from the unconditional security of protocol Convolution used in Step 1 and the black-box access to the ideal functionality $\mathcal{F}_{\text{EQZ}}$ used in the second step of this phase. Moreover, the information revealed to Bob in Phase 2 is exactly that of his prescribed output as in the functionality definition. Hence, it is impossible for Bob to learn anything about $\mathbf{T}$ other than his prescribed output. □

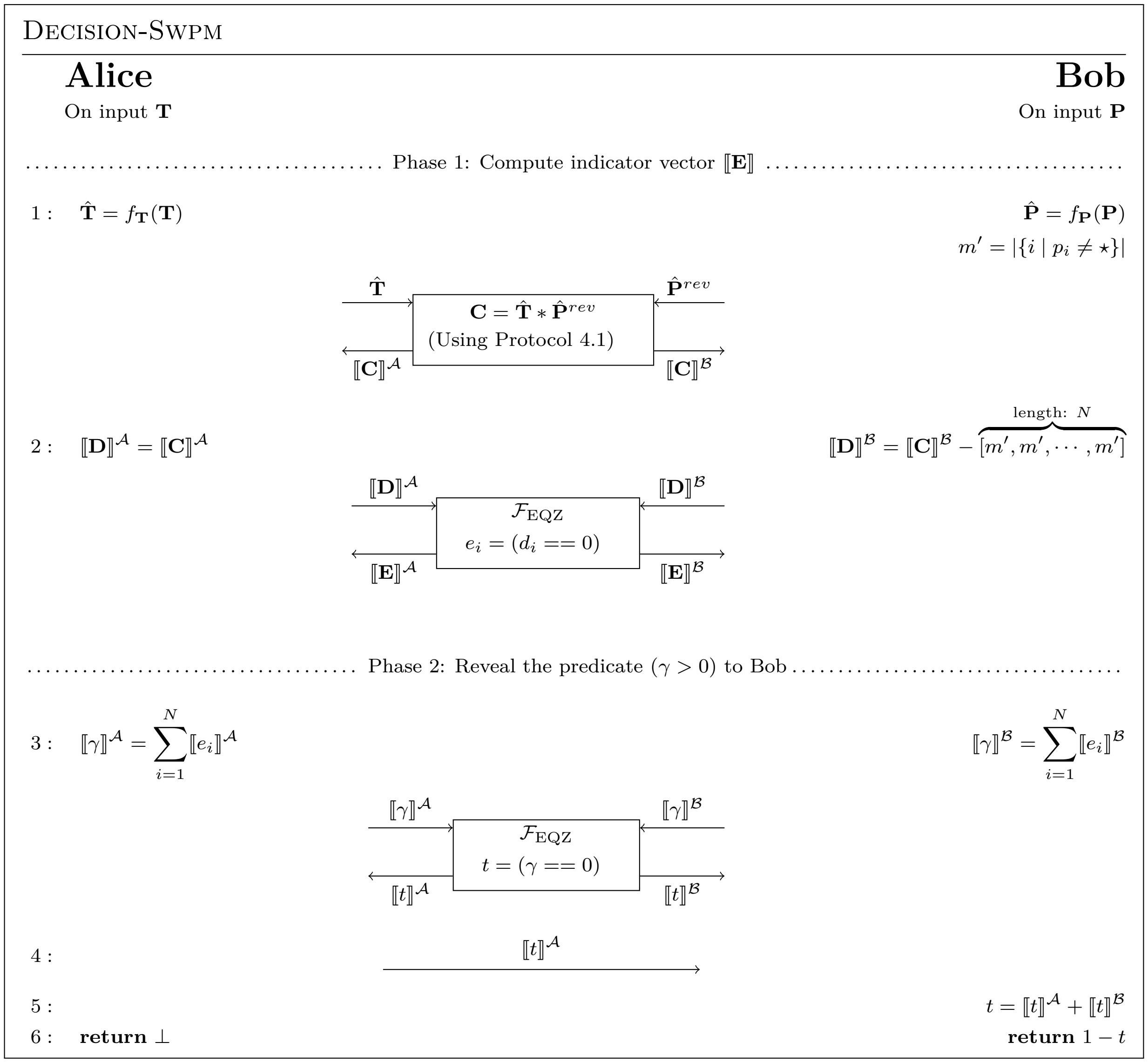
Complexity. Protocol 4.3 requires three rounds of communication; moreover, its computation and communication complexities are, respectively, $\mathcal{O}(n \log m)$ and $\mathcal{O}(n)$.

Secure Protocol for $(\perp, (\gamma > 0)) = \mathcal{F}^{D}_{\mathbf{SWPM}}(\mathbf{T}, \mathbf{P})$

In order for Bob to learn the predicate $(\gamma > 0)$, Alice and Bob first securely compute the count $[\![\gamma]\!]$ as discussed earlier. However, it is unacceptable if Alice sends her share $[\![\gamma]\!]^{\mathcal{A}}$ to Bob; doing so would reveal the count γ to Bob while the functionality $\mathcal{F}^{D}_{\text{SWPM}}$ requires Bob to learn nothing about γ other than whether it is zero or not. Hence, Alice and Bob need to invoke an instance of "equal to zero" functionality, i.e., $([\![t]\!]^{\mathcal{A}}, [\![t]\!]^{\mathcal{B}}) = \mathcal{F}_{\text{EQZ}}([\![\gamma]\!]^{\mathcal{A}}, [\![\gamma]\!]^{\mathcal{B}})$. At this point, Alice sends her share $[\![t]\!]^{\mathcal{A}}$ to Bob, and he reconstructs t and outputs the desired predicate value $(\gamma > 1) = 1 - t$.

Lemma 4.6 (Correctness and Security of Protocol 4.4)**.** *Protocol* Decision-Swpm, *on Alice's and Bob's respective private inputs* $\mathbf{T}$ *and* $\mathbf{P}$*, securely computes the functionality* $(\perp, (\gamma > 0)) = \mathcal{F}^{D}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ *in the semi-honest threat model. Moreover, given black-box access to the* $\mathcal{F}_{\text{EQZ}}$ *functionality, the protocol is unconditionally secure in the ABB model of computation.*

Proof. Both correctness and security of Protocol 4.4 follow a similar argument as in the proof of Lemma 4.5 for Protocol 4.3. This is because Protocol 4.4 is the same as Protocol


Decision-Swpm
Alice
On input $\mathbf{T}$
Bob
On input $\mathbf{P}$
Phase 1: Compute indicator vector $[\![\mathbf{E}]\!]$
1 : $\hat{\mathbf{T}} = f_{\mathbf{T}}(\mathbf{T})$
$\hat{\mathbf{P}} = f_{\mathbf{P}}(\mathbf{P})$
$m' = |\{i \mid p_i \neq \star\}|$
$\hat{\mathbf{T}}$
$\hat{\mathbf{P}}^{rev}$
$\mathbf{C} = \hat{\mathbf{T}} * \hat{\mathbf{P}}^{rev}$
(Using Protocol 4.1)
$[\![\mathbf{C}]\!]^{\mathcal{A}}$
$[\![\mathbf{C}]\!]^{\mathcal{B}}$
2 : $[\![\mathbf{D}]\!]^{\mathcal{A}} = [\![\mathbf{C}]\!]^{\mathcal{A}}$
$[\![\mathbf{D}]\!]^{\mathcal{B}} = [\![\mathbf{C}]\!]^{\mathcal{B}} - \overbrace{[m', m', \cdots, m']}^{\text{length: } N}$
$[\![\mathbf{D}]\!]^{\mathcal{A}}$
$[\![\mathbf{D}]\!]^{\mathcal{B}}$
$\mathcal{F}_{\text{EQZ}}$
$e_i = (d_i == 0)$
$[\![\mathbf{E}]\!]^{\mathcal{A}}$
$[\![\mathbf{E}]\!]^{\mathcal{B}}$
Phase 2: Reveal the predicate $(\gamma > 0)$ to Bob
3 : $[\![\gamma]\!]^{\mathcal{A}} = \sum_{i=1}^{N} [\![e_i]\!]^{\mathcal{A}}$
$[\![\gamma]\!]^{\mathcal{B}} = \sum_{i=1}^{N} [\![e_i]\!]^{\mathcal{B}}$
$[\![\gamma]\!]^{\mathcal{A}}$
$[\![\gamma]\!]^{\mathcal{B}}$
$\mathcal{F}_{\text{EQZ}}$
$t = (\gamma == 0)$
$[\![t]\!]^{\mathcal{A}}$
$[\![t]\!]^{\mathcal{B}}$
4 : $[\![t]\!]^{\mathcal{A}}$
5 : $t = [\![t]\!]^{\mathcal{A}} + [\![t]\!]^{\mathcal{B}}$
6 : **return** $\perp$
return $1 - t$

Protocol 4.4. Secure realization of $(\perp, (\gamma > 0)) = \mathcal{F}^{D}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ based on $\mathcal{F}_{\text{EQZ}}$

4.3 except that in Phase 2 it evaluates the predicate $(\gamma == 0)$ through a black-box access to the ideal functionality $\mathcal{F}_{\text{EQZ}}$ and reveals its value to Bob instead of revealing the count γ to him. $\square$

Complexity. Protocol 4.4 requires four rounds of communication; moreover, its computation and communication complexities are, respectively, $\mathcal{O}(n \log m)$ and $\mathcal{O}(n)$.

4.5.3 Secure SWPM Protocols Using Relaxed Equality Testing

In this section, we present another group of protocols for securely computing all three variants of SWPM. These protocols are built on the observation that in protocols 4.2, 4.3 and 4.4, Bob learns the outcome of (some) $\mathcal{F}_{\text{EQZ}}$ instances; this allows Alice and Bob to use a *relaxed* "equal to zero" functionality, $(\llbracket e \rrbracket^{\mathcal{A}}, \llbracket e \rrbracket^{\mathcal{B}}) = \mathcal{F}_{\text{REQZ}}(\llbracket s \rrbracket^{\mathcal{A}}, \llbracket s \rrbracket^{\mathcal{B}})$, as illustrated in Figure 4.5. In particular, the result of this relaxed equality test is as follows:

- If $\llbracket s \rrbracket = 0$, then $\llbracket e \rrbracket = 0$.
- If $\llbracket s \rrbracket \neq 0$, then $\llbracket e \rrbracket$ is a uniformly distributed non-zero random, i.e., $\llbracket e \rrbracket \overset{\$}{\leftarrow} \mathbb{F}_q \setminus \{0\}$.

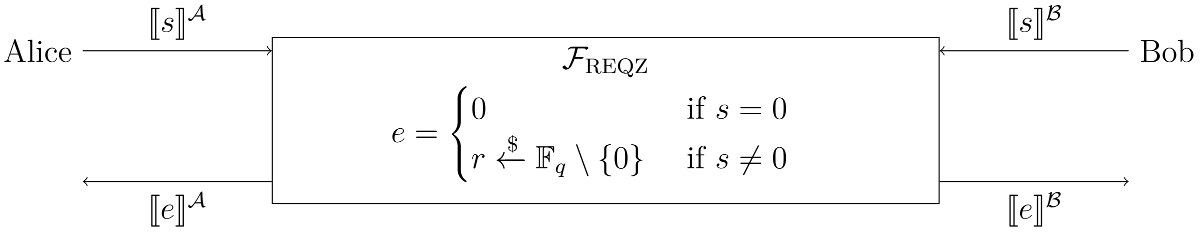
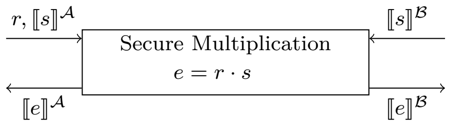

Alice
$\llbracket s \rrbracket^{\mathcal{A}}$
$\mathcal{F}_{\text{REQZ}}$
$e = \begin{cases} 0 & \text{if } s = 0 \\ r \overset{\$}{\leftarrow} \mathbb{F}_q \setminus \{0\} & \text{if } s \neq 0 \end{cases}$
$\llbracket s \rrbracket^{\mathcal{B}}$
Bob
$\llbracket e \rrbracket^{\mathcal{A}}$
$\llbracket e \rrbracket^{\mathcal{B}}$

Figure 4.5. $(\llbracket e \rrbracket^{\mathcal{A}}, \llbracket e \rrbracket^{\mathcal{B}}) = \mathcal{F}_{\text{REQZ}}(\llbracket s \rrbracket^{\mathcal{A}}, \llbracket s \rrbracket^{\mathcal{B}})$

Protocol 4.5, ZeroTest-Relaxed, securely computes the functionality $\mathcal{F}_{\text{REQZ}}$ through exactly one secure multiplication. It is enough to use a uniformly chosen non-zero random $r \overset{\$}{\leftarrow} \mathbb{F}_q \setminus \{0\}$ to compute $\llbracket e \rrbracket = \llbracket s \cdot r \rrbracket$.

ZeroTest-Relaxed

Alice	**Bob**
On input $\llbracket s \rrbracket^{\mathcal{A}}$	On input $\llbracket s \rrbracket^{\mathcal{B}}$
1 : $\llbracket r \rrbracket^{\mathcal{A}} \overset{\$}{\leftarrow} \mathbb{F}_q \setminus \{0\}$	$\llbracket r \rrbracket^{\mathcal{B}} = 0$


$r, \llbracket s \rrbracket^{\mathcal{A}}$
$\llbracket s \rrbracket^{\mathcal{B}}$
Secure Multiplication
$e = r \cdot s$
$\llbracket e \rrbracket^{\mathcal{A}}$
$\llbracket e \rrbracket^{\mathcal{B}}$

2 : **return** $\llbracket e \rrbracket^{\mathcal{A}}$	**return** $\llbracket e \rrbracket^{\mathcal{B}}$

Protocol 4.5. Secure realization of $(\llbracket e \rrbracket^{\mathcal{A}}, \llbracket e \rrbracket^{\mathcal{B}}) = \mathcal{F}_{\text{REQZ}}(\llbracket s \rrbracket^{\mathcal{A}}, \llbracket s \rrbracket^{\mathcal{B}})$

Remark 4.4. Note that in protocol ZeroTest-Relaxed Alice chooses the non-zero random r; hence, the test output $[\![e]\!]$ must never be revealed to her, otherwise, she learns the input s. In order to obtain a protocol after which the output can be revealed to either party, it suffices to instead use a split uniform non-zero random $[\![r]\!] \xleftarrow{\$} \mathbb{F}_q \setminus \{0\}$ that is unknown to both parties.

Lemma 4.7 (Correctness and Security of Protocol 4.5)**.** *Protocol ZeroTest-Relaxed, on Alice's and Bob's respective private inputs* $[\![s]\!]^{\mathcal{A}}$ *and* $[\![s]\!]^{\mathcal{B}}$*, securely computes the functionality* $([\![e]\!]^{\mathcal{A}}, [\![e]\!]^{\mathcal{B}}) = \mathcal{F}_{\text{REQZ}}([\![s]\!]^{\mathcal{A}}, [\![s]\!]^{\mathcal{B}})$ *in the semi-honest threat model. Moreover, the protocol is unconditionally secure in the ABB model of computation.*

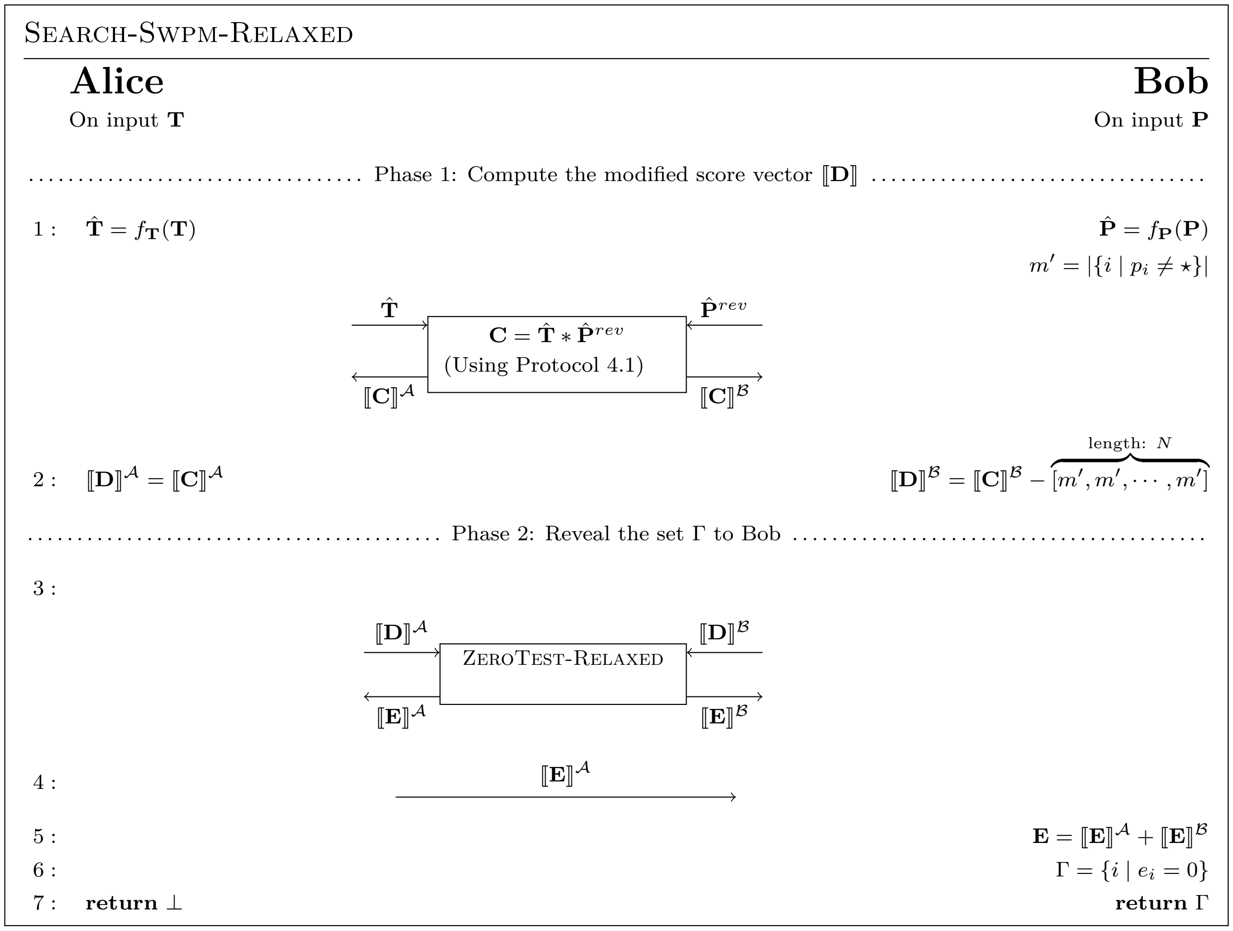
Proof. Correctness follows from the facts that multiplying the input by a uniform non-zero random i) preserves the "equal to zero" property, and ii) it uniformly distributes any non-zero input. Moreover, the protocol's security follows immediately from the security of multiplication in split form. □

Complexity. Protocol 4.5 requires 1 round of communication; moreover, both its computation and communication complexities are $\mathcal{O}(1)$.

The above secure realization of $\mathcal{F}_{\text{REQZ}}$ is much more efficient, in both computation and communication, compared to the existing solutions for $\mathcal{F}_{\text{EQZ}}$ (see Chapter 3). Thus, using this relaxed functionality provides a noticeable performance advantage; however, considering that the outcome of $\mathcal{F}_{\text{REQZ}}$ can be any random integer in $\mathbb{F}_q$ (rather than either 0 or 1), simply replacing $\mathcal{F}_{\text{EQZ}}$ with $\mathcal{F}_{\text{REQZ}}$ in protocols 4.2, 4.3 and 4.4 would result in incorrect solutions. Below, we present our modified protocols that use $\mathcal{F}_{\text{REQZ}}$ instead of $\mathcal{F}_{\text{EQZ}}$ for securely computing functionalities $\mathcal{F}^{S}_{\text{SWPM}}, \mathcal{F}^{C}_{\text{SWPM}}$ and $\mathcal{F}^{D}_{\text{SWPM}}$.

Secure Protocol for $(\perp, \Gamma) = \mathcal{F}^{S}_{\textbf{SWPM}}(\mathbf{T}, \mathbf{P})$

Protocol 4.6 below, Search-Swpm-Relaxed, securely computes $\mathcal{F}^{S}_{\text{SWPM}}$. This protocol is similar to Protocol 4.2 except that i) it uses $\mathcal{F}_{\text{REQZ}}$ instead of $\mathcal{F}_{\text{EQZ}}$, and ii) Bob constructs the set $\Gamma = \{i \mid e_i = 0\}$ rather than using Equation 4.14 (i.e., $\Gamma = \{i \mid e_i = 1\}$).


SEARCH-SWPM-RELAXED
Alice
On input $\mathbf{T}$
Bob
On input $\mathbf{P}$
Phase 1: Compute the modified score vector $[\![\mathbf{D}]\!]$
1 : $\hat{\mathbf{T}} = f_{\mathbf{T}}(\mathbf{T})$
$\hat{\mathbf{P}} = f_{\mathbf{P}}(\mathbf{P})$
$m' = |\{i \mid p_i \neq \star\}|$
$\hat{\mathbf{T}}$
$\hat{\mathbf{P}}^{rev}$
$\mathbf{C} = \hat{\mathbf{T}} * \hat{\mathbf{P}}^{rev}$
(Using Protocol 4.1)
$[\![\mathbf{C}]\!]^{\mathcal{A}}$
$[\![\mathbf{C}]\!]^{\mathcal{B}}$
2 : $[\![\mathbf{D}]\!]^{\mathcal{A}} = [\![\mathbf{C}]\!]^{\mathcal{A}}$
$[\![\mathbf{D}]\!]^{\mathcal{B}} = [\![\mathbf{C}]\!]^{\mathcal{B}} - \overbrace{[m', m', \cdots, m']}^{\text{length: } N}$
Phase 2: Reveal the set Γ to Bob
3 :
$[\![\mathbf{D}]\!]^{\mathcal{A}}$
$[\![\mathbf{D}]\!]^{\mathcal{B}}$
ZEROTEST-RELAXED
$[\![\mathbf{E}]\!]^{\mathcal{A}}$
$[\![\mathbf{E}]\!]^{\mathcal{B}}$
4 : $[\![\mathbf{E}]\!]^{\mathcal{A}}$
5 : $\mathbf{E} = [\![\mathbf{E}]\!]^{\mathcal{A}} + [\![\mathbf{E}]\!]^{\mathcal{B}}$
6 : $\Gamma = \{i \mid e_i = 0\}$
7 : **return** $\perp$
return Γ

Protocol 4.6. Secure realization of $(\perp, \Gamma) = \mathcal{F}^{S}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ based on $\mathcal{F}_{\text{REQZ}}$

Lemma 4.8 (Correctness and Security of Protocol 4.6)**.** *Protocol* SEARCH-SWPM-RELAXED*, on Alice's and Bob's respective private inputs* $\mathbf{T}$ *and* $\mathbf{P}$*, securely computes the functionality* $(\perp, \Gamma) = \mathcal{F}^{S}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ *in the semi-honest threat model. Moreover, the protocol is unconditionally secure in the ABB model of computation.*

Proof. Correctness follows a similar argument as in Lemma 4.5 except that in Phase 2 the vector $[\![\mathbf{E}]\!]$ is such that $[\![e_i]\!] = 0$ if and only if $\mathbf{T}_i \overset{\star}{=} \mathbf{P}$ for $1 \leq i \leq N$, based on which, in Step 6 Bob constructs the set Γ as follows:

$$\Gamma = \{i \mid e_i = 0\} \tag{4.16}$$

rather than using Equation 4.14. In particular, the resulting set Γ is as desired because

$$\begin{tikzcd} \llbracket c_i \rrbracket = m' \iff \llbracket d_i \rrbracket = 0 \iff \llbracket e_i \rrbracket = \llbracket r_i \cdot d_i \rrbracket = 0 \arrow[d, Rightarrow] \\ \Gamma = \{i \mid \llbracket c_i \rrbracket = m'\} = \{i \mid \llbracket d_i \rrbracket = 0\} = \{i \mid e_i = 0\} \end{tikzcd}$$

The security of Phase 1 follows from the unconditional security of protocol Convolution in Step 1 and the security of (non-interactive) addition in split form. Hence, to prove the overall security of Protocol 4.2, we only need to argue that revealing vector $\mathbf{E}$ to Bob in Phase 2 does not provide him with any information about $\mathbf{T}$ that he would not be able to learn from his prescribed output $\Gamma = \{i \mid \mathbf{T}_i \overset{\star}{=} \mathbf{P}\}$. For that, it suffices to show that given Γ, Bob can construct a vector $\hat{\mathbf{E}}$ that is perfectly indistinguishable from $\mathbf{E}$; Bob constructs such a vector $\hat{\mathbf{E}}$ from Γ without any additional information or any help from Alice by computing $\hat{e}_i$ for $1 \le i \le N$ as follows:

$$\hat{e}_i = \begin{cases} 0 & \text{if } i \in \Gamma \\ r \overset{\$}{\leftarrow} \mathbb{F}_q \setminus \{0\} & \text{if } i \notin \Gamma \end{cases}$$

The vectors $\hat{\mathbf{E}}$ and $\mathbf{E}$ are statistically identical because i) $\hat{e}_i = 0$ if and only if $e_i = 0$, and ii) any non-zero entry $\hat{e}_i$ and its corresponding entry e_i are both uniform integers in $\mathbb{F}_q \setminus \{0\}$. This implies that the vector $\mathbf{E}$ is just a representation of Bob's prescribed output Γ, and this completes the proof. □

Complexity. Protocol 4.6 requires three rounds of communication; moreover, its computation and communication complexities are, respectively, $\mathcal{O}(n \log m)$ and $\mathcal{O}(n)$.

Secure Protocol for $(\perp, \gamma) = \mathcal{F}^{C}_{\mathbf{SWPM}}(\mathbf{T}, \mathbf{P})$

Protocol 4.8 below, Counting-Swpm-Relaxed, extends Protocol 4.6 such that it hides both actual and relative matching positions (if any) while counting them remains possible. The key idea is that instead of revealing vector $\mathbf{E}$ to Bob, reveal to him a random permutation of $\mathbf{E}$. In other words, Protocol 4.8 provides Bob with $\mathbf{E}' = \pi(\mathbf{E})$, where $\pi : \{1, 2, \cdots, N\} \to$

$\{1, 2, \cdots, N\}$ is a uniformly chosen permutation that is unknown to him. Afterwards, Bob simply counts the number of zero entries in $\mathbf{E}'$ to learn the prescribed output $\gamma = |\Gamma|$. It therefore suffices to give a protocol that permutes the split vector $[\![\mathbf{E}]\!]$ according to such a permutation without revealing π to Bob. Note that Alice can know the permutation π because the resulting permuted vector will never be revealed to her. Moreover, due to the linearity of permutation

$$\pi(\mathbf{E}) = \pi([\![\mathbf{E}]\!]^{\mathcal{A}} + [\![\mathbf{E}]\!]^{\mathcal{B}}) = \pi([\![\mathbf{E}]\!]^{\mathcal{A}}) + \pi([\![\mathbf{E}]\!]^{\mathcal{B}}) \tag{4.17}$$

Hence, Alice can locally compute $\pi([\![\mathbf{E}]\!]^{\mathcal{A}})$, and it remains to give a secure protocol that computes $\pi([\![\mathbf{E}]\!]^{\mathcal{B}})$ without revealing π to Bob and $[\![\mathbf{E}]\!]^{\mathcal{B}}$ to Alice. We call such a permutation functionality *Single-blind Permutation*; below, we describe the required characteristics of single-blind permutation and propose Protocol 4.7 that securely computes it.

Single-blind Permutation

The single-blind permutation functionality (illustrated in Figure 4.6) takes as inputs, Alice's private (uniformly chosen) permutation $\pi : \{1, 2, \cdots, \lambda\} \to \{1, 2, \cdots, \lambda\}$ and Bob's private length-λ vector $\mathbf{X}$, and outputs $[\![\pi(\mathbf{X})]\!]$. It is crucial that the output is a split vector; otherwise, a party who would learn $\pi(\mathbf{X})$ may reconstruct the other party's input.

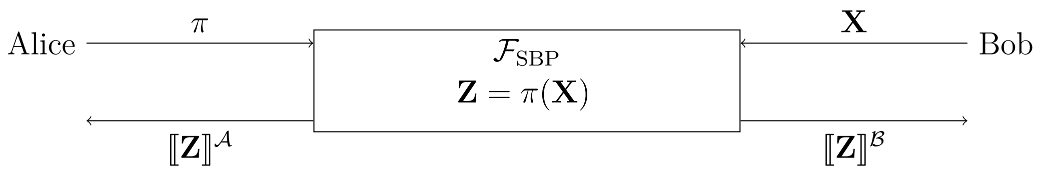

Alice
π
$\mathbf{X}$
Bob
$\mathcal{F}_{\text{SBP}}$
$\mathbf{Z} = \pi(\mathbf{X})$
$[\![\mathbf{Z}]\!]^{\mathcal{A}}$
$[\![\mathbf{Z}]\!]^{\mathcal{B}}$

Figure 4.6. Single-blind Permutation Functionality

Protocol 4.7 below, Single-Blind-Perm, is a lightweight protocol that is particularly designed for the use in the secret splitting framework. Our protocol requires auxiliary inputs consisting of correlated random length-λ vectors $\mathbf{R}^{\mathcal{A}} \xleftarrow{\$} \mathbb{F}_q^{\lambda}$ (private to Alice) and $\mathbf{R}^{\mathcal{B}} \xleftarrow{\$} \mathbb{F}_q^{\lambda}, \mathbf{U}^{\mathcal{B}} \in \mathbb{F}_q^{\lambda}$ (private to Bob) such that $\mathbf{U}^{\mathcal{B}} = \pi(\mathbf{R}^{\mathcal{A}} - \mathbf{R}^{\mathcal{B}})$. Note that these auxiliary inputs and the permutation π are independent of Bob's input vector $\mathbf{X}$; hence, Alice and Bob may choose $\pi, \mathbf{R}^{\mathcal{A}}$ and $\mathbf{R}^{\mathcal{B}}$, and compute $\mathbf{U}^{\mathcal{B}}$ ahead of time. Henceforth, we assume that these correlated vectors are provided to Alice and Bob ahead of time by a helper server that is trusted to not collude with either party.

```
SINGLE-BLIND-PERM
Alice                                                              Bob
On input $\pi$                                                     On input $\mathbf{X}$
Aux. input $\mathbf{R}^{\mathcal{A}}$                              Aux. inputs $\mathbf{R}^{\mathcal{B}}, \mathbf{U}^{\mathcal{B}}$
.........................................................................
1:                          $\xleftarrow{\mathbf{X}+\mathbf{R}^{\mathcal{B}}}$
2: $\mathbf{Y} = \pi(\mathbf{X}+\mathbf{R}^{\mathcal{B}}-\mathbf{R}^{\mathcal{A}})$
   $\mathbf{U}^{\mathcal{A}} \xleftarrow{\$} \mathbb{F}_q^{\lambda}$
                            $\xrightarrow{\mathbf{Y}-\mathbf{U}^{\mathcal{A}}}$
3: $[\![\mathbf{Z}]\!]^{\mathcal{A}} = \mathbf{U}^{\mathcal{A}}$        $[\![\mathbf{Z}]\!]^{\mathcal{B}} = \mathbf{Y}-\mathbf{U}^{\mathcal{A}}+\mathbf{U}^{\mathcal{B}}$
4: return $[\![\mathbf{Z}]\!]^{\mathcal{A}}$                           return $[\![\mathbf{Z}]\!]^{\mathcal{B}}$
```

Protocol 4.7. Secure realization of $([\![\mathbf{Z}]\!]^{\mathcal{A}}, [\![\mathbf{Z}]\!]^{\mathcal{A}}) = \mathcal{F}_{\text{SBP}}(\pi, \mathbf{X})$ using pre-computed auxiliary inputs $\mathbf{R}^{\mathcal{A}}$, $\mathbf{R}^{\mathcal{B}}$ and $\mathbf{U}^{\mathcal{B}} = \pi(\mathbf{R}^{\mathcal{A}} - \mathbf{R}^{\mathcal{B}})$

Lemma 4.9 (Correctness and Security of Protocol 4.7)**.** *Protocol* SINGLE-BLIND-PERM*, on Alice's private inputs π and Bob's private inputs $\mathbf{X}$ (as well as their respective auxiliary inputs $\mathbf{R}^{\mathcal{A}}$ and $\mathbf{R}^{\mathcal{B}}, \mathbf{U}^{\mathcal{B}}$), securely computes the functionality $([\![\mathbf{Z}]\!]^{\mathcal{A}}, [\![\mathbf{Z}]\!]^{\mathcal{B}}) = \mathcal{F}_{\text{FSB}}(\pi, \mathbf{X})$ in the semi-honest threat model. Moreover, the protocol is unconditionally secure in the ABB model of computation.*

Proof. The correctness of Protocol 4.7 follows from the linearity of permutation. We show that the sum of Alice's and Bob's respective outputs, $[\![\mathbf{Z}]\!]^{\mathcal{A}} = \mathbf{U}^{\mathcal{A}}$ and $[\![\mathbf{Z}]\!]^{\mathcal{B}} = \mathbf{Y} - \mathbf{U}^{\mathcal{A}} + \mathbf{U}^{\mathcal{B}}$, is in fact equal to $\pi(\mathbf{X})$:

$$\begin{aligned}
\mathbf{Z} &= [\![\mathbf{Z}]\!]^{\mathcal{A}} + [\![\mathbf{Z}]\!]^{\mathcal{B}} \\
&= \mathbf{U}^{\mathcal{A}} + \mathbf{Y} - \mathbf{U}^{\mathcal{A}} + \mathbf{U}^{\mathcal{B}} \\
&= \mathbf{Y} + \mathbf{U}^{\mathcal{B}} \\
&= \pi(\mathbf{X} + \mathbf{R}^{\mathcal{B}} - \mathbf{R}^{\mathcal{A}}) + \mathbf{U}^{\mathcal{B}} && (\mathbf{Y} = \pi(\mathbf{X} + \mathbf{R}^{\mathcal{B}} - \mathbf{R}^{\mathcal{A}})) \\
&= \pi(\mathbf{X} + \mathbf{R}^{\mathcal{B}} - \mathbf{R}^{\mathcal{A}}) + \pi(\mathbf{R}^{\mathcal{A}} - \mathbf{R}^{\mathcal{B}}) && (\mathbf{U}^{\mathcal{B}} = \pi(\mathbf{R}^{\mathcal{A}} - \mathbf{R}^{\mathcal{B}})) \\
&= \pi(\mathbf{X} + \mathbf{R}^{\mathcal{B}} - \mathbf{R}^{\mathcal{A}} + \mathbf{R}^{\mathcal{A}} - \mathbf{R}^{\mathcal{B}}) && (\text{Linearity of } \pi) \\
&= \pi(\mathbf{X})
\end{aligned}$$

Security of Protocol 4.7 follows from the unconditional security of (additive) one-time pad encryption. Note that Alice receives only $\mathbf{X} + \mathbf{R}^{\mathcal{B}}$ that is indistinguishable from a uniformly chosen random vector to her (because $\mathbf{R}^{\mathcal{B}} \overset{\$}{\leftarrow} \mathbb{F}_q^{\lambda}$ is private to Bob). On the other hand, Bob receives only $\mathbf{Y} - \mathbf{U}^{\mathcal{A}}$ that is indistinguishable from a uniformly chosen random vector to him (because $\mathbf{U}^{\mathcal{A}} \overset{\$}{\leftarrow} \mathbb{F}_q^{\lambda}$ is private to Alice). Although Bob knows $\mathbf{U}^{\mathcal{B}} = \pi(\mathbf{R}^{\mathcal{A}} - \mathbf{R}^{\mathcal{B}})$ and $\mathbf{R}^{\mathcal{B}}$, he cannot learn anything about either π or $\mathbf{R}^{\mathcal{A}}$ because both these objects are chosen uniformly at random. □

Complexity. Protocol 4.7 requires two rounds of communication; moreover, its computation and communication complexities are both $\mathcal{O}(\lambda)$. Note that the protocol uses only addition and does not involve any (secure) multiplications.

Observation 4.1. Given that Alice and Bob can use the protocol SINGLE-BLIND-PERM to securely compute the functionality $([\![\mathbf{Z}]\!]^{\mathcal{A}}, [\![\mathbf{Z}]\!]^{\mathcal{B}}) = \mathcal{F}_{\text{SBP}}(\pi, [\![\mathbf{E}]\!]^{\mathcal{A}})$, they can securely obtain $[\![\mathbf{E}']\!] = [\![\pi(\mathbf{E})]\!]$ by obtaining its private shares as follows:

$$[\![\mathbf{E}']\!]^{\mathcal{A}} = \pi([\![\mathbf{E}]\!]^{\mathcal{A}}) + [\![\mathbf{Z}]\!]^{\mathcal{A}} \qquad \text{and} \qquad [\![\mathbf{E}']\!]^{\mathcal{B}} = [\![\mathbf{Z}]\!]^{\mathcal{B}}$$

Proof. Below, we show that $\mathbf{E}'$ is indeed equal to $\pi(\mathbf{E})$:

$$\begin{aligned}
\mathbf{E}' &= [\![\mathbf{E}']\!]^{\mathcal{A}} + [\![\mathbf{E}']\!]^{\mathcal{B}} \\
&= \Big(\pi([\![\mathbf{E}]\!]^{\mathcal{A}}) + [\![\mathbf{Z}]\!]^{\mathcal{A}}\Big) + [\![\mathbf{Z}]\!]^{\mathcal{B}} \\
&= \pi([\![\mathbf{E}]\!]^{\mathcal{A}}) + \Big([\![\mathbf{Z}]\!]^{\mathcal{A}} + [\![\mathbf{Z}]\!]^{\mathcal{B}}\Big) \\
&= \pi([\![\mathbf{E}]\!]^{\mathcal{A}}) + \pi([\![\mathbf{E}]\!]^{\mathcal{B}}) && \Big(\pi([\![\mathbf{E}]\!]^{\mathcal{B}}) = [\![\mathbf{Z}]\!]^{\mathcal{A}} + [\![\mathbf{Z}]\!]^{\mathcal{B}}\Big) \\
&= \pi([\![\mathbf{E}]\!]^{\mathcal{A}} + [\![\mathbf{E}]\!]^{\mathcal{B}}) && \text{(Linearity of } \pi) \\
&= \pi(\mathbf{E})
\end{aligned}$$

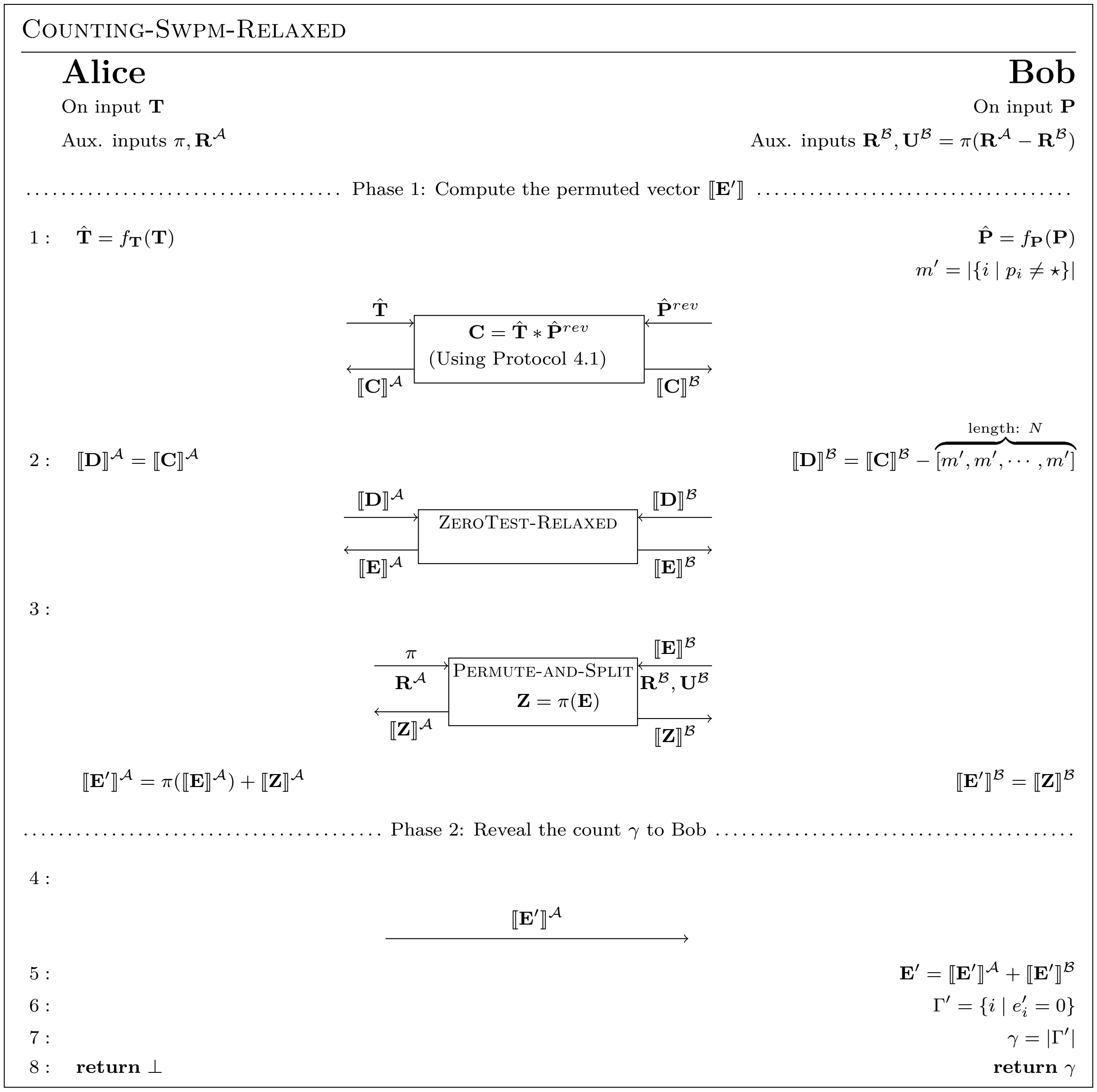
□

Protocol 4.8 uses Observation 4.1 to extend Protocol 4.6 for securely computing the $\mathcal{F}^{C}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ functionality.

Lemma 4.10 (Correctness and Security of Protocol 4.8)**.** *Protocol* Counting-Swpm-Relaxed*, on Alice's and Bob's respective private inputs* $\mathbf{T}$ *and* $\mathbf{P}$*, securely computes the functionality* $(\perp, \gamma) = \mathcal{F}^{C}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ *in the semi-honest threat model. Moreover, the protocol is unconditionally secure in the ABB model of computation.*

Proof. Correctness of Protocol 4.8 follows from Observation 4.1 that states $\mathbf{E}' = \pi(\mathbf{E})$. As a result, the count of zeros in $\mathbf{E}'$ is equal to the count of zeros in $\mathbf{E}$; hence, the computed output $\gamma = |\Gamma'|$ is indeed equal to Bob's prescribed output $|\Gamma|$. For the proof of security, note that Protocol 4.8 is a modification of of Protocol 4.6 such that it reveals $\mathbf{E}' = \pi(\mathbf{E})$ to Bob (rather than revealing $\mathbf{E}$); moreover, π is a uniform permutation that is unknown to Bob. Hence, it suffices to show that given the prescribed output γ, Bob can obtain a vector $\hat{\mathbf{E}}$ that is perfectly indistinguishable from $\mathbf{E}'$. To do so, Bob fixes a set of γ random indices $I = \{i_1, i_2, \cdots, i_\gamma\}$ such that $1 \leq i_1 < i_2 < \cdots < i_\gamma \leq N$; then, he constructs $\hat{\mathbf{E}}$ without any additional information or any help from Alice by computing $\hat{e}_i$ for $1 \leq i \leq N$ as follows:

$$\hat{e}_i = \begin{cases} 0 & \text{if } i \in I \\ r \xleftarrow{\$} \mathbb{F}_q \setminus \{0\} & \text{if } i \notin I \end{cases}$$



COUNTING-SWPM-RELAXED
Alice
On input $\mathbf{T}$
Aux. inputs $\pi, \mathbf{R}^{\mathcal{A}}$
Bob
On input $\mathbf{P}$
Aux. inputs $\mathbf{R}^{\mathcal{B}}, \mathbf{U}^{\mathcal{B}} = \pi(\mathbf{R}^{\mathcal{A}} - \mathbf{R}^{\mathcal{B}})$
Phase 1: Compute the permuted vector $[\![\mathbf{E}']\!]$
1 : $\hat{\mathbf{T}} = f_{\mathbf{T}}(\mathbf{T})$ | $\hat{\mathbf{P}} = f_{\mathbf{P}}(\mathbf{P})$
$m' = |\{i \mid p_i \neq \star\}|$
$\hat{\mathbf{T}}$ → $\mathbf{C} = \hat{\mathbf{T}} * \hat{\mathbf{P}}^{rev}$ (Using Protocol 4.1) ← $\hat{\mathbf{P}}^{rev}$
← $[\![\mathbf{C}]\!]^{\mathcal{A}}$ | $[\![\mathbf{C}]\!]^{\mathcal{B}}$ →
2 : $[\![\mathbf{D}]\!]^{\mathcal{A}} = [\![\mathbf{C}]\!]^{\mathcal{A}}$ | $[\![\mathbf{D}]\!]^{\mathcal{B}} = [\![\mathbf{C}]\!]^{\mathcal{B}} - \overbrace{[m', m', \cdots, m']}^{\text{length: } N}$
$[\![\mathbf{D}]\!]^{\mathcal{A}}$ → ZEROTEST-RELAXED ← $[\![\mathbf{D}]\!]^{\mathcal{B}}$
← $[\![\mathbf{E}]\!]^{\mathcal{A}}$ | $[\![\mathbf{E}]\!]^{\mathcal{B}}$ →
3 :
π, $\mathbf{R}^{\mathcal{A}}$ → PERMUTE-AND-SPLIT $\mathbf{Z} = \pi(\mathbf{E})$ ← $[\![\mathbf{E}]\!]^{\mathcal{B}}$, $\mathbf{R}^{\mathcal{B}}, \mathbf{U}^{\mathcal{B}}$
← $[\![\mathbf{Z}]\!]^{\mathcal{A}}$ | $[\![\mathbf{Z}]\!]^{\mathcal{B}}$ →
$[\![\mathbf{E}']\!]^{\mathcal{A}} = \pi([\![\mathbf{E}]\!]^{\mathcal{A}}) + [\![\mathbf{Z}]\!]^{\mathcal{A}}$ | $[\![\mathbf{E}']\!]^{\mathcal{B}} = [\![\mathbf{Z}]\!]^{\mathcal{B}}$
Phase 2: Reveal the count γ to Bob
4 : $[\![\mathbf{E}']\!]^{\mathcal{A}}$ →
5 : $\mathbf{E}' = [\![\mathbf{E}']\!]^{\mathcal{A}} + [\![\mathbf{E}']\!]^{\mathcal{B}}$
6 : $\Gamma' = \{i \mid e'_i = 0\}$
7 : $\gamma = |\Gamma'|$
8 : **return** $\perp$ | **return** γ

Protocol 4.8. Secure realization of $(\perp, \gamma) = \mathcal{F}^{C}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ based on $\mathcal{F}_{\text{REQZ}}$

The vectors $\hat{\mathbf{E}}$ and $\mathbf{E}'$ are statistically identical because i) the count of zeros in both is equal to γ, ii) any non-zero entry in either vector is a uniform integers in $\mathbb{F}_q \setminus \{0\}$, and iii) $\mathbf{E}'$ is a uniform permutation of $\mathbf{E}$; this completes the proof. □

Complexity. Protocol 4.8 requires five rounds of communication; moreover, its computation and communication complexities are, respectively, $\mathcal{O}(n \log m)$ and $\mathcal{O}(n)$.

Secure Protocol for $(\perp, (\gamma > 0)) = \mathcal{F}^{D}_{\mathbf{SWPM}}(\mathbf{T}, \mathbf{P})$

In this case, Bob only gets to learn whether or not $\mathbf{P}$ occurs in $\mathbf{T}$ at least once. Clearly, $\prod_{i=1}^{N} [\![d_i]\!] = 0$ if and only if $\mathbf{P}$ appears in $\mathbf{T}$. However, an unacceptably high number of rounds would be required if Alice and Bob were to compute the predicate value $(\gamma > 0)$ through securely evaluating $1 - \prod_{i=1}^{N} [\![d_i]\!]$, and revealing its result to Bob. In what follows, we propose Protocol 4.10, Decision-Swpm-Relaxed, that achieves a small constant number of rounds at the expense of allowing Bob to learn a loose upper bound on the count γ. In particular, Protocol 4.10 extends Protocol 4.8 such that Bob learns either $\gamma = 0$ (in case of no matches) or an upper bound on γ (if there exists at least one match). The key idea in Protocol 4.10 is to create an *augmented* vector $[\![\mathbf{E}']\!]$, such that

- It contains vector $[\![\mathbf{E}]\!]$ as a subsequence.
- It contains some *fake* entries that are either *chaff* or *clones*. A chaff entry is a fake element injected to the inputs and/or outputs to create artificial samples that are indistinguishable from the original entries [90], [91]. Here, a chaff entry is either a random non-zero integer (fake no-match) or a zero (fake match). On the other hand, a clone is a copy of an entry of $[\![\mathbf{E}]\!]$ such that multiple replicas of the same entry are split differently so they cannot be tied to each other.
- For any entry $[\![e'_i]\!]$, Bob does not know if it is a chaff or a copy of some real entry $[\![e_j]\!]$.

Let θ (chosen by Alice) and ν be, respectively, the number of chaff entries that are 0 and the total count of zeros in $[\![\mathbf{E}']\!]$. Clearly, $\theta = \nu$ if and only if $\gamma = 0$; if $\gamma > 0$, then ν is a loose upper bound on γ because not only does ν count the original matches, but also it counts the fake matches created by chaffing and cloning. Hence, it suffices for Bob to learn ν; then, Alice and Bob use $\mathcal{F}_{\text{REQZ}}$ to obliviously check whether or not $[\![\nu - \theta]\!] = 0$.

Chaffing and Cloning

The main difficulty in the above-mentioned approach is to obtain the augmented vector $[\![\mathbf{E}']\!]$ in a manner that Bob learns nothing about $[\![\mathbf{E}]\!]$ and the position of fake/real entries in it. Below, we describe how Alice and Bob achieve this.

- Alice chooses chaffing parameters $c \geq N$ and $\theta \xleftarrow{\$} \{1, 2, \cdots, c\}$, as well as a cloning parameter $r > 1$; she sends r and c (but not θ) to Bob.

- Alice generates a length-c vector **Chaff**, where exactly θ entries in it are equal to 0 and the other $c - \theta$ entries are random non-zeros.

- Alice and Bob (non-interactively) construct an intermediary vector $[\![\mathbf{E}'']\!]$ of length length $t = rN + c$ such that

$$[\![\mathbf{E}'']\!]^{\mathcal{A}} = \overbrace{[\![\mathbf{E}]\!]^{\mathcal{A}} || [\![\mathbf{E}]\!]^{\mathcal{A}} || \cdots || [\![\mathbf{E}]\!]^{\mathcal{A}}}^{r \text{ copies}} \, || \, \overbrace{\mathbf{Chaff}}^{\text{length: } c} \tag{4.18}$$

$$[\![\mathbf{E}'']\!]^{\mathcal{B}} = \overbrace{[\![\mathbf{E}]\!]^{\mathcal{B}} || [\![\mathbf{E}]\!]^{\mathcal{B}} || \ldots || [\![\mathbf{E}]\!]^{\mathcal{B}}}^{r \text{ copies}} \, || \, \overbrace{[0, 0, ..., 0]}^{\text{length: } c} \tag{4.19}$$

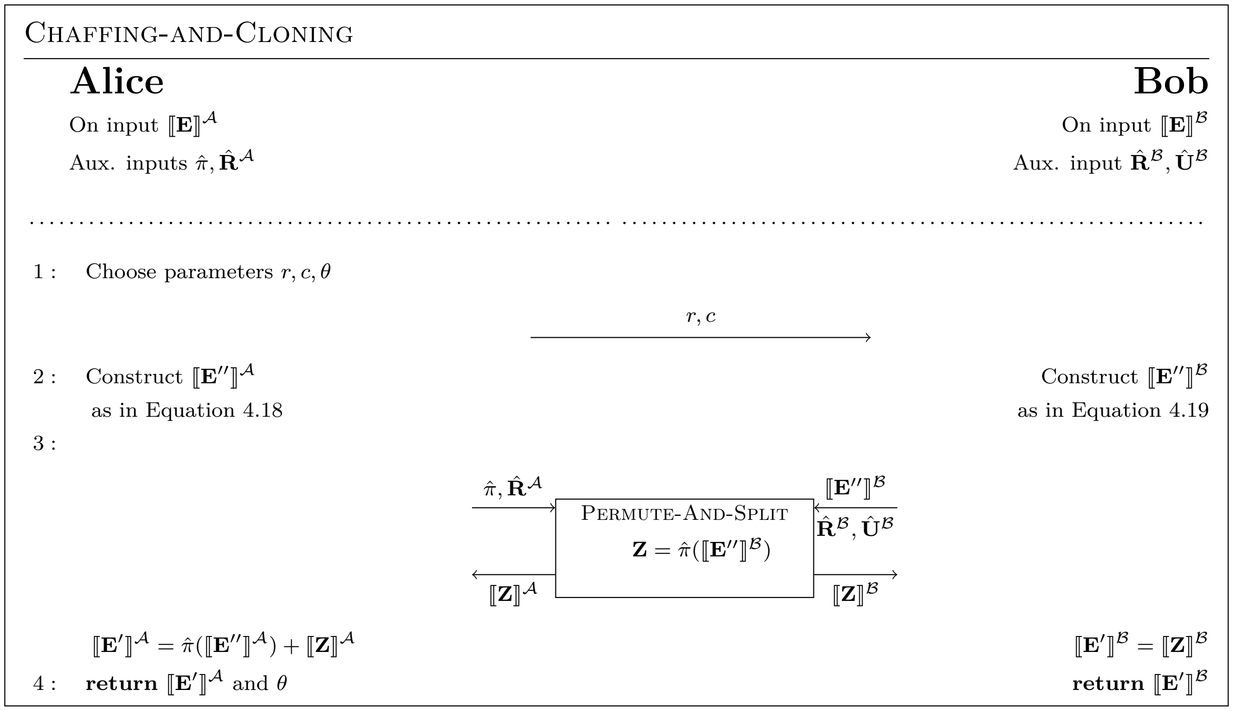
- Alice and Bob securely obtain $[\![\mathbf{E}']\!] = [\![\hat{\pi}(\mathbf{E}'')]\!]$, where $\hat{\pi} : \{1, 2, \cdots, t\} \to \{1, 2, \cdots, t\}$ is a uniform permutation that is chosen by Alice and unknown to Bob.

Lemma 4.11 (Correctness and Security of Protocol 4.9)**.** *Protocol* Chaffing-and-Cloning*, on Alice's and Bob's respective private inputs $[\![\mathbf{E}]\!]^{\mathcal{A}}$ and $[\![\mathbf{E}]\!]^{\mathcal{B}}$, securely constructs an augmented vector $[\![\mathbf{E}']\!]$ with the desired properties in the semi-honest threat model. Moreover, the protocol is unconditionally secure in the ABB model of computation.*

Proof. The fact that the resulting $[\![\mathbf{E}']\!]$ has all desired properties follows from the construction of the intermediary vector $[\![\mathbf{E}'']\!]$ and the uniform choice of permutation $\hat{\pi}$. Particularly, Alice and Bob (non-interactively) construct $[\![\mathbf{E}'']\!]$ as in equations 4.18 and 4.19 such that for $r > 1$ and $c \geq N$

$$[\![\mathbf{E}'']\!] = \overbrace{[\![\mathbf{E}]\!] || [\![\mathbf{E}]\!] || \ldots || [\![\mathbf{E}]\!]}^{r \text{ copies}} \, || [\![\overbrace{\mathbf{Chaff}}^{\text{length: } c}]\!] \tag{4.20}$$

Then, they obliviously permute and re-split it to obtain $\mathbf{E}'$. Not only is it guaranteed that $[\![\mathbf{E}']\!]$ includes $[\![\mathbf{E}]\!]$ as a subsequence, but also the uniform choice of permutation $\hat{\pi}$ (unknown to Bob) ensures that all entries are in an arbitrary order; thus, Bob cannot determine whether



CHAFFING-AND-CLONING
Alice
On input $[\![\mathbf{E}]\!]^{\mathcal{A}}$
Aux. inputs $\hat{\pi}, \hat{\mathbf{R}}^{\mathcal{A}}$
Bob
On input $[\![\mathbf{E}]\!]^{\mathcal{B}}$
Aux. input $\hat{\mathbf{R}}^{\mathcal{B}}, \hat{\mathbf{U}}^{\mathcal{B}}$
1 : Choose parameters r, c, θ
r, c
2 : Construct $[\![\mathbf{E}'']\!]^{\mathcal{A}}$ as in Equation 4.18
Construct $[\![\mathbf{E}'']\!]^{\mathcal{B}}$ as in Equation 4.19
3 :
$\hat{\pi}, \hat{\mathbf{R}}^{\mathcal{A}}$
$[\![\mathbf{E}'']\!]^{\mathcal{B}}$
$\hat{\mathbf{R}}^{\mathcal{B}}, \hat{\mathbf{U}}^{\mathcal{B}}$
PERMUTE-AND-SPLIT
$\mathbf{Z} = \hat{\pi}([\![\mathbf{E}'']\!]^{\mathcal{B}})$
$[\![\mathbf{Z}]\!]^{\mathcal{A}}$
$[\![\mathbf{Z}]\!]^{\mathcal{B}}$
$[\![\mathbf{E}']\!]^{\mathcal{A}} = \hat{\pi}([\![\mathbf{E}'']\!]^{\mathcal{A}}) + [\![\mathbf{Z}]\!]^{\mathcal{A}}$
$[\![\mathbf{E}']\!]^{\mathcal{B}} = [\![\mathbf{Z}]\!]^{\mathcal{B}}$
4 : **return** $[\![\mathbf{E}']\!]^{\mathcal{A}}$ and θ
return $[\![\mathbf{E}']\!]^{\mathcal{B}}$

Protocol 4.9. Secure construction of the augmented vector $\mathbf{E}'$ by obliviously injecting chaff and clone entries

or not an entry is a chaff or a replica of an entry in $[\![\mathbf{E}]\!]$. The security of Protocol 4.9 follows immediately from the previously discussed security of protocol PERMUTE-AND-SPLIT. □

Complexity. Protocol 4.9 requires three rounds of communication; moreover, its computation and communication complexities are both $\mathcal{O}(rN + c)$, where r and c are, respectively, the cloning and chaffing parameters.

Protocol 4.10 uses Protocol CHAFFING-AND-CLONING to extend Protocol 4.8 for securely computing the $\mathcal{F}^{D}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ functionality such that in the end Bob learns either $\gamma = 0$ or just a loose upper bound on the non-zero count γ.

Lemma 4.12 (Correctness and Security of Protocol 4.10)**.** *Protocol* DECISION-SWPM-RELAXED, *on Alice's and Bob's respective private inputs* $\mathbf{T}$ *and* $\mathbf{P}$, *approximates the functionality* $(\perp, (\gamma > 0)) = \mathcal{F}^{D}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ *in the sense that Bob learns either that* $\gamma = 0$, *or if* $\gamma > 0$, *he learns only an upper bound on* γ. *Moreover, the protocol is secure in the semi-honest threat model.*

Decision-Swpm-Relaxed

Alice | **Bob**

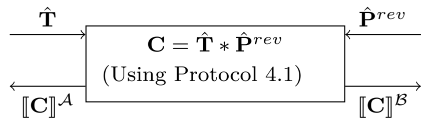
Alice: On input $\mathbf{T}$; Aux. inputs $\hat{\pi}, \hat{\mathbf{R}}^{\mathcal{A}}$

Bob: On input $\mathbf{P}$; Aux. inputs $\hat{\mathbf{R}}^{\mathcal{B}}, \hat{\mathbf{U}}^{\mathcal{B}} = \hat{\pi}(\mathbf{R}^{\mathcal{A}} - \mathbf{R}^{\mathcal{B}})$

.... Phase 1: Compute the augmented vector $[\![\mathbf{E}']\!]$ ....

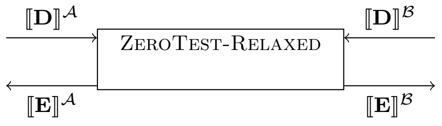
1 : Alice: $\hat{\mathbf{T}} = f_{\mathbf{T}}(\mathbf{T})$ — Bob: $\hat{\mathbf{P}} = f_{\mathbf{P}}(\mathbf{P})$

Bob: $m' = |\{i \mid p_i \neq \star\}|$



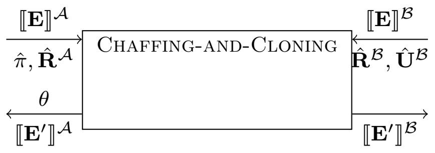
$\hat{\mathbf{T}}$
$\hat{\mathbf{P}}^{rev}$
$\mathbf{C} = \hat{\mathbf{T}} * \hat{\mathbf{P}}^{rev}$
(Using Protocol 4.1)
$[\![\mathbf{C}]\!]^{\mathcal{A}}$
$[\![\mathbf{C}]\!]^{\mathcal{B}}$

2 : Alice: $[\![\mathbf{D}]\!]^{\mathcal{A}} = [\![\mathbf{C}]\!]^{\mathcal{A}}$ — Bob: $[\![\mathbf{D}]\!]^{\mathcal{B}} = [\![\mathbf{C}]\!]^{\mathcal{B}} - \overbrace{[m', m', \cdots, m']}^{\text{length: } N}$



$[\![\mathbf{D}]\!]^{\mathcal{A}}$
$[\![\mathbf{D}]\!]^{\mathcal{B}}$
ZeroTest-Relaxed
$[\![\mathbf{E}]\!]^{\mathcal{A}}$
$[\![\mathbf{E}]\!]^{\mathcal{B}}$

3 :



$[\![\mathbf{E}]\!]^{\mathcal{A}}$
$\hat{\pi}, \hat{\mathbf{R}}^{\mathcal{A}}$
$[\![\mathbf{E}]\!]^{\mathcal{B}}$
$\hat{\mathbf{R}}^{\mathcal{B}}, \hat{\mathbf{U}}^{\mathcal{B}}$
Chaffing-and-Cloning
θ
$[\![\mathbf{E}']\!]^{\mathcal{A}}$
$[\![\mathbf{E}']\!]^{\mathcal{B}}$

.... Phase 2: Reveal the predicate value $(\gamma > 0)$ to Bob ....

4 :

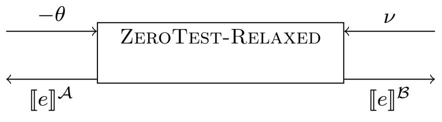
$$\xrightarrow{\quad [\![\mathbf{E}']\!]^{\mathcal{A}} \quad}$$

5 : Bob: $\mathbf{E} = [\![\mathbf{E}']\!]^{\mathcal{A}} + [\![\mathbf{E}']\!]^{\mathcal{B}}$

6 : Bob: $\Gamma' = \{i \mid e'_i = 0\}$

7 : Bob: $\nu = |\Gamma'|$

8 :



9 :

$$\xrightarrow{\quad [\![e]\!]^{\mathcal{A}} \quad}$$

10 : Bob: $e = [\![e]\!]^{\mathcal{A}} + [\![e]\!]^{\mathcal{B}}$

11 : Alice: **return** $\perp$ — Bob: **return** $1 - \mathbf{sgn}(e)$

Protocol 4.10. Secure approximation of $(\perp, (\gamma > 0)) = \mathcal{F}^{D}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ based on $\mathcal{F}_{\text{REQZ}}$

Proof. The correctness of Protocol 4.10 follows from the construction of the augmented vector $\mathbf{E}' = \pi(\mathbf{E}'')$. The count of zeros in $\mathbf{E}''$ (hence, $\mathbf{E}'$) is $\nu = r\gamma + \theta$: the first term, $r\gamma$, is because of the replicas component of $\mathbf{E}''$, i.e., $\overbrace{[\![\mathbf{E}]\!]^{\mathcal{A}} || \cdots || [\![\mathbf{E}]\!]^{\mathcal{A}}}^{r \text{ copies}}$, and the second term, θ, is due to the structure of the vector **Chaff** chosen by Alice that has exactly $\theta \geq 1$ zeros. Clearly, $\nu = \theta$ if and only if $\gamma = 0$. Hence, computing $([\![e]\!]^{\mathcal{A}}, [\![e]\!]^{\mathcal{B}}) = \textsc{ZeroTest-Relaxed}(-\theta, \nu)$ results in $[\![e]\!] = 0$ if and only if $\gamma = 0$, which ensures that $1 - \mathbf{sgn}(e)$ is equal to the predicate value $(\gamma > 0)$.

Security of Phase 1 in Protocol 4.10 follows from the previously discussed security of the protocols 4.1, 4.5 and 4.9. Thus, it suffices to show that in the second phase Bob learns no more than an upper bound on γ; note that he learns ν, which is the number of genuine matches (γ) plus the count of fake matches ($\nu - \gamma = (r-1)\gamma + \theta$). Existence of the fake entries guarantees that Bob does not learn a lower bound on the number of matches. On the other hand, the use of protocol ZeroTest-Relaxed for revealing to Bob that whether or not $\nu = \theta$ ensures that he does not learn anything about the count gap $\nu - \gamma$. □

4.6 Experimental Results

We implemented our SWPM Protocols based on the $\mathcal{F}_{\text{REQZ}}$ functionality, that we call **LiLiP** (**Li**ghtweight, **Li**nearithmic & **P**rivate). We implemented the scheme using Java programming language, 64-bit JRE 8; the experiments were performed using a 2.80 GHz Intel Core i7 CPU with 16GB RAM on macOS. Although our implementation is a proof-of-concept and a more optimized implementation can obtain even better performances, the experimental results show that our protocols perform very well in practice.

4.6.1 Performance: Execution Time & Transferred Data

The tables below show the *execution time* and *volume of transferred data* by LiLiP on various input size combinations. All input strings used in our experiments are over the alphabet of all ASCII characters (including both standard and extended ASCII characters), i.e., size $|\Sigma| = 256$; this choice of alphabet demonstrates practicality of the proposed protocols for larger alphabets (i.e., not only binary, DNA or natural language alphabets). The text string lengths in our experiments ranged from thousands to more than a million characters. Tables 4.1, 4.2 and 4.3 present the cases where the pattern string $\mathbf{P}$ consists of 64, 128 and 512 characters respectively. Furthermore, for the decision version, we used chaffing and cloning parameters $c \simeq n$ and $r = 2$.

As the computations involving the generation and pre-computations on randoms, roots of unity, and alphabet translations do not depend on the input strings, we did not group them with computations that depend on the input strings; the reason is that the former can be done offline, ahead of time, and in fact stored on external storage to be used later by Alice and Bob whenever they need to do pattern matching. Hence, the tables below show performance of computations that involve the input and pattern strings.

Table 4.1.
All times are measured in seconds, and all transferred data volumes are measured in megabytes. This table demonstrates LiLiP's performance for pattern queries of 64 characters, i.e., $|\mathbf{P}| = m = 64$.

$\lvert\mathbf{T}\rvert = n$	2^{10}		2^{14}		2^{16}		2^{20}	
	Time	**Data**	**Time**	**Data**	**Time**	**Data**	**Time**	**Data**
Search-Relaxed	0.12	0.036	0.48	0.61	1.24	2.45	18.54	39.18
Counting-Relaxed	0.15	0.055	0.55	0.91	1.40	3.67	20.34	58.78
Decision-Relaxed	0.18	0.136	0.64	2.43	2.08	9.79	53.37	156.73

Table 4.2.
All times are measured in seconds, and all transferred data volumes are measured in megabytes. This table demonstrates LiLiP's performance for pattern queries of 128 characters, i.e., $|\mathbf{P}| = m = 128$.

$\lvert\mathbf{T}\rvert = n$	2^{10}		2^{14}		2^{16}		2^{20}	
	Time	**Data**	**Time**	**Data**	**Time**	**Data**	**Time**	**Data**
Search-Relaxed	0.15	0.036	0.53	0.61	1.41	2.45	20.16	39.18
Counting-Relaxed	0.18	0.053	0.54	0.92	1.46	3.67	20.99	58.78
Decision-Relaxed	0.19	0.144	0.82	2.44	2.12	9.79	57.09	156.74

Table 4.3.
All times are measured in seconds, and all transferred data volumes are measured in megabytes. This table demonstrates LiLiP's performance for pattern queries of 512 characters, i.e., $|\mathbf{P}| = m = 512$.

$\lvert\mathbf{T}\rvert = n$	2^{10}		2^{14}		2^{16}		2^{20}	
	Time	**Data**	**Time**	**Data**	**Time**	**Data**	**Time**	**Data**
SEARCH-RELAXED	0.23	0.028	1.27	0.60	3.89	2.44	59.32	39.18
COUNTING-RELAXED	0.27	0.038	1.33	0.90	4.02	3.65	60.89	58.76
DECISION-RELAXED	0.46	0.086	1.40	2.38	4.15	9.73	95.95	156.69

A comparison of tables 4.1, 4.2 and 4.3 shows that the volume of transferred data decreases as the pattern size $|\mathbf{P}| = m$ increases. This may seem contradictory at first sight; however, an accurate scrutiny clarifies this phenomenon: Recall that we consider wildcard pattern matching without wraparounds; hence, the larger the pattern length m, the smaller the max index $N = n - m + 1$. Thus, avoiding the wraparound indices $N + 1, N + 2, \cdots, n$ results in smaller data transfer among Alice and Bob.

4.6.2 Comparison with Previous State of the Art

We shall now compare LiLiP with PriSearch [59] because the latter had a better performance among previous SWPM schemes. For instance, PriSearch performs 15× faster than [85] which is itself more efficient compared to other automaton-based approaches; furthermore, PriSearch is 2× faster than the 5PM scheme of [56]; see [59] for more details on these comparisons.

Before we compare LiLiP and PriSearch, we note that the system in [59] (Intel Core i7 CPU @ 3.4 GHz processors, 12GB RAM with 64-bit Ubuntu 14 operating system) is somewhat faster than the setting we used for our experiments. Below, we compare our scheme to the two cases of secure wildcard pattern matching (search version) considered in [59]:

1. $n = 2^{16}$ and $m = 100$ ($m = 128$ in our case): LiLiP performs almost 21.5× faster than PriSearch. LiLiP takes 2.12 seconds while PriSearch's reported execution time is 45.49 seconds. Moreover, the volume of transferred data is 9.78 MB for LiLiP, which is drastically smaller than 4.74 GB for PriSearch.

2. $n = 2^{20}$ and $m = 10$ ($m = 16$ in our case): LiLiP performs almost 1.4× faster than PriSearch. LiLiP takes 51.02 seconds while PriSearch's reported execution time is 69.37 seconds. Moreover, the volume of transferred data is 176.54 MB for LiLiP, which is drastically smaller than 7.21 GB for PriSearch.

Note that our speeds are much better even though we used slower hardware and somewhat larger values for m. It is noticeable that LiLiP's performance advantage increases as m grows larger; this is expected because our scheme has sub-quadratic computation complexity and linear communication complexity, whereas PriSearch's computation and communication complexities are both $\mathcal{O}(nm \log |\Sigma|)$.

4.7 Discussion: SWPM in Stronger Adversarial Models

The most broadly used and studied variant of secure wildcard pattern matching is its *search* version, $(\perp, \Gamma) = \mathcal{F}^{S}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$, in which Bob learns the set Γ that contains all positions in Alice's text $\mathbf{T}$ at which $\mathbf{P}$ occurs. In what follows, we point out that in $\mathcal{F}^{S}_{\text{SWPM}}$ a judicious choice of pattern $\mathbf{P}$ would result in Bob learning an unacceptable amount of information about Alice's private text $\mathbf{T}$. In other words, in the augmented semi-honest and malicious threat models that allow participants to use any input rather than their prescribed input, the output-receiving party (i.e., Bob) can obtain more information about $\mathbf{T}$ than he is supposed to learn. This drawback is due to the functionality definition, and can be exploited regardless of the protocol used for secure computation.

The key idea in such an attack by Bob is that if $\mathbf{P}$ is a single alphabet symbol (i.e., $m = 1$ and $\mathbf{P} \neq$ "$\star$"), then the set Γ contains all positions in $\mathbf{T}$ where that alphabet symbol appears. This, together with the power of the wildcard symbol $\star$ that matches any alphabet symbol, allows the dishonest Bob to (partially) reconstruct Alice's private text $\mathbf{T}$. Example 4.3 below, illustrates how Bob can carry out such an attack.

Example 4.3. Let $\Sigma = \{1, 2\}$, $n = 16$, and $m = 4$, where

$$\mathbf{T} = \text{“1122121121222121”}, \qquad \text{and} \qquad \mathbf{P} = \text{“}122\star\text{”}$$

Moreover, we assume that Alice and Bob have access to a trusted third party who receives the private inputs from both party's and returns Γ to Bob. This shows that the following attack by Bob is independent of the secure protocols used for obtaining Γ. In this example, we compare what Bob learns about Alice's private text $\mathbf{T}$ in two cases:

- **Bob honestly inputs the prescribed pattern $\mathbf{P}$ = “$122\star$”:** The given pattern $\mathbf{P} \in (\Sigma \cup \{\star\})^4$ is a valid input for $\mathcal{F}^{S}_{\text{SWPM}}$, hence the trusted party proceeds and returns $\Gamma = \{i \mid \mathbf{T}_i \overset{\star}{=} \mathbf{P}\} = \{2, 10\}$. From this, Bob learns that $\mathbf{T}$ = “?122?????122????” and that $t_i t_{i+1} t_{i+2} \neq$ “122” for any $i < 14$ such that $i \notin \{2, 10\}$.

- **Bob dishonestly inputs the crafted pattern $\tilde{\mathbf{P}}$ = “$1\star\star\star$”:** The given pattern $\tilde{\mathbf{P}} \in (\Sigma \cup \{\star\})^4$ is a valid input for $\mathcal{F}^{S}_{\text{SWPM}}$, hence the trusted party proceeds and returns $\Gamma' = \{i \mid \mathbf{T}_i \overset{\star}{=} \tilde{\mathbf{P}}\} = \{1, 2, 5, 7, 8, 10\}$. From this, what Bob learns about $\mathbf{T}$ is that $\mathbf{T}$ = “1122121121222???”.

Clearly, Bob learns more about $\mathbf{T}$ in the latter case. Moreover, engaging in another instance of $\mathcal{F}^{S}_{\text{SWPM}}$ in which Bob uses $\tilde{\mathbf{P}}'$ = “$\star\star\star 1$” would enable him to learn $\mathbf{T}$ in its entirety.

Note that in the case of $m = 1$, Alice would have known the risk of exposure and may not have engaged in the computation. However, the fact that $m > 1$ together with the requirement that Alice learns nothing about Bob's private pattern make it impossible for her to determine whether or not Bob's behavior is baleful.

To remedy the weakness described above, we propose a *Filtered* SWPM, denoted by $\mathcal{F}^{\tau}_{\text{SWPM}}$, in which Bob learns only $\tau(\Gamma)$ for an agreed-upon *output-filtering* function τ; Figure 4.7 illustrates this functionality. It is evident that $\mathcal{F}^{\tau}_{\text{SWPM}}$ is the most general instantiation of SWPM in the 2PC setting; all previously discussed versions of SWPM can be obtained by a proper choice of the filter function $\tau(\cdot)$ as discussed below:

- Search version: $\mathcal{F}^{S}_{\text{SWPM}}$ is equivalent to the filtered SWPM with $\tau(\cdot)$ being the “identity” function, i.e., $\tau_{\text{id}}(\Gamma) = \Gamma$.

- Counting version: $\mathcal{F}^{C}_{\text{SWPM}}$ is equivalent to the filtered SWPM with $\tau(\cdot)$ being the "cardinality" function, i.e., $\tau_{\text{card}}(\Gamma) = |\Gamma|$.
- Decision version: $\mathcal{F}^{D}_{\text{SWPM}}$ is equivalent to the filtered SWPM with $\tau(\cdot)$ being the "is-not-empty" function, i.e., $\tau_{\text{ine}}(\Gamma) = (\Gamma \neq \emptyset)$.

Our solutions in Section 4.5 directly reflect the above instances of the filtered SWPM functionality. In the remainder of this section, we give two more instances of filtered SWPM in which Bob learns only one member of Γ (only if the set is non-empty).

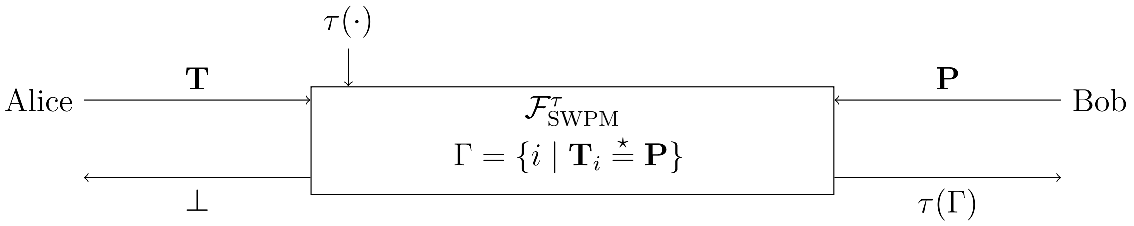
4.7.1 Bob Learns Only the Leftmost Match

Filtering function $\tau_{\min}(\Gamma)$ allows Bob to learn only the smallest i such that $\mathbf{T}_i \overset{\star}{=} \mathbf{P}$; i.e.,

$$\tau_{\min}(\Gamma) = \begin{cases} \min(\Gamma) & \text{if } \Gamma \neq \emptyset \\ 0 & \text{if } \Gamma = \emptyset \end{cases} \tag{4.21}$$

Protocol 4.11, Filtered-Swpm-LeftMost, securely realizes $\mathcal{F}^{\tau_{\min}}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$. In a nutshell, Alice and Bob first obtain the indicator vector $[\![\mathbf{E}]\!]$ such that $e_i = 1$ if and only if $\mathbf{T}_i \overset{\star}{=} \mathbf{P}$ (as discussed in Section 4.5.2). Then, they overwrite all entries in it, except one, with 0. For that, Alice and Bob first compute (non-interactively) an intermediary vector $[\![\bar{\mathbf{E}}]\!]$ such that $[\![\bar{e}_i]\!] = \sum_{j=1}^{i} j \cdot [\![e_j]\!]$ for all $1 \leq i \leq N$; then, they obliviously compute a final matching vector $[\![\mathbf{L}]\!]$ as in Equation 4.22 below:

$$[\![l_i]\!] = \begin{cases} [\![\bar{e}_i]\!] & \text{if } i = 1 \\ [\![(\bar{e}_{i-1} == 0) \cdot \bar{e}_i]\!] & \text{if } 2 \leq i \leq N \end{cases} \tag{4.22}$$



$\tau(\cdot)$
Alice
$\mathbf{T}$
$\mathbf{P}$
Bob
$\mathcal{F}^{\tau}_{\text{SWPM}}$
$\Gamma = \{i \mid \mathbf{T}_i \overset{\star}{=} \mathbf{P}\}$
$\perp$
$\tau(\Gamma)$

Figure 4.7. $(\perp, \tau(\Gamma)) = \mathcal{F}^{\tau}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$

By construction, $[\![\mathbf{L}]\!]$ has at most one non-zero entry that is equal to the leftmost non-zero entry in $[\![\bar{\mathbf{E}}]\!]$, which is itself the smallest integer in Γ. Hence, Alice and Bob can simply compute (non-interactively)

$$[\![\tau_{\text{min}}(\Gamma)]\!] = \sum_{k=1}^{N} [\![l_k]\!]$$

Then, Alice sends her share $[\![\tau_{\text{min}}(\Gamma)]\!]^{\mathcal{A}}$ to Bob so he can reconstruct the desired output $\tau_{\text{min}}(\Gamma)$.

4.7.2 Bob Learns One Match at Random

Filtering function $\tau_{\text{rand1}}(\Gamma)$ allows Bob to learn exactly one index $i \in \Gamma$ (if any) chosen uniformly at random; i.e.,

$$\tau_{\text{rand1}}(\Gamma) = \begin{cases} i \overset{\$}{\leftarrow} \Gamma & \text{if } \Gamma \neq \emptyset \\ 0 & \text{if } \Gamma = \emptyset \end{cases} \tag{4.23}$$

Filtered-Swpm-LeftMost (Protocol 4.11) can easily be modified to obtain a secure realization of $(\perp, \tau_{\text{rand1}}(\Gamma)) = \mathcal{F}_{\text{SWPM}}^{\tau_{\text{rand1}}}(\mathbf{T}, \mathbf{P})$. The key idea is to find the leftmost non-zero entry in $[\![\mathbf{E}']\!] = [\![\pi(\mathbf{E})]\!]$ (rather than in $[\![\mathbf{E}]\!]$), where π is a random permutation that is unknown to Bob. Hence, the required modifications to Filtered-Swpm-LeftMost are:

1. After computing the indicator vector $[\![\mathbf{E}]\!]$ (right before Phase 2), Alice and Bob use the protocol Single-Blind-Perm and Observation 4.1 to obtain $[\![\mathbf{E}']\!] = [\![\pi(\mathbf{E})]\!]$ as described above.

2. In Phase 2, Alice and Bob find the leftmost none-zero entry in $[\![\mathbf{E}']\!]$ (rather than $[\![\mathbf{E}]\!]$). For that it suffices to only modify Step 3, by replacing $[\![e_j]\!]^{\mathcal{A}}$ and $[\![e_j]\!]^{\mathcal{B}}$ with $[\![e'_j]\!]^{\mathcal{A}}$ and (respectively) $[\![e'_j]\!]^{\mathcal{B}}$.

Filtered-Swpm-LeftMost

Alice — On input $\mathbf{T}$

Bob — On input $\mathbf{P}$

.... Phase 1: Compute the indicator vector $[\![\mathbf{E}]\!]$ ....

1 : Alice: $\hat{\mathbf{T}} = f_{\mathbf{T}}(\mathbf{T})$ — Bob: $\hat{\mathbf{P}} = f_{\mathbf{P}}(\mathbf{P})$, $m' = |\{i \mid p_i \neq \star\}|$

$$\begin{tikzcd} {} \arrow[r, shift left, "\hat{\mathbf{T}}"] & \boxed{\begin{array}{c}\mathbf{C} = \hat{\mathbf{T}} * \hat{\mathbf{P}}^{rev} \\ \text{(Using Protocol 4.1)}\end{array}} \arrow[l, shift left, "{[\![\mathbf{C}]\!]^{\mathcal{A}}}"] \arrow[r, shift right, "{[\![\mathbf{C}]\!]^{\mathcal{B}}}"'] & {} \arrow[l, shift right, "\hat{\mathbf{P}}^{rev}"'] \end{tikzcd}$$

2 : Alice: $[\![\mathbf{D}]\!]^{\mathcal{A}} = [\![\mathbf{C}]\!]^{\mathcal{A}}$ — Bob: $[\![\mathbf{D}]\!]^{\mathcal{B}} = [\![\mathbf{C}]\!]^{\mathcal{B}} - \overbrace{[m', m', \cdots, m']}^{\text{length: } N}$

$$\begin{tikzcd} {} \arrow[r, shift left, "{[\![\mathbf{D}]\!]^{\mathcal{A}}}"] & \boxed{\begin{array}{c}\mathcal{F}_{\text{EQZ}} \\ e_i = (d_i == 0)\end{array}} \arrow[l, shift left, "{[\![\mathbf{E}]\!]^{\mathcal{A}}}"] \arrow[r, shift right, "{[\![\mathbf{E}]\!]^{\mathcal{B}}}"'] & {} \arrow[l, shift right, "{[\![\mathbf{D}]\!]^{\mathcal{B}}}"'] \end{tikzcd}$$

.... Phase 2: Reveal $t = \tau_{\min}(\Gamma)$ to Bob ....

3 : Alice: Compute $[\![\bar{\mathbf{E}}]\!]^{\mathcal{A}}$ s.t. for $1 \leq j \leq n$ $[\![\bar{e}_j]\!]^{\mathcal{A}} = j \cdot [\![e_j]\!]^{\mathcal{A}}$ — Bob: Compute $[\![\bar{\mathbf{E}}]\!]^{\mathcal{B}}$ s.t. for $1 \leq j \leq n$ $[\![\bar{e}_j]\!]^{\mathcal{B}} = j \cdot [\![e_j]\!]^{\mathcal{B}}$

4 : Alice: Compute $[\![\hat{\mathbf{E}}]\!]^{\mathcal{A}}$ s.t. for $1 \leq i \leq n$ $[\![\hat{e}_i]\!]^{\mathcal{A}} = \sum_{k=1}^{i} [\![\bar{e}_k]\!]^{\mathcal{A}}$ — Bob: Compute $[\![\hat{\mathbf{E}}]\!]^{\mathcal{B}}$ s.t. for $1 \leq i \leq n$ $[\![\hat{e}_i]\!]^{\mathcal{B}} = \sum_{k=1}^{i} [\![\bar{e}_k]\!]^{\mathcal{B}}$

$$\begin{tikzcd} {} \arrow[r, shift left, "{[\![\hat{\mathbf{E}}]\!]^{\mathcal{A}}}"] & \boxed{\begin{array}{c}\text{Compute } \mathbf{L} \\ \text{(Using Equation 4.22)}\end{array}} \arrow[l, shift left, "{[\![\mathbf{L}]\!]^{\mathcal{A}}}"] \arrow[r, shift right, "{[\![\mathbf{L}]\!]^{\mathcal{B}}}"'] & {} \arrow[l, shift right, "{[\![\hat{\mathbf{E}}]\!]^{\mathcal{B}}}"'] \end{tikzcd}$$

5 : Alice: $[\![t]\!]^{\mathcal{A}} = \sum_{k=1}^{N} [\![l_k]\!]^{\mathcal{A}}$ — Bob: $[\![t]\!]^{\mathcal{B}} = \sum_{k=1}^{N} [\![l_k]\!]^{\mathcal{B}}$

6 :

$$\begin{tikzcd} \text{Alice} \arrow[r, "{[\![t]\!]^{\mathcal{A}}}"] & \text{Bob} \end{tikzcd}$$

7 : Bob: $t = [\![t]\!]^{\mathcal{A}} + [\![t]\!]^{\mathcal{B}}$

8 : Alice: **return** $\perp$ — Bob: **return** t

Protocol 4.11. Secure realization of $(\perp, \tau_{\min}(\Gamma)) = \mathcal{F}^{\tau_{\min}}_{\text{SWPM}}(\mathbf{T}, \mathbf{P})$ based on $\mathcal{F}_{\text{EQZ}}$

5. SECURE TWO-PARTY INPUT-SIZE REDUCTION

Secure two-party computation has recently become a more practically applicable cryptographic technology, and many researchers have been pursuing efficient solutions to a wide variety of 2PC problems (e.g., [27], [29]–[37]). Some protocols use preprocessing of inputs to improve the overall performance, mainly using techniques that are either problem-specific or approach-specific [31], [60]. In this work, we develop a generic preprocessing mechanism, applicable for *all* 2PC problems that rely on equality tests (e.g., pattern matching). Our mechanism reduces the bit-length of inputs such that using the size-reduced inputs to solve the problem of interest results in the same output as if the original inputs had been used. Such input-size reduction is especially advantageous when its cost can be amortized over multiple subsequent computations that all benefit from the already-done size-reduction. This is true for most 2PC approaches, e.g., in garbled circuits, the circuit size depends on the inputs' bit-length, and in many number-theoretic approaches, the arithmetic cost depends on the size of input integers.

Aiming for faster information retrieval and saving memory space in a 2PC setting, Goldreich et al. [60] considered the problem of mapping long names into smaller abbreviations. The solution in [60] (i) requires a trusted party who knows all input names, and (ii) allows a small probability of collision for each pair of names, which is when two distinct long names are mapped into the same abbreviation. In this chapter, we consider the above size-reduction problem when there is no such trusted party. Moreover, we impose two further desiderata: We require (i) a very small abbreviation space (*codomain*), where small means equal or close to the number of input items (we later make this notion more precise); and, (ii) a guarantee of *no collisions at all*, which is particularly challenging because of the above requirement of a very small codomain. Although these requirements make the problem considerably more difficult, they are necessitated by our results' specific applications.

5.1 Motivation and Overview

Consider a 2PC functionality $\mathcal{F}$ whose inputs are over a large domain $\Sigma \subset \mathbb{F}_q$, but only a small subset of Σ appears in the inputs (see Section 5.9 for examples). In such situations, it is desirable to avoid the communication and computation costs corresponding to symbols of Σ that do not occur in the inputs. If security were not a concern, it would be easy to replace the large inputs with shorter ones whose bit-length is the logarithm of the count of occurring symbols (rather than $\log_2 |\Sigma|$). However, in the 2PC setting where each input is private to a party, it would be inappropriate to reveal the set of occurring symbols because doing so would leak information about the private inputs. Hence, a secure preprocessing mechanism is needed to obtain the size-reduced symbols. In other words, symbols that appear in the inputs must be securely mapped to smaller values in such a way that

- the mapping is consistent among all parties;
- the mapping is collision-free; and,
- the bit-length of size-reduced symbols is as small as possible.

Section 5.9 gives examples of 2PC problems that benefit from such a secure input-size reduction.

The secure input-size reduction problem discussed in this chapter is reminiscent of (but different in fundamental ways from) the classic perfect hashing problem: Let S be a set containing large integers (i.e., $S \subset \Sigma = \{0, \ldots, 2^\sigma - 1\} = \{0,1\}^\sigma$). A perfect hash function (PHF) for S is a function $\rho : \Sigma \to \{0, 1, \ldots, N-1\}$ such that N is a small integer (close to $|S|$) and ρ is injective on S [92], [93]. In this chapter, we propose a *secure* perfect hashing approach to the above-mentioned secure input-size reduction problem. Although perfect hashing is a well-studied and broadly used topic when security and privacy are of no concern [92]–[100], the patent document by Nawaz *et al.* [101] seems to be the only existing work that considers perfect hashing in a secure setting; however, the results of [101] do not solve the secure input-size reduction problem (see Section 5.2 for details). We first formally introduce Secure Perfect Hash Functions (SPHFs) for the union of two private sets $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$. Then, we propose provably secure protocols to construct such a hash function and argue that SPHFs are sufficient for secure input-size reduction.

5.2 Related Work

This section reviews the existing work in the literature related to perfect hashing and secure input-size reduction. Construction and analysis of perfect hash functions are well-studied in the area of algorithms and data structures when security and privacy are not of concern [92]–[100]. Some constructions [94]–[96] rely on the theoretical properties of sets of integers. Though these methods are simple and theoretically interesting, they tend to be slow either in the construction step or the evaluation step (or both). Fredman *et al.* [97] proposed an efficient two-level PHF construction that obtains asymptotically minimal codomain size $\mathcal{O}(|S|)$. The key idea in [97] is to partition the items into m subsets using a universal family of hash functions [102], followed by another layer of universal hashing to obtain injective behavior on each subset. More recent works such as [98]–[100] focus on efficient solutions to (nearly) minimal perfect hashing.

Nawaz *et al.* [101] seems to be the only existing work that addresses the need of a secure perfect hash function for data matching applications between private databases. They propose a solution based on cryptographic hash functions and the simple and space-efficient minimal perfect hashing "*hash, displace, and compress*" technique [98], [103]. In [101], authors argue that their scheme is secure with respect to the private sets due to the preimage-resistant properties of cryptographic hash functions like SHA2. However, this is not necessarily true since their design uses only a few bytes of the cryptographic hash output to construct a perfect hash function. Thus, the resulting SPHF is prone to a membership-testing attack that will be described in Section 5.4. Also, [101] lacks formal security, correctness, and performance analyses and is vague as to how parties interact.

Following the GGM construction for Pseudorandom Functions (PRFs) [104], Goldreich *et al.* [60] considered the problem of mapping long names into smaller abbreviations with a small probability of collision for each pair of items. This achieves faster information retrieval and saves memory space [60]. As an application, Goldreich *et al.* [60] designed a Friend or Foe Identification (FFI) mechanism, which enables members of a secret club to identify each other. The PRF-based solution to FFI [60] is suitable for cryptographic purposes as it is more robust than schemes based on classical primitives such as universal hashing [102].

In FFI, only "club members" can use the system, and their designated leader knows all members' names. By contrast, we consider the above name-reduction problem when there is no such leader. As stated earlier, we also impose the requirement of no collisions at all, while achieving a very small codomain.

In 2015, Pinkas *et al.* [31] proposed a bit-length reduction scheme specifically geared towards faster Private Set Intersection (PSI) in the MPC framework. The proposed method in [31] elegantly applies simple hashing for partitioning the input sets into a number of bins such that the reduced bit-representations for two items *in the same bin* are equal if and only if the two items are equal. The scheme in [31] results in no intra-bin collisions, but does not guarantee the absence of inter-bin collisions; i.e., the size-reduction technique of [31] allows two items to have the same size-reduced representations as long as they are not in the same bin. Although the above works perfectly for solving PSI, the fact that it allows inter-bin collisions prevents its use for many other problems. Our solutions do not allow collisions of any type.

Some articles use terminologies reminiscent of ours but are different in nature. One is Oblivious Hashing [105], which is a software integrity verification scheme. Another is PerfectDedup [106], which is a mechanism for data deduplication in cloud storage management that uses a PHF on encrypted data to securely identify the popular data segments for deduplication.

5.3 Summary of Contributions

We present the first formal attempt to define and solve the secure perfect hashing problem and use it for secure input-size reduction in the 2PC framework. Although our definitions and solutions have natural extensions for any number of parties, we focus on the two-party case. Below, we give an overview of our main contributions.

5.3.1 Problem Formulation

In Section 5.4, we formulate secure two-party input-size reduction by formalizing the notion of a Secure Perfect Hash Function (SPHF) for the union of private sets $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$

as a PHF for $S = S^{\mathcal{A}} \cup S^{\mathcal{B}}$ such that the knowledge of its mapping on $S^{\mathcal{A}}$ (resp. $S^{\mathcal{B}}$) does not reveal anything about $S^{\mathcal{B}}$ (resp. $S^{\mathcal{A}}$). Definition 5.1 presents the formal definition of a SPHF.

5.3.2 Proposed Constructions

We first propose an approach, FINDSPHF, to construct a minimal perfect SPHF for S; we give two embodiments for it that trade off round complexity for computation complexity. We further improve both round and computation complexities by introducing a *Distribution* level that divides S into m disjoint subproblems; then, each subproblem is solved independently using FINDSPHF. Table 5.1 compares the performance and resulting codomain size of our SPHF constructions.

- **Perfect SPHF Construction (Section 5.6):** We show that if a hash function $\rho : \Sigma \to \{0, 1, \dots, N-1\}$ is chosen uniformly at random among all functions that are injective on $S = S^{\mathcal{A}} \cup S^{\mathcal{B}}$, then ρ is a minimal perfect SPHF for S. Afterward, we propose a Las Vegas approach, FINDSPHF, that securely and efficiently obtains such a function ρ. The key idea in FINDSPHF is assigning unique random labels to items in $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$, followed by a judicious label unification step for duplicates $x \in S^{\mathcal{A}} \cap S^{\mathcal{B}}$. We give two embodiments for FINDSPHF, namely, LABEL-THEN-UNIFY and MERGE-THEN-UNIFY.

- **Distribution-Resolution Construction (Sections 5.7 and 5.8):** For more efficient SPHF construction, Section 5.7 introduces a Distribution level that uses a standard balls and bins analysis [107] to find a *distributor* function Ψ that securely partitions S into $m = |S^{\mathcal{A}}| + |S^{\mathcal{B}}|$ subproblems of size $\mathcal{O}(\log m)$ each. Then, FINDSPHF solves these subproblems independently (but in parallel) giving SPHFs ρ_i that we call *resolvers* (because they resolve intra-bin collisions). Section 5.8 describes how to combine Ψ and the ρ_is to obtain the desired SPHF ρ for S. One of the security measures in Distribution level is formulating the notion of two-party *Oblivious Distributed PRF* (ODPRF), whose DPRF part is in the spirit of Naor *et al.* [72]. The use of an ODPRF as a distributor function forces a party (querier) to seek assistance of the other party

Table 5.1.
Performance of proposed SPHF constructions: Let $m_{\mathcal{A}} = |S^{\mathcal{A}}|, m_{\mathcal{B}} = |S^{\mathcal{B}}|$, $m = m_{\mathcal{A}} + m_{\mathcal{B}}$ and the security-performance parameter κ be a small integer; as m grows larger, a smaller κ suffices for the same level of security (see Section 5.8 for choice of κ).

Scheme	Computation & Communication	Rounds	Codomain Cardinality
Label-then-Unify (§5.6.1)	$\mathcal{O}(m_{\mathcal{A}} \cdot m_{\mathcal{B}})$	$\mathcal{O}(1)$	m
Merge-then-Unify (§5.6.2)	$\mathcal{O}(m \log m)$	$\mathcal{O}(\log m)$	m
Distribute-Label (§5.7 and §5.6.1)	$\mathcal{O}(m(\kappa + (\frac{\log m}{\log\log m})^2))$	$\mathcal{O}(1)$	$\mathcal{O}(\frac{m \log m}{\log\log m})$
Distribute-Merge (§5.7 and §5.6.2)	$\mathcal{O}(m(\kappa + \log m))$	$\mathcal{O}(\log\log m)$	$\mathcal{O}(\frac{m \log m}{\log\log m})$

for computing $\Psi(x)$, without compromising either x or $\Psi(x)$. We propose a two-party ODPRF based on a combination of Micali and Sidney's xor-PRFs [108] and any standard Oblivious PRF (e.g., [28], [70], [71], [109], [110]).

5.3.3 Use Cases and Implications

In Section 5.9, we describe two applications for which input-size reduction through SPHFs results in significant performance improvements. We also discuss the implicit connection of secure perfect hashing with the important and well-studied problem of Private Set Intersection (PSI). In particular, we show that given an SPHF for $S = S^{\mathcal{A}} \cup S^{\mathcal{B}}$, PSI on inputs $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$ can be solved through exactly $|S^{\mathcal{A}}| + |S^{\mathcal{B}}|$ invocations of secure multiplication in one round of communication.

5.4 Problem Definition

The first step in designing Secure Perfect Hash Functions (SPHFs) is to precisely formulate the security requirements with respect to the private input sets $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$ (recall that $S = S^{\mathcal{A}} \cup S^{\mathcal{B}} \subset \Sigma$). One requirement is that neither Alice nor Bob should be able to *individually* compute $\rho(x)$ for $x \in \Sigma$, where ρ is the desired SPHF. Otherwise, the inherently small codomain of ρ would enable a party to obtain information about the other party's private set via a *membership-testing attack* – because of ρ's small codomain, it would be trivial to find integers that collide under ρ, resulting in information leakage about the private sets. For example, if $y \in S^{\mathcal{A}}$, Alice learns that any $x \neq y$ with $\rho(x) = \rho(y)$ cannot be in $S^{\mathcal{B}}$, thereby leaking information about $S^{\mathcal{B}}$ to Alice. The above vulnerability cannot be fixed by enlarging the codomain as doing so would defeat the purpose of input-size reduction and perfect hashing. To overcome this vulnerability, our definition will require that Alice and Bob learn only the image of their own respective private set under ρ as illustrated in Figure 5.1.

Definition 5.1 (Secure Perfect Hash Function)**.** *A hash function $\rho : \Sigma \to \{0, 1, \ldots, N-1\}$ is an SPHF for $S = S^{\mathcal{A}} \cup S^{\mathcal{B}}$ if and only if*

1. *Correctness: ρ is a PHF for S (ρ is injective on S).*

2. *Security: Given Alice's private output $\varrho^{\mathcal{A}} = \{(x, \rho(x))\}_{x \in S^{\mathcal{A}}}$, but not ρ itself, Alice must not be able to learn anything about $S^{\mathcal{B}}$. Formally, for any $y \in \Sigma$*

$$|\Pr[y \in S^{\mathcal{B}} \mid \varrho^{\mathcal{A}}] - \Pr[y \in S^{\mathcal{B}}]| \leq \mathsf{negl}, \tag{5.1}$$

Similarly, for Bob's private output $\varrho^{\mathcal{B}} = \{(x, \rho(x))\}_{x \in S^{\mathcal{B}}}$ and any $y \in \Sigma$

$$|\Pr[y \in S^{\mathcal{A}} \mid \varrho^{\mathcal{B}}] - \Pr[y \in S^{\mathcal{A}}]| \leq \mathsf{negl}, \tag{5.2}$$

*where negl is a negligible function in the implicit security parameter. In the special case of $\mathsf{negl} = 0$, ρ is called a **perfect SPHF**.*

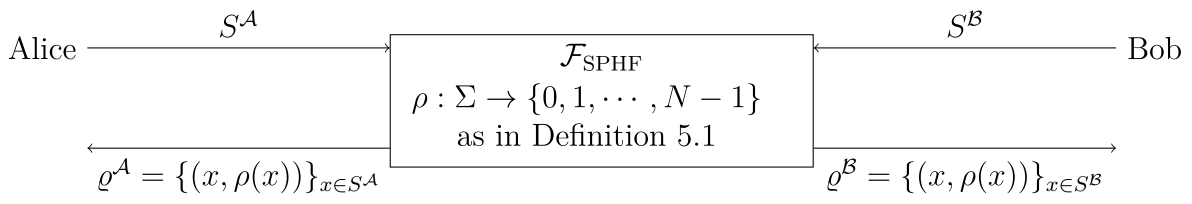

Alice
$S^{\mathcal{A}}$
$S^{\mathcal{B}}$
Bob
$\mathcal{F}_{\text{SPHF}}$
$\rho : \Sigma \to \{0, 1, \cdots, N-1\}$
as in Definition 5.1
$\varrho^{\mathcal{A}} = \{(x, \rho(x))\}_{x \in S^{\mathcal{A}}}$
$\varrho^{\mathcal{B}} = \{(x, \rho(x))\}_{x \in S^{\mathcal{B}}}$

Figure 5.1. $(\varrho^{\mathcal{A}}, \varrho^{\mathcal{B}}) = \mathcal{F}_{\text{SPHF}}(S^{\mathcal{A}}, S^{\mathcal{B}})$

The security requirement in Definition 5.1 implies that a *minimal* SPHF has a codomain of $N = |S^{\mathcal{A}}| + |S^{\mathcal{B}}|$ possible hash values. Note that defining SPHF minimality as $N = |S^{\mathcal{A}} \cup S^{\mathcal{B}}|$ would reveal information about $S^{\mathcal{A}} \cap S^{\mathcal{B}}$.

5.5 Major Challenges

There are two major obstacles towards obtaining an SPHF for $S = S^{\mathcal{A}} \cup S^{\mathcal{B}}$:

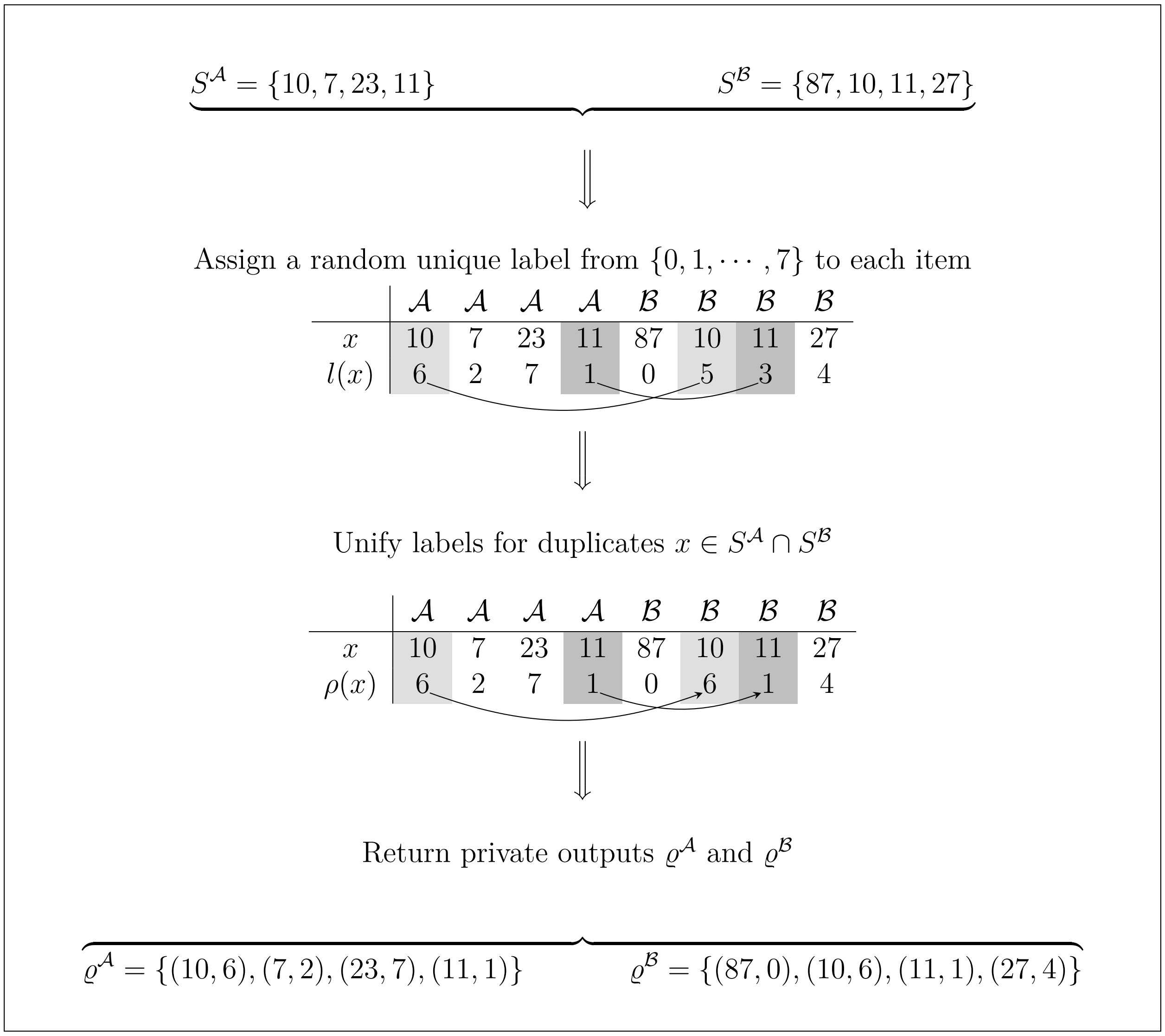
1. *Privacy and Security:* The classical perfect hashing algorithms (e.g., [97], [98]) require centralized knowledge of S to obtain the injective behavior. Even the secure size-reduction scheme in [60] depends on a trusted party who knows all the input items (the scheme also allows collisions with a small probability). Our solutions reveal only a value m that is equal to $|S^{\mathcal{A}}| + |S^{\mathcal{B}}|$. If it is desired to hide $|S^{\mathcal{A}}|$ and $|S^{\mathcal{B}}|$ [111], parties may add a random number of dummy items to their respective private sets before engaging in our SPHF construction protocols, thereby revealing only a loose upper bound on the sizes of their respective sets.

2. *Efficiency:* It is not clear how to use the general 2PC techniques to practically and securely implement the classical perfect hashing algorithms. For example, currently there is no known practical circuit for implementing such schemes through garbled circuits. Moreover, such a (hypothetical) circuit would have a large number of gates, because of the large input items in $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$. Our solutions mitigate this "curse of large inputs" through a judicious combination of local computations and fast cryptographic primitives such as lightweight computations in additive split form.

5.6 Constructing a Minimal Perfect SPHF for $S = S^{\mathcal{A}} \cup S^{\mathcal{B}}$

W.l.o.g., we assume $|S^{\mathcal{A}}| = |S^{\mathcal{B}}| = \gamma$ resulting in $m = 2\gamma$. Let $\mathsf{Inj}_{\Sigma,2\gamma}(S)$ denote the set of all functions $f : \Sigma \xrightarrow{\text{Injective on } S} \{0, 1, \ldots, 2\gamma - 1\}$. Lemma 5.1 formally argues that any function chosen uniformly at random from $\mathsf{Inj}_{\Sigma,2\gamma}(S)$ is a minimal perfect SPHF for $S = S^{\mathcal{A}} \cup S^{\mathcal{B}}$ (recall that a minimal SPHF has a codomain of size $N = m$). Based on this observation, we propose the FINDSPHF approach that gives a minimal perfect SPHF for S. The key idea in FINDSPHF consists of two steps:

1. Obliviously assigning distinct random hash values in the range $\{0, 1, \ldots, 2\gamma - 1\}$ to items in $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$; then,
2. obliviously unifying the labels for duplicate items $x \in S^{\mathcal{A}} \cap S^{\mathcal{B}}$.



$\underbrace{S^{\mathcal{A}} = \{10, 7, 23, 11\} \qquad S^{\mathcal{B}} = \{87, 10, 11, 27\}}$

$\Downarrow$

Assign a random unique label from $\{0, 1, \cdots, 7\}$ to each item

	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{B}$	$\mathcal{B}$	$\mathcal{B}$
x	10	7	23	11	87	10	11	27
$l(x)$	6	2	7	1	0	5	3	4

$\Downarrow$

Unify labels for duplicates $x \in S^{\mathcal{A}} \cap S^{\mathcal{B}}$

	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{B}$	$\mathcal{B}$	$\mathcal{B}$
x	10	7	23	11	87	10	11	27
$\rho(x)$	6	2	7	1	0	6	1	4

$\Downarrow$

Return private outputs $\varrho^{\mathcal{A}}$ and $\varrho^{\mathcal{B}}$

$\overbrace{\varrho^{\mathcal{A}} = \{(10, 6), (7, 2), (23, 7), (11, 1)\} \qquad \varrho^{\mathcal{B}} = \{(87, 0), (10, 6), (11, 1), (27, 4)\}}$

Figure 5.2. High-level illustration of FINDSPHF through an example ($\gamma = 4$). Sections 5.6.1 and 5.6.2 propose protocols that securely and efficiently apply this approach.

Figure 5.2 illustrates the FINDSPHF approach for a small example ($\gamma = 4$). We propose two embodiments of the above idea: The first embodiment, LABEL-THEN-UNIFY (Section 5.6.1), requires $\mathcal{O}(\gamma^2)$ computation in $\mathcal{O}(1)$ rounds; the second embodiment, MERGE-THEN-UNIFY (Section 5.6.2), needs $\mathcal{O}(\gamma \log \gamma)$ computation in $\mathcal{O}(\log \gamma)$ rounds.

Lemma 5.1. *Let $S^{\mathcal{A}}, S^{\mathcal{B}} \subseteq \Sigma$ such that $|S^{\mathcal{A}}| = |S^{\mathcal{B}}| = \gamma$; any function ρ chosen uniformly at random from $\mathsf{Inj}_{\Sigma,2\gamma}(S)$ is a minimal perfect SPHF for $S = S^{\mathcal{A}} \cup S^{\mathcal{B}}$.*

Proof. We show that ρ satisfies the requirements of Definition 5.1. Injectivity and minimality of ρ follow from the choice $\rho \in \mathsf{Inj}_{\Sigma,2\gamma}(S)$. For the security requirement, we prove security against Alice by showing that for $\varrho^{\mathcal{A}} = \{(x, \rho(x))\}_{x \in S^{\mathcal{A}}}$ and any $y \in \Sigma$

$$\Pr[y \in S^{\mathcal{B}} \mid \varrho^{\mathcal{A}}] = \Pr[y \in S^{\mathcal{B}}] \tag{5.3}$$

that is equivalent to Equation 5.1 with $\mathsf{negl} = 0$; security against Bob as in Equation 5.2 follows a symmetric argument.

Let $\phi : \Sigma \to \{0, 1 \ldots, 2\gamma - 1\}$ be constructed as in Figure 5.3. In particular, the construction samples hash values for all items in a certain order: First, it samples (*without*

```
$\phi \xleftarrow{\$} \mathsf{Inj}_{\Sigma,2\gamma}(S)$
1 :  $\Gamma = \{0, 1, \cdots, 2\gamma - 1\}$
2 :  for all $x \in S^{\mathcal{A}}$ do
3 :      $\phi(x) \xleftarrow{\$} \Gamma$
4 :      $\Gamma = \Gamma \setminus \{\phi(x)\}$
5 :  endfor
6 :  for all $x \in S^{\mathcal{B}} \setminus S^{\mathcal{A}}$
7 :      $\phi(x) \xleftarrow{\$} \Gamma$
8 :      $\Gamma = \Gamma \setminus \{\phi(x)\}$
9 :  endfor
10 : for all $x \notin S$ do
11 :     $\phi(x) \xleftarrow{\$} \{0, 1, \cdots, 2\gamma - 1\}$
12 : endfor
```

Figure 5.3. The hypothetical function ϕ used in the proof of Lemma 5.1

replacement) hashes $\phi(x)$ for $x \in S^{\mathcal{A}}$, and then continues for $x \in S^{\mathcal{B}} \setminus S^{\mathcal{A}}$; finally, it samples (*with* replacement) hashes $\phi(x)$ for $x \in \Sigma \setminus S$. By construction, ϕ is chosen uniformly at random from $\mathsf{Inj}_{\Sigma,2\gamma}(S)$. Moreover, the mapping of $S^{\mathcal{A}}$ under ϕ is independent of the set $S^{\mathcal{B}}$; thus, by definition

$$\Pr[y \in S^{\mathcal{B}} \mid \{(x, \phi(x))\}_{x \in S^{\mathcal{A}}}] = \Pr[y \in S^{\mathcal{B}}]$$

Furthermore, any function $\rho \xleftarrow{\$} \mathsf{Inj}_{\Sigma,2\gamma}(S)$ is statistically identical to ϕ, hence,

$$\begin{aligned} \Pr[y \in S^{\mathcal{B}} \mid \varrho^{\mathcal{A}}] &= \Pr[y \in S^{\mathcal{B}} \mid \{(x, \rho(x))\}_{x \in S^{\mathcal{A}}}] \\ &= \Pr[y \in S^{\mathcal{B}} \mid \{(x, \phi(x))\}_{x \in S^{\mathcal{A}}}] \\ &= \Pr[y \in S^{\mathcal{B}}] \end{aligned}$$

This completes the proof that Equation 5.3 holds. □

Our secure constructions for the FindSPHF approach use the *Double-blind Permutation* functionality as illustrated in Figure 5.4: Given a length-λ input vector $[\![\mathbf{X}]\!]$, the functionality $([\![\mathbf{Y}]\!]^{\mathcal{A}}, [\![\mathbf{Y}]\!]^{\mathcal{B}}) = \mathcal{F}_{\text{DBP}}([\![\mathbf{X}]\!]^{\mathcal{A}}, [\![\mathbf{X}]\!]^{\mathcal{B}})$ is such that $[\![\mathbf{Y}]\!] = [\![\pi(\mathbf{X})]\!]$ for a random permutation $\pi : \{1, 2, \cdots, \lambda\} \to \{1, 2, \cdots, \lambda\}$ that is unknown to both parties. There are protocols in the literature that securely realize this functionality [112], [113]. Preferable is a symmetric double-use (with two independent random permutations) of protocol Single-Blind-Perm that was proposed in Chapter 4, which is particularly suitable for use in the secret splitting framework. Such a double use of Protocol 4.7 requires $\mathcal{O}(\lambda)$ computation and communication in a total of 2 rounds. Henceforth, we use the Double-blind Permutation functionality in a black-box manner.

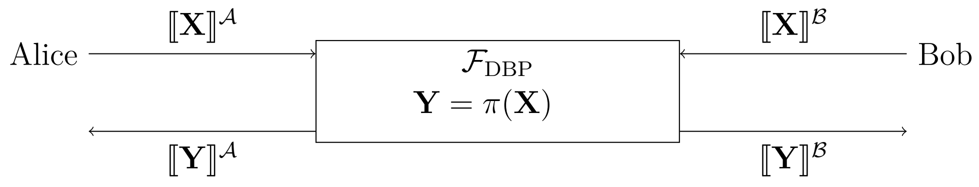

Alice
$[\![\mathbf{X}]\!]^{\mathcal{A}}$
$[\![\mathbf{X}]\!]^{\mathcal{B}}$
Bob
$\mathcal{F}_{\text{DBP}}$
$\mathbf{Y} = \pi(\mathbf{X})$
$[\![\mathbf{Y}]\!]^{\mathcal{A}}$
$[\![\mathbf{Y}]\!]^{\mathcal{B}}$

Figure 5.4. $([\![\mathbf{Y}]\!]^{\mathcal{A}}, [\![\mathbf{Y}^{\mathcal{B}}]\!]) = \mathcal{F}_{\text{DBP}}([\![\mathbf{X}]\!]^{\mathcal{A}}, [\![\mathbf{X}]\!]^{\mathcal{B}})$ such that $[\![\mathbf{Y}]\!] = [\![\pi(\mathbf{X})]\!]$ for a random permutation π that is unknown to both parties.

Notation 5.1. Henceforth, for simpler representation we use the established predicate notation to denote the general "private equality test" functionality that can be obtained from $\mathcal{F}_{\text{EQZ}}$. Recall that the output of predicate evaluation is always additively split, while the operands may or may not be split; for example, $(\llbracket a \rrbracket == b)$ gives $\llbracket 1 \rrbracket$ if $a == b$, and $(a == b)$ gives $\llbracket 0 \rrbracket$ if $a \neq b$.

5.6.1 FindSPHF: Randomized Label-then-Unify

Protocol 5.1, Label-then-Unify, securely computes FindSPHF in two phases:

1. **Label items randomly and uniquely, then unify labels for duplicate items**: Alice (resp. Bob) creates a length-γ vector $\mathbf{U}$ (resp. $\mathbf{V}$) with items of $S^{\mathcal{A}}$ (resp. $S^{\mathcal{B}}$) in arbitrary order. Alice and Bob assign random distinct labels to entries in $\mathbf{U}$ and $\mathbf{V}$ by generating an initial label vector $\llbracket \mathbf{L} \rrbracket$ consisting of hash values $\{0, 1, \cdots, 2\gamma - 1\}$ in a random order; i.e., $\llbracket \mathbf{L} \rrbracket = \llbracket \pi([0, 1, \ldots, 2\gamma - 1]) \rrbracket$, where π is unknown to both parties. Pairing $\mathbf{U}$ and the first γ entries of $\llbracket \mathbf{L} \rrbracket$ gives an initial mapping of item-hash pairs $\{(u_i, \llbracket l_i \rrbracket)\}$ for $1 \leq i \leq \gamma$. Similarly, pairing $\mathbf{V}$ and the second γ entries of $\llbracket \mathbf{L} \rrbracket$ gives an initial mapping $\{(v_j, \llbracket l_{\gamma+j} \rrbracket)\}$ for $1 \leq j \leq \gamma$. However, this initial mapping is inconsistent (by construction) when $S^{\mathcal{A}} \cap S^{\mathcal{B}}$ is non-empty, in the sense that Alice's and Bob's initial mappings give different hash values for any $x \in S^{\mathcal{A}} \cap S^{\mathcal{B}}$; more formally

$$\forall x \in S^{\mathcal{A}} \cap S^{\mathcal{B}}\ \exists\, 1 \leq i, j \leq \gamma \ \text{ s.t. } \ u_i = x = v_j \ \wedge \ l_i \neq l_{\gamma+j} \tag{5.4}$$

The following Protocol 5.1 unifies such inconsistent labels for x by overwriting $\llbracket l_{\gamma+j} \rrbracket$ with $\llbracket l_i \rrbracket$. To do so, Alice and Bob first compute a $\gamma \times \gamma$ indicator matrix $\llbracket \mathbf{\Delta} \rrbracket$ that obliviously captures the position of duplicate items, i.e., $\llbracket \delta_{i,j} \rrbracket = (u_i == v_j)$. Then, an updated label vector $\llbracket \mathbf{L}' \rrbracket$ is computed as follows:

$$\begin{aligned} \llbracket l'_i \rrbracket &= \llbracket l_i \rrbracket && \text{for } 1 \leq i \leq \gamma \\ \llbracket l'_{\gamma+j} \rrbracket &= \llbracket \delta_{i,j} \cdot l_i \rrbracket + \llbracket (1 - \delta_{i,j}) \cdot l_j \rrbracket && \text{for } 1 \leq j \leq \gamma \ \wedge \ \llbracket \delta_{i,j} \rrbracket = 1 \end{aligned} \tag{5.5}$$

Remark 5.1. Note that

(a) The initial label vector $[\![\mathbf{L}]\!] = [\![\pi([0, 1, \cdots, 2\gamma - 1])]\!]$ is independent of $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$ and can be computed ahead of time, which allows Protocol 5.1 to take $[\![\mathbf{L}]\!]$ as an auxiliary input.

(b) The random permutation π determines the choice of $\rho \in \mathsf{Inj}_{\Sigma,2\gamma}(S)$; π enforces injectivity on S, while allowing (implicit) arbitrary mapping of items in $\Sigma \setminus S$.

(c) Alice and Bob cannot determine which labels have been unified because any label $[\![l_{\gamma+j}]\!]$ is overwritten either with itself (if $[\![\delta_{i,j}]\!] = 0$ for all $1 \leq i \leq \gamma$), or with some $[\![l_i]\!]$ (if $[\![\delta_{i,j}]\!] = 1$ for exactly one index i). Recall that the secure multiplication in split form implicitly re-splits the resulting secret shares, which prevents parties from recognizing if a label has been overwritten with itself or with another label.

2. **Return private outputs**: At this point, Alice and Bob have an SPHF in split form. Alice sends $[\![l'_{\gamma+1}]\!]^{\mathcal{A}}, [\![l'_{\gamma+2}]\!]^{\mathcal{A}}, \ldots, [\![l'_{2\gamma}]\!]^{\mathcal{A}}$ to Bob, and Bob sends $[\![l'_1]\!]^{\mathcal{B}}, [\![l'_2]\!]^{\mathcal{B}}, \ldots, [\![l'_\gamma]\!]^{\mathcal{B}}$ to Alice. Following this, each party computes their respective private output:

$$\begin{aligned} \varrho^{\mathcal{A}} &= \{(x, \rho(x))\}_{x \in S^{\mathcal{A}}} = \{(u_i, l'_i)\} \quad \text{for } 1 \leq i \leq \gamma \\ \varrho^{\mathcal{B}} &= \{(x, \rho(x))\}_{x \in S^{\mathcal{B}}} = \{(v_j, l'_{\gamma+j})\} \text{ for } 1 \leq j \leq \gamma \end{aligned} \tag{5.6}$$

Lemma 5.2 (Correctness of Protocol 5.1). *A function $\rho : \Sigma \to \{0, 1 \ldots, 2\gamma - 1\}$ obtained by* Label-then-Unify *is a minimal perfect SPHF for $S = S^{\mathcal{A}} \cup S^{\mathcal{B}}$.*

Proof. Due to Lemma 5.1, it suffices to prove that the protocol returns a function chosen uniformly at random from $\mathsf{Inj}_{\Sigma,2\gamma}(S)$; below, we prove that for any $\phi \in \mathsf{Inj}_{\Sigma,2\gamma}(S)$

$$\Pr[\rho = \phi] = \frac{1}{|\mathsf{Inj}_{\Sigma,2\gamma}(S)|}, \text{ where } |\mathsf{Inj}_{\Sigma,2\gamma}(S)| = \frac{(2\gamma)!}{(2\gamma - |S|)!} \cdot (2\gamma)^{|\Sigma \setminus S|} \tag{5.7}$$

Below, we partition Σ into three disjoint sets and analyze ρ's behavior for each one separately:

1. "$S' = S \setminus (S^{\mathcal{A}} \cap S^{\mathcal{B}})$": Protocol 5.1 merely uses the random permutation π to assign unique hash values to items in S'; the unification process does *not* modify these hashes. Thus, choice of π fixes one of the $\frac{(2\gamma)!}{(2\gamma - |S'|)!}$ possible mappings for items in S'.

2. "$S^{\mathcal{A}} \cap S^{\mathcal{B}}$": Step 1 in the protocol assigns *two* distinct and unique hash values to each $x \in S^{\mathcal{A}} \cap S^{\mathcal{B}}$; then, Step 3 overwrites one of these labels with the other one. Hence, the choice of π determines one of the $\frac{(2\gamma-|S'|)!}{(2\gamma-|S'|-|S^{\mathcal{A}} \cap S^{\mathcal{B}}|)!}$ possible mappings for items in $S^{\mathcal{A}} \cap S^{\mathcal{B}}$, and it does not matter which one is used.

3. "$\Sigma \setminus S$": Protocol 5.1 allows (implicit) arbitrary mapping of items in $\Sigma \setminus S$ to hash values $\{0, 1 \ldots, 2\gamma - 1\}$; there are $(2\gamma)^{|\Sigma \setminus S|}$ such mappings.

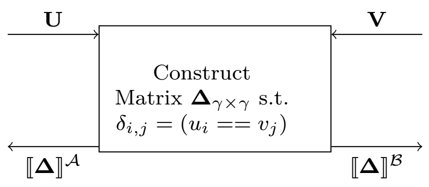
FindSPHF: Label-then-Unify (public parameter γ)

Alice	**Bob**
On input $S^{\mathcal{A}}$	On input $S^{\mathcal{B}}$
Aux. input $[\![\mathbf{L}]\!]^{\mathcal{A}}$	Aux. input $[\![\mathbf{L}]\!]^{\mathcal{B}}$

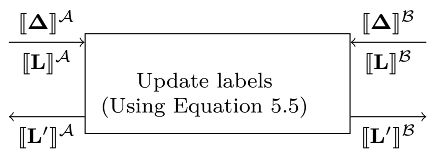
.... Phase 1: Label items randomly and uniquely, then unify labels for duplicate item ....

1 : Create vector $\mathbf{U}$ that contains items of $S^{\mathcal{A}}$ in an arbitrary order — Create vector $\mathbf{V}$ that contains items of $S^{\mathcal{B}}$ in an arbitrary order

2 :


$\mathbf{U}$
$\mathbf{V}$
Construct
Matrix $\mathbf{\Delta}_{\gamma\times\gamma}$ s.t.
$\delta_{i,j} = (u_i == v_j)$
$[\![\mathbf{\Delta}]\!]^{\mathcal{A}}$
$[\![\mathbf{\Delta}]\!]^{\mathcal{B}}$

3 : $[\![\mathbf{\Delta}]\!]^{\mathcal{A}}, [\![\mathbf{L}]\!]^{\mathcal{A}}$ — $[\![\mathbf{\Delta}]\!]^{\mathcal{B}}, [\![\mathbf{L}]\!]^{\mathcal{B}}$


$[\![\mathbf{\Delta}]\!]^{\mathcal{A}}$
$[\![\mathbf{L}]\!]^{\mathcal{A}}$
$[\![\mathbf{\Delta}]\!]^{\mathcal{B}}$
$[\![\mathbf{L}]\!]^{\mathcal{B}}$
Update labels
(Using Equation 5.5)
$[\![\mathbf{L}']\!]^{\mathcal{A}}$
$[\![\mathbf{L}']\!]^{\mathcal{B}}$

.... Phase 2: Return private outputs ....

4 : $\xrightarrow{[\![l'_{\gamma+1}]\!]^{\mathcal{A}}, [\![l'_{\gamma+2}]\!]^{\mathcal{A}}, \ldots, [\![l'_{2\gamma}]\!]^{\mathcal{A}}}$

$\xleftarrow{[\![l'_{1}]\!]^{\mathcal{B}}, [\![l'_{2}]\!]^{\mathcal{B}}, \ldots, [\![l'_{\gamma}]\!]^{\mathcal{B}}}$

5 : $\varrho^{\mathcal{A}} = \{(u_i, l'_i)\}_{1 \le i \le \gamma}$ — $\varrho^{\mathcal{B}} = \{(v_j, l'_{\gamma+j})\}_{1 \le j \le \gamma}$

6 : **return** $\varrho^{\mathcal{A}}$ — **return** $\varrho^{\mathcal{B}}$

Protocol 5.1. Secure realization of FindSPHF approach in constant rounds

Hence, the function ρ obtained by Protocol 5.1 is chosen uniformly at random among

$$\frac{(2\gamma)!}{(2\gamma - |S'|)!} \cdot \frac{(2\gamma - |S'|)!}{(2\gamma - |S'| - |S^{\mathcal{A}} \cap S^{\mathcal{B}}|)!} \cdot (2\gamma)^{|\Sigma \setminus S|}$$

possible functions in $\mathsf{Inj}_{\Sigma,2\gamma}(S)$. As a result, recalling that S' and $S^{\mathcal{A}} \cap S^{\mathcal{B}}$ are disjoint, we get

$$\begin{aligned}\Pr[\rho = \phi] &= \frac{(2\gamma - |S'|)!}{(2\gamma)!} \cdot \frac{(2\gamma - |S'| - |S^{\mathcal{A}} \cap S^{\mathcal{B}}|)!}{(2\gamma - |S'|)!} \cdot \frac{1}{(2\gamma)^{|\Sigma \setminus S|}} \\ &= \frac{(2\gamma - |S|)!}{(2\gamma)! \cdot (2\gamma)^{|\Sigma \setminus S|}} = \frac{1}{|\mathsf{Inj}_{\Sigma,2\gamma}(S)|}\end{aligned}$$

□

Lemma 5.3 (Security of Protocol 5.1)**.** *Protocol* Label-then-Unify*, on Alice's and Bob's respective private inputs $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$, that computes the functionality $(\varrho^{\mathcal{A}}, \varrho^{\mathcal{B}}) = \mathcal{F}_{\text{SPHF}}(S^{\mathcal{A}}, S^{\mathcal{B}})$ is secure in the semi-honest threat model. Moreover, given black-box access to the $\mathcal{F}_{\text{EQZ}}$ functionality, the protocol is unconditionally secure in the ABB model of computation.*

Proof. It suffices to discuss the security of steps 2 and 3: Step 2 merely uses the "equal to zero" functionality ($\mathcal{F}_{\text{EQZ}}$ from Chapter 3) in a black-box manner, and Step 3 applies the mapping summarized in Equation 5.5 that only uses secure addition and multiplication in split form. Hence, Protocol 5.1 is unconditionally secure in the ABB model of computation.

□

5.6.2 FindSPHF: Randomized Merge-then-Unify

The key observation for improving computation and communication complexities of the Label-then-Unify protocol is that the $\gamma \times \gamma$ indicator matrix $[\![\mathbf{\Delta}]\!]$ computed in Step 2 of Protocol 5.1 has *at most* one non-zero entry in each row and each column. This allows reducing the dimension of $[\![\mathbf{\Delta}]\!]$ from $\gamma \times \gamma$ to $1 \times 2\gamma$ through the use of *Oblivious Merge* functionality as illustrated in Figure 5.5: Given two sorted input vectors $[\![\mathbf{X}]\!]$ and $[\![\mathbf{Y}]\!]$, the functionality $([\![\mathbf{Z}]\!]^{\mathcal{A}}, [\![\mathbf{Z}]\!]^{\mathcal{B}}) = \mathcal{F}_{\text{MERGE}}\big(([\![\mathbf{X}]\!]^{\mathcal{A}}, [\![\mathbf{Y}]\!]^{\mathcal{A}}), ([\![\mathbf{X}]\!]^{\mathcal{B}}, [\![\mathbf{Y}]\!]^{\mathcal{B}})\big)$ is such that $[\![\mathbf{Z}]\!] = [\![\textsc{Merge}(\mathbf{X}, \mathbf{Y})]\!]$, where $\textsc{Merge}(\mathbf{X}, \mathbf{Y})$ is the sorted combination of $\mathbf{X}$ and $\mathbf{Y}$. For computing this functionality, we shall use the results of Jónsson *et al.* [114] that give a secure two-party protocol adaptation of Batcher's merge [115] for secure merging in split form. For two input vectors of length γ, the secure merge protocol of [114] requires $\mathcal{O}(\gamma \log \gamma)$ computation and communication in $\mathcal{O}(\log \gamma)$ rounds.

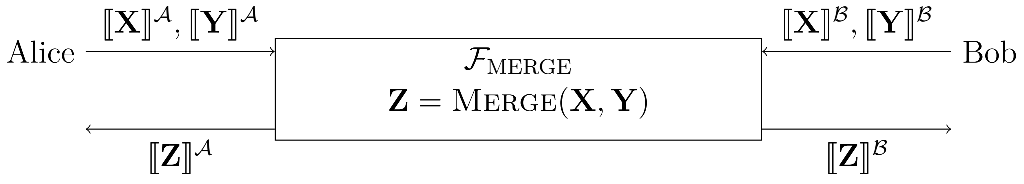

Alice
$[\![\mathbf{X}]\!]^{\mathcal{A}}, [\![\mathbf{Y}]\!]^{\mathcal{A}}$
$\mathcal{F}_{\text{MERGE}}$
$\mathbf{Z} = \text{Merge}(\mathbf{X}, \mathbf{Y})$
$[\![\mathbf{X}]\!]^{\mathcal{B}}, [\![\mathbf{Y}]\!]^{\mathcal{B}}$
Bob
$[\![\mathbf{Z}]\!]^{\mathcal{A}}$
$[\![\mathbf{Z}]\!]^{\mathcal{B}}$

Figure 5.5. $([\![\mathbf{Z}]\!]^{\mathcal{A}}, [\![\mathbf{Z}]\!]^{\mathcal{B}}) = \mathcal{F}_{\text{MERGE}}\big(([\![\mathbf{X}]\!]^{\mathcal{A}}, [\![\mathbf{Y}]\!]^{\mathcal{A}}), ([\![\mathbf{X}]\!]^{\mathcal{B}}, [\![\mathbf{Y}]\!]^{\mathcal{B}})\big)$, which takes sorted vectors $[\![\mathbf{X}]\!]$ and $[\![\mathbf{Y}]\!]$ as input, and outputs their sorted combination as vector $[\![\mathbf{Z}]\!]$ in split form.

Following the observation above, we propose another embodiment for our FindSPHF approach, Merge-then-Unify, which needs $\mathcal{O}(\gamma \log \gamma)$ computation and communication in $\mathcal{O}(\log \gamma)$ rounds. Merge-then-Unify consists of three phases:

1. **Mark ownership, sort and merge**: First, Alice and Bob locally mark their private inputs so they can determine the owner of each item after merging, randomly permuting, and processing the data in split form. To do so, they locally construct marked sets $S^{\mathcal{A}} \times \{0\}$ and $S^{\mathcal{B}} \times \{1\}$, thereby, pairing each item x with an ownership indicator (denoted by $\text{Owner}(x)$).

   After marking ownership, Alice and Bob *locally sort* their private sets and obtain strictly increasing length-γ vectors $\mathbf{U}$ and $\mathbf{V}$. Then, they obliviously merge $\mathbf{U}$ and $\mathbf{V}$ to obtain the sorted combination of both, i.e., a length-2γ vector $[\![\mathbf{Z}]\!]$. Note that for any $x \in S^{\mathcal{A}} \cap S^{\mathcal{B}}$, $(x, 0)$ from Alice's input *immediately* precedes $(x, 1)$ from Bob's input (the ownership bits will be also split).

2. **Label items randomly and uniquely, then unify labels for duplicate items**: Alice and Bob assign random distinct labels from $\{0, 1 \ldots, 2\gamma - 1\}$ to items in $[\![\mathbf{Z}]\!]$ by pairing $[\![\mathbf{Z}]\!]$ with an initial label vector $[\![\mathbf{L}]\!] = [\![\pi_1([0, 1, \cdots, 2\gamma - 1])]\!]$, where π_1 is a double-blind permutation. This results in an initial mapping $\{([\![z_i]\!], [\![l_i]\!])\}_{1 \le i \le 2\gamma}$ that is

inconsistent (by construction) when $S^{\mathcal{A}} \cap S^{\mathcal{B}}$ is non-empty, in the sense that Alice's and Bob's initial mappings give different hash values for any $x \in S^{\mathcal{A}} \cap S^{\mathcal{B}}$; more formally

$$\forall x \in S^{\mathcal{A}} \cap S^{\mathcal{B}} \; \exists \, 2 \leq i \leq 2\gamma \;\; \text{s.t.} \;\; z_{i-1} = (x, 0) \;\wedge\; z_i = (x, 1) \wedge l_{i-1} \neq l_i \tag{5.8}$$

This phase unifies the inconsistent labels for duplicates by overwriting $[\![l_i]\!]$ with $[\![l_{i-1}]\!]$. To do so, Alice and Bob compute a $1 \times 2\gamma$ vector $[\![\boldsymbol{\Delta}]\!]$ that obliviously captures the position of duplicates, i.e., $[\![\delta_1]\!] = 0$ and $[\![\delta_i]\!] = (z_{i-1} == z_i)$ for $2 \leq i \leq 2\gamma$. Then, they compute an updated label vector $[\![\mathbf{L}']\!]$ such that

$$\begin{aligned} &[\![l'_1]\!] = [\![l_1]\!] \\ &[\![l'_i]\!] = [\![\delta_i \cdot l_{i-1}]\!] + [\![(1 - \delta_i) \cdot l_i]\!] \qquad \text{for } 2 \leq i \leq 2\gamma \end{aligned} \tag{5.9}$$

Remark 5.2. Note that (similar to Remark 5.1)

(a) The initial label vector $[\![\mathbf{L}]\!] = [\![\pi_1([0, 1, \cdots, 2\gamma - 1])]\!]$ is independent of $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$ and can be computed ahead of time.

(b) The random permutation π_1 determines the choice of $\rho \in \mathsf{Inj}_{\Sigma, 2\gamma}(S)$; π_1 enforces injectivity on S, while allowing (implicit) arbitrary mapping of items in $\Sigma \setminus S$.

(c) Alice and Bob cannot determine which labels have been unified because any label $[\![l_i]\!]$ for $2 \leq i \leq 2\gamma$ is overwritten either with itself (if $[\![\delta_i]\!] = 0$), or with $[\![l_{i-1}]\!]$ (if $[\![\delta_i]\!] = 1$). Recall that the secure multiplication in split form implicitly re-splits the secret shares, which prevents parties from recognizing if a label has been overwritten with itself or with another label.

3. **Shuffle, and return private outputs**: After unifying labels for duplicates, Alice and Bob have an SPHF in split form. Since $[\![\mathbf{Z}]\!]$ is sorted, revealing the ownership bits would reveal information about the rank of items in S, and thus about $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$. Hence, before revealing ownership bits, Alice and Bob shuffle both $[\![\mathbf{Z}]\!]$ and $[\![\mathbf{L}']\!]$ using

a double-blind permutation π_2 to obtain $[\![\mathbf{Z}']\!] = \pi_2([\![\mathbf{Z}]\!])$ and $[\![\mathbf{L}'']\!] = \pi_2([\![\mathbf{L}']\!])$. At this point, Alice and Bob safely reveal the ownership bits for all items in $[\![\mathbf{Z}']\!]$ and obtain

$$\begin{aligned} I^{\mathcal{A}} &= \{i \mid 1 \le i \le 2\gamma \wedge \textsc{Owner}(z'_i) = 0\}, \\ I^{\mathcal{B}} &= \{i \mid 1 \le i \le 2\gamma \wedge \textsc{Owner}(z'_i) = 1\}, \end{aligned} \tag{5.10}$$

Then, Alice sends $\{[\![z'_i]\!]^{\mathcal{A}}, [\![l''_i]\!]^{\mathcal{A}}\}_{i \in I^{\mathcal{B}}}$ to Bob, and Bob sends $\{[\![z'_i]\!]^{\mathcal{B}}, [\![l''_i]\!]^{\mathcal{B}}\}_{i \in I^{\mathcal{A}}}$ to Alice. Finally, each party computes his/her respective private output as follows:

$$\begin{aligned} \varrho^{\mathcal{A}} &= \{(x, \rho(x))\}_{x \in S^{\mathcal{A}}} = \{(z'_i, l''_i)\}_{i \in I^{\mathcal{A}}} \\ \varrho^{\mathcal{B}} &= \{(x, \rho(x))\}_{x \in S^{\mathcal{B}}} = \{(z'_i, l''_i)\}_{i \in I^{\mathcal{B}}} \end{aligned} \tag{5.11}$$

Lemma 5.4 (Correctness of Merge-then-Unify). *A function $\rho : \Sigma \to \{0, 1 \ldots, 2\gamma - 1\}$ obtained by Merge-then-Unify is a minimal perfect SPHF for $S = S^{\mathcal{A}} \cup S^{\mathcal{B}}$.*

Proof. The proof that ρ satisfies Definition 5.1 follows a similar argument to the proof of Lemma 5.2; i.e., the function ρ is chosen uniformly at random from $\mathsf{Inj}_{\Sigma,2\gamma}(S)$ due to the uniform choice of permutation π_1 in the second phase. $\square$

Lemma 5.5 (Security of Protocol 5.1). *Protocol Merge-then-Unify, on Alice's and Bob's respective private inputs $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$, that computes the functionality $(\varrho^{\mathcal{A}}, \varrho^{\mathcal{B}}) = \mathcal{F}_{\mathrm{SPHF}}(S^{\mathcal{A}}, S^{\mathcal{B}})$ is secure in the semi-honest threat model. Moreover, given black-box access to the $\mathcal{F}_{\mathrm{EQZ}}$, $\mathcal{F}_{\mathrm{MERGE}}$ and $\mathcal{F}_{\mathrm{DBP}}$ functionalitities, the protocol is unconditionally secure in the ABB model of computation.*

Proof. Similar to the security argument in the proof of Lemma 5.3, all joint computations are consist of either (i) a black-box access to one of the functionalities $\mathcal{F}_{\mathrm{EQZ}}$, $\mathcal{F}_{\mathrm{MERGE}}$ and $\mathcal{F}_{\mathrm{DBP}}$ for which there are secure realizations or (ii) secure arithmetic in split form. Hence, it suffices to ensure that neither party learns anything about the other party's private data due to the sorted order of items in vector $\mathbf{Z}$; this follows from the use of random permutations π_1 and π_2 in a double-blind manner:

- The use of random permutation π_1 in Phase 2 of the protocol guarantees that the label assigned to any item z_i has no correlation with the rank of that item.

- The use of random permutation π_2 in Phase 3 of the protocol before revealing the ownership bits for all items ensures that neither party learns anything about the rank and order of items. $\square$

5.7 Distribution: Probabilistic Input Partitioning

Although the FindSPHF approach gives a solution to both secure perfect hashing and secure two-party input-size reduction problems, it requires either quadratic computation and communication (Label-then-Unify) or a logarithmic number of rounds (Merge-then-Unify). In this section, we further improve the performance of our constructions by building on the FindSPHF approach. To do so, we use a standard balls and bins analysis [107] to create m subproblems of size $\mathcal{O}(\log m)$ through a consistent partitioning of private sets $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$. Then, each subproblem will be solved independently using the FindSPHF approach; finally, our scheme combines the obtained SPHFs for these subproblems to provide an overall SPHF for the original set S. According to the balls and bins analysis, if m balls (input items) are distributed into m bins uniformly and independently at random, then the probability of having a bin with more than γ balls is at most $m \cdot (\frac{e}{\gamma})^{\gamma}$, which is not more than $\frac{1}{m}$ for sufficiently large m and a choice of $\gamma \geq \frac{3 \ln m}{\ln \ln m}$ [107].

For a function $\Psi : \Sigma \to \{0, 1, \cdots, m-1\}$, we define bins (subproblems) S_i for $0 \leq i \leq m-1$ as follows (private bins $S_i^{\mathcal{A}}$ and $S_i^{\mathcal{B}}$ are defined accordingly):

$$S_i = \{x \mid x \in S \wedge \Psi(x) = i\} \tag{5.12}$$

We are interested in finding a function Ψ that, when applied to the private sets $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$ separately, results in a maximum load bin of at most γ items; formally,

$$\max_{0 \leq i < m} (|S_i^{\mathcal{A}}|) \leq \gamma \qquad \text{and} \qquad \max_{0 \leq i < m} (|S_i^{\mathcal{B}}|) \leq \gamma \tag{5.13}$$

If Equation 5.13 holds, we say Ψ is a *distributor* for S. Note that Equation 5.13 implies that $\max_{0 \leq i < m}(|S_i|) \leq 2\gamma$.

Remark 5.3 (Relaxed PHF [98])**.** For any set S and an integer $k > 0$, a function h is considered to be a (S, k)-PHF if each bin S_i under h contains at most k items. In other words, a (S, k)-PHF guarantees no more than k collisions per hash value. According to this relaxed definition of perfect hashing, a distributor Ψ for S must be both a $(S^{\mathcal{A}}, \gamma)$-PHF and a $(S^{\mathcal{B}}, \gamma)$-PHF; this guarantees Ψ being a $(S, 2\gamma)$-PHF. Clearly, additional security measures (discussed below) are necessary for a distributor function.

Because of the security and performance requirements, there are two major challenges for finding a distributor Ψ. We first (in sections 5.7.1 and 5.7.2) discuss these challenges, and how we address them. Then (in Section 5.7.3), we propose our FindDistribtor protocol to obtain such a distributor Ψ for set S.

5.7.1 Challenge 1: Preventing Exploitation of Distributor Function's Structure

Given a distributor Ψ and any $x \in \Sigma$, neither party should be able to unilaterally compute $\Psi(x)$. Otherwise, an adversarial party (say, Alice) may exploit Ψ to gain information about the other party's set, e.g., through a membership-testing attack; for example, Alice would easily find $\gamma' > \gamma$ items which lay in the same bin under Ψ, and learn that some of these γ' items are not in $S^{\mathcal{B}}$ (this would violate the security property in Definition 5.1). In order to prevent such a structure exploitation, we first define the Distribution functionality (as illustrated in Figure 5.6) analogous to the SPHF functionality: Alice and Bob must learn only the mapping of their respective private sets under Ψ, which gives the desired partitioning. To satisfy all the requirements, Ψ must be a *cooperatively and securely computable PRF* in the sense that:

- Cooperatively computable Ψ: A querier who wants to compute $\Psi(x)$ for a private query x *must* need the other party's cooperation via execution of a protocol. Any evaluation of $\Psi(x)$ without cooperation of both parties must be no better than a random guess.

- Securely Computable Ψ: A protocol that enables the querier party to compute $\Psi(x)$ must be secure with respect to the querier's input and output; i.e., the other party involved in computing $\Psi(x)$ must not learn anything about x and $\Psi(x)$.

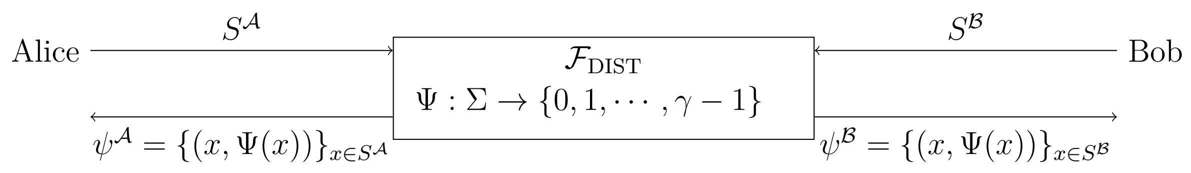

Alice
$S^{\mathcal{A}}$
$\mathcal{F}_{\text{DIST}}$
$\Psi : \Sigma \to \{0, 1, \cdots, \gamma - 1\}$
$S^{\mathcal{B}}$
Bob
$\psi^{\mathcal{A}} = \{(x, \Psi(x))\}_{x \in S^{\mathcal{A}}}$
$\psi^{\mathcal{B}} = \{(x, \Psi(x))\}_{x \in S^{\mathcal{B}}}$

Figure 5.6. Distribution functionality $(\psi^{\mathcal{A}}, \psi^{\mathcal{B}}) = \mathcal{F}_{\text{DIST}}(S^{\mathcal{A}}, S^{\mathcal{B}})$, where (i) Ψ is cooperatively and securely computable; and, (ii) the maximum load bins for $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$ under Ψ have at most γ items, i.e., $|S_i^{\mathcal{A}}|, |S_i^{\mathcal{B}}| \leq \gamma$ for $0 \leq i < m$.

To obtain the properties above, we combine the notions of OPRF and DPRF (both reviewed in Section 2.3) that leads to the formulation of an *Oblivious Distributed PRF* (ODPRF). Definition 5.2 formalizes 2-out-of-2 ODPRFs (denoted by ODPRF_2^2). Then, we propose Protocol 5.2 that combines Micali and Sidney's xor-PRFs [108] with any standard OPRF, $\langle F, \text{Prot}_F^{\text{OPRF}} \rangle$, to obtain an ODPRF_2^2.

Definition 5.2 (Oblivious Distributed PRF)**.** *Let the keyed function Ψ be a PRF with the security parameter τ. A 2-out-of-2 oblivious computation of Ψ is a triple of efficient protocols $\langle \textsc{Gen}, \Psi^{\mathcal{A}:\mathcal{B}}, \Psi^{\mathcal{B}:\mathcal{A}} \rangle$ such that*

- *$\textsc{Gen}(1^\tau)$ generates a key $k = (\alpha, \beta)$ consisting of two secret keys α and β, one for each party.*

- *$\Psi^{\mathcal{A}:\mathcal{B}}$ is a secure protocol that enables Alice to efficiently compute $\Psi_k(x)$ for her private query x with Bob's cooperation. Moreover, Alice cannot evaluate $\Psi_k(x)$ without invoking $\Psi^{\mathcal{A}:\mathcal{B}}$; i.e., for a truly random function $f(\cdot)$ and any PPT distinguisher D*

$$|\Pr[D^{\Psi_k(\cdot)}(1^\tau, \alpha) = 1] - \Pr[D^{f(\cdot)}(1^\tau, \alpha) = 1]| \leq \mathsf{negl}(\tau) \tag{5.14}$$

- *$\Psi^{\mathcal{B}:\mathcal{A}}$ is a secure protocol that enables Bob to efficiently compute $\Psi_k(x)$ for his private query x with Alice's cooperation. Moreover, Bob cannot evaluate $\Psi_k(x)$ without invoking $\Psi^{\mathcal{B}:\mathcal{A}}$; i.e., for a truly random function $f(\cdot)$ and any PPT distinguisher D*

$$|\Pr[D^{\Psi_k(\cdot)}(1^\tau, \beta) = 1] - \Pr[D^{f(\cdot)}(1^\tau, \beta) = 1]| \leq \mathsf{negl}(\tau) \tag{5.15}$$

In both equations 5.14 and 5.15, the first probability is over the randomness of D and choices of secret keys α and β; the second probability is over the randomness of D and choices of secret keys α, β and the truly random function f; superscripts of D denote oracle access to the corresponding function.

Lemma 5.6. *Let $\langle F, Prot_F^{OPRF} \rangle$ be an OPRF with the security parameter τ. Then, the function $\Psi_k(x) = F_\alpha(x) \oplus F_\beta(x)$, where α and β are respectively, Alice's and Bob's secret keys, together with the triple $\langle \textsc{Gen}, \Psi^{\mathcal{A}:\mathcal{B}}, \Psi^{\mathcal{B}:\mathcal{A}} \rangle$ as in Protocol 5.2 is an ODPRF$_2^2$.*

Proof. Since subprotocol Gen generates two random bit-strings of length τ as secret keys α and β, $\alpha = \beta$ may happen only with the negligible probability $2^{-\tau}$; thus, we assume $\alpha \neq \beta$. Also, we consider secure computation of F through $\mathrm{Prot}_F^{\mathrm{OPRF}}$ in a black-box manner.

The triple $\langle \textsc{Gen}, \Psi^{\mathcal{A}:\mathcal{B}}, \Psi^{\mathcal{B}:\mathcal{A}} \rangle$ in Protocol 5.2 satisfies Definition 5.2: Gen generates the secret keys α and β, each of them being a key for F. Protocol $\Psi^{\mathcal{A}:\mathcal{B}}$ (resp. $\Psi^{\mathcal{B}:\mathcal{A}}$) enables

Construction of ODPRF$_2^2$ (security parameter τ)

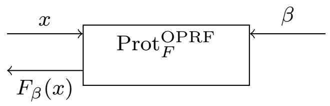
Alice **Bob**

Subprotocol Gen(1^τ): Generate key $k = (\alpha, \beta)$

1 : $\alpha \xleftarrow{\$} \{0,1\}^\tau$ $\beta \xleftarrow{\$} \{0,1\}^\tau$

Subprotocol $\Psi^{\mathcal{A}:\mathcal{B}}$: Compute $\Psi_k(x)$ when Alice is the querier

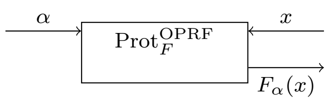
2 : On input $x \in S^{\mathcal{A}}, \alpha$ On input β



3 : **return** $\Psi_k(x) = F_\alpha(x) \oplus F_\beta(x)$ **return** $\perp$

Subprotocol $\Psi^{\mathcal{B}:\mathcal{A}}$: Compute $\Psi_k(x)$ when Bob is the querier

4 : On input α On input $x \in S^{\mathcal{B}}, \beta$



5 : **return** $\perp$ **return** $\Psi_k(x) = F_\alpha(x) \oplus F_\beta(x)$

Protocol 5.2. Construction of ODPRF$_2^2$

Alice (resp. Bob) to securely evaluate $\Psi_k(x)$ via invoking $\mathrm{Prot}_F^{\mathrm{OPRF}}$ once, followed by only local computations. Hence, both $\Psi^{\mathcal{A}:\mathcal{B}}$ and $\Psi^{\mathcal{B}:\mathcal{A}}$ are as secure as the underlying OPRF.

It remains to prove that equations 5.14 and 5.15 hold; i.e., as long as at least one of the secret keys α and β is unknown, $\Psi_k(\cdot)$ is indistinguishable from a truly random function $f(\cdot)$. Since Ψ_k is symmetric by construction, we prove only Equation 5.14. For the sake of contradiction, assume there exists a PPT distinguisher D and a non-negligible ε s.t.

$$|\Pr[D^{\Psi_k(\cdot)}(1^\tau, \alpha) = 1] - \Pr[D^{f(\cdot)}(1^\tau, \alpha) = 1]| = \varepsilon$$

Below, we use D to construct a PPT distinguisher $\bar{D}$, which distinguishes $F_\beta(\cdot)$ from a truly random function $\bar{f}(\cdot)$ with probability ε. $\bar{D}$ intervenes the messages between D and an oracle (namely, Orac) that computes either F_β or $\bar{f}$ as follows:

- On D's query x, $\bar{D}$ passes it to Orac and receives the oracle's response $\mathsf{Orac}(x)$.
- $\bar{D}$ computes $\mathsf{Orac}(x) \oplus F_\alpha(x)$ and sends it to D as the oracle's response to query x.

In the end, $\bar{D}$ returns 1 if and only if D returns 1. D runs in polynomial-time and F is efficiently computable, thus, $\bar{D}$ also runs in polynomial-time. If the oracle computes F_β, then, $\mathsf{Orac}(x) \oplus F_\alpha(x) = \Psi_k(x)$; or else, if the oracle computes a random function $\bar{f}$, then, $f(x) = \mathsf{Orac}(x) \oplus F_\alpha(x) = \bar{f}(x) \oplus F_\alpha(x)$ is also a truly random function. Thus,

$$|\Pr[\bar{D}^{F_\beta(\cdot)}(1^\tau, \alpha) = 1] - \Pr[\bar{D}^{\bar{f}(\cdot)}(1^\tau, \alpha) = 1]| = \varepsilon,$$

which contradicts the assumption that F is a PRF. $\square$

Corollary 5.1. *Assuming a PPT adversary, the ODPRF$_2^2$ in Lemma 5.6 is a cooperatively and securely computable function:*

1. *Due to Lemma 5.6, in the absence of at least one of the secret keys α and β, any evaluation of $\Psi_k(x)$ is no better than a random guess. Thus, the cooperation of both parties is necessary for computing $\Psi_k(\cdot)$ on any input x.*

2. *Computing $\Psi_k(x)$ through $\Psi^{\mathcal{A}:\mathcal{B}}$ (resp. $\Psi^{\mathcal{B}:\mathcal{A}}$) does not reveal anything about x and $\Psi_k(x)$ to Bob (resp. Alice). Thus, $\Psi_k(\cdot)$ is securely computable for any input x.*

For a distributor, we need an ODPRF_2^2 with domain Σ and codomain $\{0, 1, \cdots, m-1\}$; thus, the input and output lengths for the underlying PRF must be, respectively, $\ell_{in} = \sigma$ and $\ell_{out} = \log_2 m$.

5.7.2 Challenge 2: Possible Failure of a Distributor Candidate and Its Consequences

In order to find a distributor, Alice and Bob need to randomly choose a *distributor candidate*, and obliviously verify if it satisfies Equation 5.13; if the candidate fails (with probability $p_{\mathsf{fail}} \leq \frac{1}{m}$), then they try another candidate. This is a cause for two concerns:

1. An undetermined number of rounds.
2. Information leakage if the candidate fails. For example, if a candidate fails but gives a maximum load of at most γ for $S^{\mathcal{A}}$, Alice learns that $S^{\mathcal{B}} \setminus S^{\mathcal{A}} \neq \emptyset$ and that violates the privacy of Bob's private set $S^{\mathcal{B}}$.

Our scheme addresses these issues by using a security-performance parameter κ to make the probability of failure arbitrarily small. Initially, Alice and Bob agree on κ independent distributor candidates and verify their validity independently, but in parallel. This reduces the overall probability of failure to $(p_{\mathsf{fail}})^{\kappa} \leq m^{-\kappa}$, which is negligible in κ. This completely handles the concern for undetermined number of rounds. Moreover, it provides a way to address the second concern: It is enough to inform Alice and Bob that one specific candidate is a valid distributor without giving them any information about the validity of other candidates. In order to do this, Alice and Bob compute an indicator vector $[\![\mathbf{D}]\!]$ of length κ such that $[\![d_i]\!] = 1$ if and only if the i^{th} candidate is a valid distributor. Then, they use an auxiliary secure lightweight protocol, PickAnyOne (Protocol 5.3), which returns the index of a valid distributor uniformly at random; Figure 5.7 below, illustrates the corresponding "Pick-any-One" functionality (denoted by $\mathcal{F}_{\mathrm{PA1}}$).

Protocol PickAnyOne securely computes the functionality $(t, t) = \mathcal{F}_{\mathrm{PA1}}([\![\mathbf{D}]\!]^{\mathcal{A}}, [\![\mathbf{D}]\!]^{\mathcal{B}})$ in two phases:

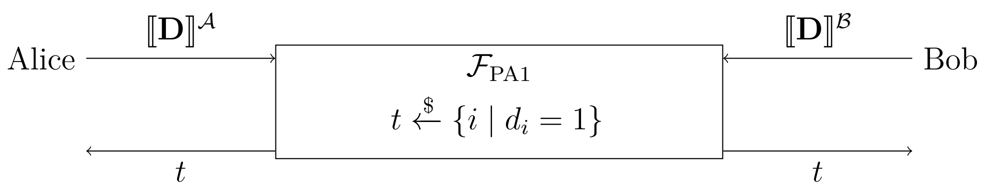

Alice
$[\![\mathbf{D}]\!]^{\mathcal{A}}$
$\mathcal{F}_{\text{PA1}}$
$t \xleftarrow{\$} \{i \mid d_i = 1\}$
$[\![\mathbf{D}]\!]^{\mathcal{B}}$
Bob
t
t

Figure 5.7. "Pick-any-One" functionality $(t, t) = \mathcal{F}_{\text{PA1}}([\![\mathbf{D}]\!]^{\mathcal{A}}, [\![\mathbf{D}]\!]^{\mathcal{B}})$: The input $[\![\mathbf{D}]\!]$ is a length-κ vector of 0s and 1s. If $\{i \mid d_i = 1\} = \emptyset$, then $t = 0$; or else, $t \xleftarrow{\$} \{i \mid d_i = 1\}$.

1. **Random shuffling**: Alice and Bob first (non-interactively) compute a vector $[\![\mathbf{D}']\!]$ such that $[\![d'_i]\!] = [\![d_i]\!] \cdot i$; for any non-zero entry $[\![d_i]\!]$, the corresponding entry $[\![d'_i]\!]$ obliviously stores the index i, i.e.,

$$[\![d'_i]\!] = \begin{cases} i & \text{if } [\![d_i]\!] = 1 \\ 0 & \text{if } [\![d_i]\!] = 0 \end{cases}$$

Then, Alice and Bob use the double-blind permutation functionality, $\mathcal{F}_{\text{DBP}}$, to shuffle the vector $[\![\mathbf{D}']\!]$ and obtain $[\![\mathbf{Y}]\!] = [\![\pi(\mathbf{D}')]\!]$.

2. **Finding the leftmost non-zero entry in $[\![\mathbf{Y}]\!]$**: Any non-zero entry in $[\![\mathbf{Y}]\!]$ corresponds to a non-zero entry in $[\![\mathbf{D}]\!]$. To obliviously search for the leftmost non-zero entry in the shuffled vector $[\![\mathbf{Y}]\!]$, Alice and Bob first compute (non-interactively) a vector $[\![\mathbf{U}]\!]$ as the prefix-sum of $[\![\mathbf{Y}]\!]$, i.e., for $1 \le i \le \kappa$

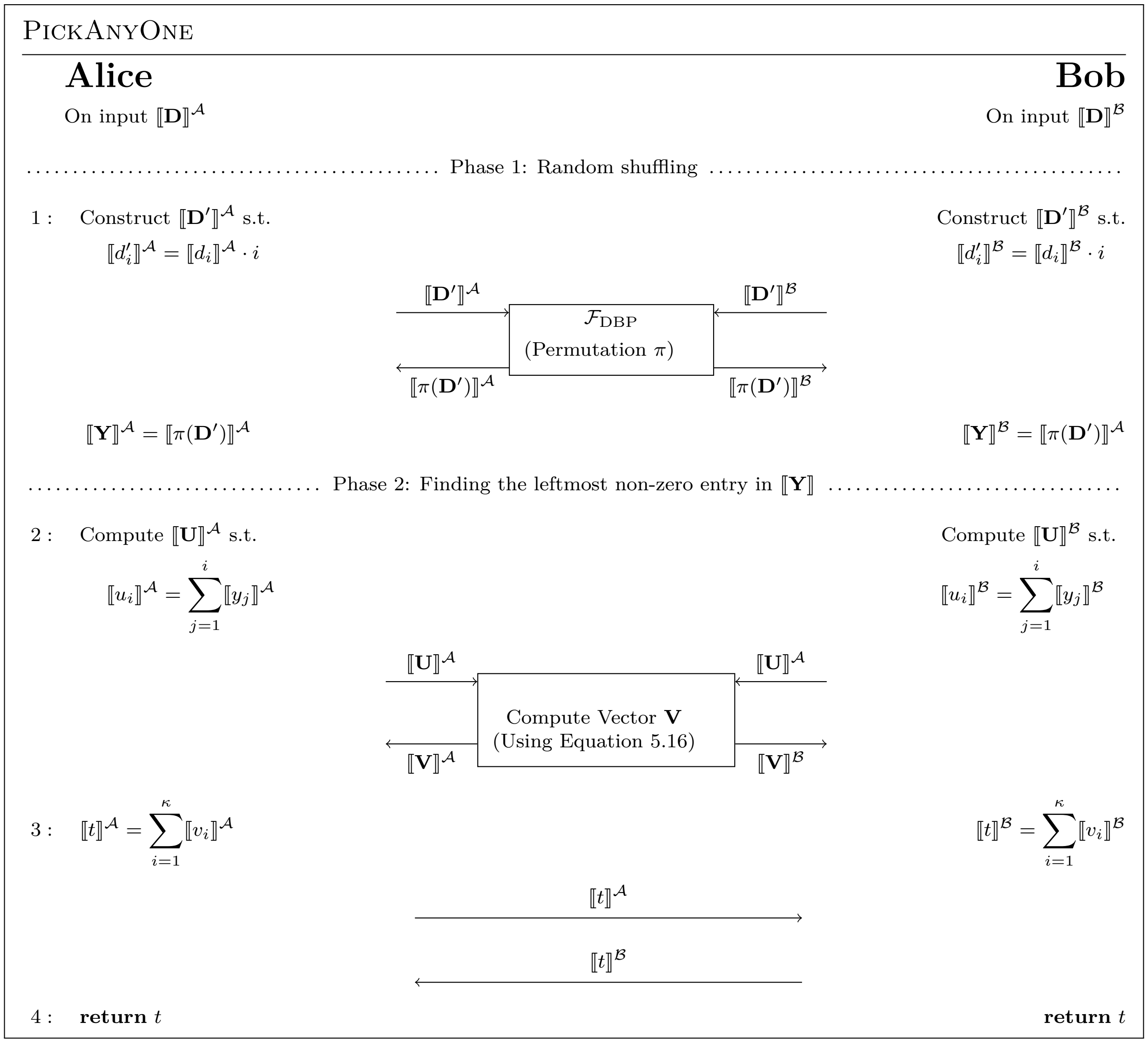
$$[\![u_i]\!] = \sum_{j=1}^{i} [\![y_j]\!]$$

By construction, vector $[\![\mathbf{U}]\!]$ consists of (some) zeros followed by (some) non-zeros. Also, the leftmost non-zero entry in $[\![\mathbf{U}]\!]$ is equal to the leftmost non-zero entry in $[\![\mathbf{Y}]\!]$; to find this entry, it suffices to compute vector $[\![\mathbf{V}]\!]$ as follows:

$$[\![v_i]\!] = \begin{cases} [\![u_i]\!] & \text{if } i = 1 \\ [\![(u_{i-1} == 0) \cdot u_i]\!] & \text{if } 2 \le i \le \kappa \end{cases} \tag{5.16}$$

On the one hand, $[\![\mathbf{Y}]\!] \neq \vec{0}$ results in $[\![\mathbf{V}]\!]$ with *exactly* one non-zero entry equal to the leftmost non-zero entry in $[\![\mathbf{Y}]\!]$; on the other hand, $[\![\mathbf{Y}]\!] = \vec{0}$ results in $[\![\mathbf{V}]\!] = \vec{0}$. Thus, it is enough for Alice and Bob to compute $[\![t]\!] = \sum_{i=1}^{\kappa} [\![v_i]\!]$ (non-interactively), and exchange their respective shares of $[\![t]\!]$ so they both learn the desired output t.

Complexity. Protocol 5.3, PickAnyOne, requires $\mathcal{O}(\kappa)$ computation and communication in $\mathcal{O}(1)$ rounds.



PickAnyOne
Alice
Bob
On input $[\![\mathbf{D}]\!]^{\mathcal{A}}$
On input $[\![\mathbf{D}]\!]^{\mathcal{B}}$
Phase 1: Random shuffling
1 : Construct $[\![\mathbf{D'}]\!]^{\mathcal{A}}$ s.t. $[\![d'_i]\!]^{\mathcal{A}} = [\![d_i]\!]^{\mathcal{A}} \cdot i$
Construct $[\![\mathbf{D'}]\!]^{\mathcal{B}}$ s.t. $[\![d'_i]\!]^{\mathcal{B}} = [\![d_i]\!]^{\mathcal{B}} \cdot i$
$[\![\mathbf{D'}]\!]^{\mathcal{A}}$
$[\![\mathbf{D'}]\!]^{\mathcal{B}}$
$\mathcal{F}_{\text{DBP}}$
(Permutation π)
$[\![\pi(\mathbf{D'})]\!]^{\mathcal{A}}$
$[\![\pi(\mathbf{D'})]\!]^{\mathcal{B}}$
$[\![\mathbf{Y}]\!]^{\mathcal{A}} = [\![\pi(\mathbf{D'})]\!]^{\mathcal{A}}$
$[\![\mathbf{Y}]\!]^{\mathcal{B}} = [\![\pi(\mathbf{D'})]\!]^{\mathcal{A}}$
Phase 2: Finding the leftmost non-zero entry in $[\![\mathbf{Y}]\!]$
2 : Compute $[\![\mathbf{U}]\!]^{\mathcal{A}}$ s.t. $[\![u_i]\!]^{\mathcal{A}} = \sum_{j=1}^{i} [\![y_j]\!]^{\mathcal{A}}$
Compute $[\![\mathbf{U}]\!]^{\mathcal{B}}$ s.t. $[\![u_i]\!]^{\mathcal{B}} = \sum_{j=1}^{i} [\![y_j]\!]^{\mathcal{B}}$
$[\![\mathbf{U}]\!]^{\mathcal{A}}$
$[\![\mathbf{U}]\!]^{\mathcal{A}}$
Compute Vector $\mathbf{V}$
(Using Equation 5.16)
$[\![\mathbf{V}]\!]^{\mathcal{A}}$
$[\![\mathbf{V}]\!]^{\mathcal{B}}$
3 : $[\![t]\!]^{\mathcal{A}} = \sum_{i=1}^{\kappa} [\![v_i]\!]^{\mathcal{A}}$
$[\![t]\!]^{\mathcal{B}} = \sum_{i=1}^{\kappa} [\![v_i]\!]^{\mathcal{B}}$
$[\![t]\!]^{\mathcal{A}}$
$[\![t]\!]^{\mathcal{B}}$
4 : **return** t
return t

Protocol 5.3. Secure realization of $(t, t) = \mathcal{F}_{\text{PA1}}([\![\mathbf{D}]\!]^{\mathcal{A}}, [\![\mathbf{D}]\!]^{\mathcal{B}})$

Lemma 5.7 (Correctness of Protocol 5.3)**.** *Let* $[\![\mathbf{D}]\!]$ *be a length-*κ *vector of 0s and 1s. On Alice's and Bob's respective inputs* $[\![\mathbf{D}]\!]^{\mathcal{A}}$ *and* $[\![\mathbf{D}]\!]^{\mathcal{B}}$*, protocol* PickAnyOne *correctly computes the functionality* $(t,t) = \mathcal{F}_{\mathrm{PA1}}([\![\mathbf{D}]\!]^{\mathcal{A}}, [\![\mathbf{D}]\!]^{\mathcal{B}})$*.*

Proof. The proof is straightforward, but is given nevertheless for the sake of completeness. Clearly, if $[\![\mathbf{D}]\!] = \vec{0}$, then $[\![\mathbf{D}']\!] = [\![\mathbf{Y}]\!] = [\![\mathbf{U}]\!] = \vec{0}$, which by construction results in $t = 0$. If $[\![\mathbf{D}]\!]$ has at least one non-zero entry, Phase 1 of Protocol 5.3 first computes $[\![\mathbf{D}']\!]$ such that $[\![d'_i]\!] = i$ if and only if $[\![d_i]\!] = 1$ (otherwise, $[\![d'_i]\!] = 0$). Then, it uses a random double-blind permutation π to shuffle $[\![\mathbf{D}']\!]$ resulting in $[\![\mathbf{Y}]\!]$; the uniform choice of π implies that all non-zero values in $[\![\mathbf{D}']\!]$ are equiprobable to become the leftmost non-zero entry in $[\![\mathbf{Y}]\!]$. Then, Phase 2 of the protocol finds the leftmost non-zero entry in $[\![\mathbf{Y}]\!]$ as described earlier. □

Lemma 5.8 (Security of Protocol 5.3)**.** *Protocol* PickAnyOne *is secure with respect to* $[\![\mathbf{D}]\!]$ *in the semi-honest threat model. Moreover, given a black-box access to the functionalities* $\mathcal{F}_{\mathrm{EQZ}}$ *and* $\mathcal{F}_{\mathrm{DBP}}$*, the protocol is unconditionally secure in the ABB model of computation.*

Proof. The security of Protocol 5.3 follows immediately from the fact that its only joint computations are (i) the use of $\mathcal{F}_{\mathrm{DBP}}$ in Phase 1, and applying Equation 5.16 in Phase 2, which uses only secure multiplication and functionality $\mathcal{F}_{\mathrm{EQZ}}$. □

5.7.3 FindDistributor: Putting Pieces Together to Find a Valid Distributor

Protocol 5.4, FindDistributor, takes sets $S^{\mathcal{A}}$, $S^{\mathcal{B}}$, as well as integer parameters τ, κ and γ as input; it finds an ODPRF_2^2 that is a valid distributor for $S = S^{\mathcal{A}} \cup S^{\mathcal{B}}$. The protocol consists of two phases:

1. **Randomly generate candidates and verify local validity**: Alice and Bob generate ODPRF_2^2s $\Psi_1, \ldots, \Psi_\kappa$ uniformly and independently at random (Ψ_i abbreviates Ψ_{k_i}). After computing all private bins for all candidate functions, each party locally verifies whether or not each candidate has a local maximum load of at most γ.

2. **Pick a valid candidate uniformly at random**: Alice and Bob compute a length-κ indicator vector $[\![\mathbf{Z}]\!]$ such that $[\![z_i]\!] = 1$ if and only if Ψ_i is a valid distributor. Then,

Alice and Bob invoke protocol PickAnyOne from Section 5.7.2 on input vector $[\![\mathbf{Z}]\!]$ to securely find an index t s.t. $[\![z_t]\!] = 1$. Finally, Alice and Bob keep only item-hash pairs corresponding to Ψ_t and dispose of all other pairs. Formally, Alice's and Bob's private outputs are

$$\begin{aligned} \psi_t^{\mathcal{A}} &= \{(x, \Psi_t(x)) \mid x \in S^{\mathcal{A}}\} \\ \psi_t^{\mathcal{B}} &= \{(x, \Psi_t(x)) \mid x \in S^{\mathcal{B}}\} \end{aligned} \tag{5.17}$$

Complexity. Protocol 5.4, FindDistributor, requires $\mathcal{O}(\kappa m)$ computation and communication in $\mathcal{O}(1)$ rounds (assuming that $\text{Prot}_F^{\text{OPRF}}$ requires $\mathcal{O}(1)$ rounds).

Lemma 5.9 (Correctness of Protocol 5.4)**.** *Protocol FindDistributor on private inputs $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$, and public parameters τ, κ and γ*

1. *finds a valid distributor Ψ_t for $S = S^{\mathcal{A}} \cup S^{\mathcal{B}}$ w.p. at least $1 - m^{-\kappa}$.*

2. *given $\psi_t^{\mathcal{A}}$ (resp. $\psi_t^{\mathcal{B}}$), Alice (resp. Bob) learns nothing about $S^{\mathcal{B}}$ (resp. $S^{\mathcal{A}}$).*

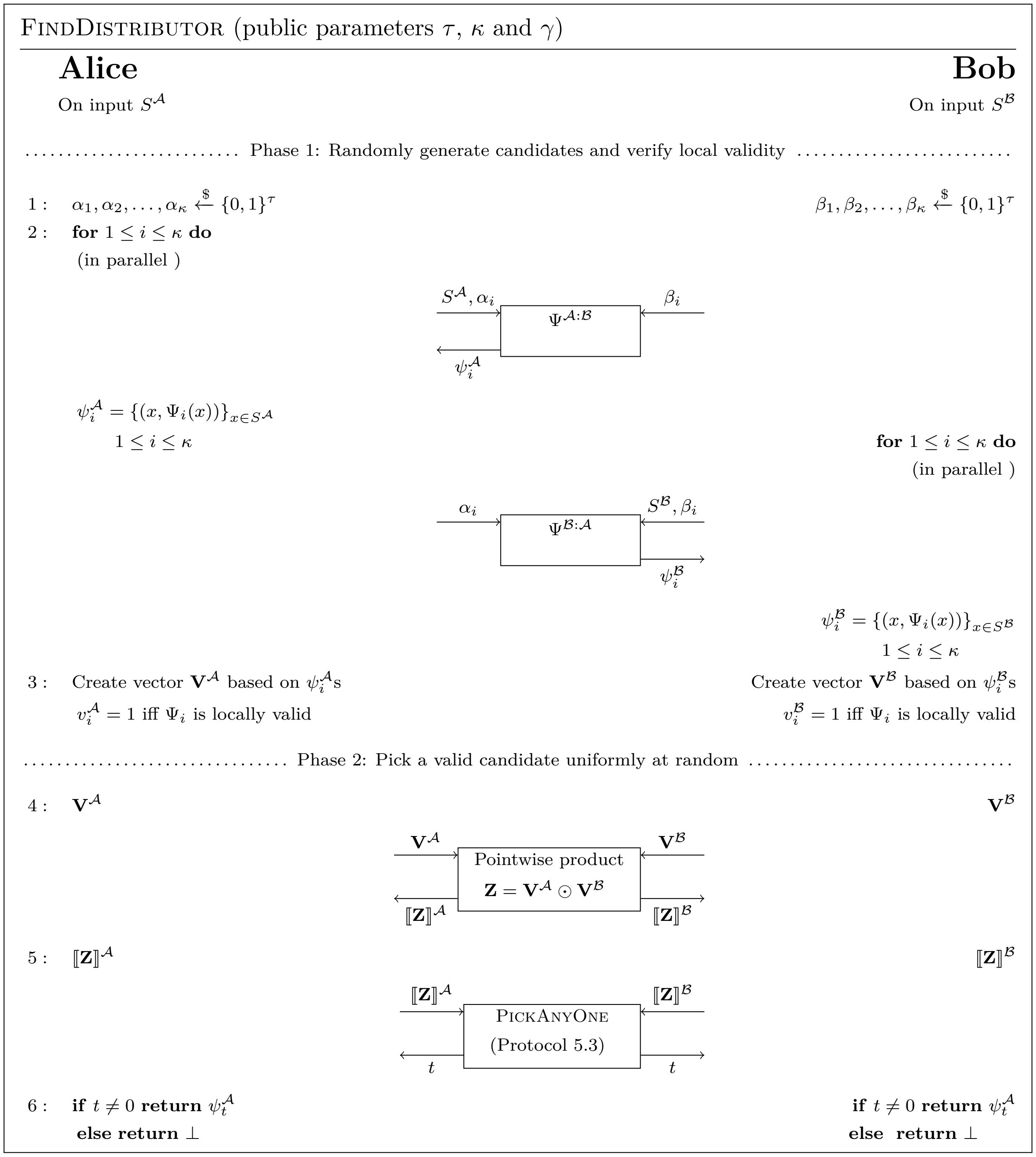
Proof. Claim (1): Since $S^{\mathcal{A}}, S^{\mathcal{B}} \subseteq S$, it suffices to prove that the protocol finds a function Ψ_t that gives a maximum load of at most γ for S; i.e.,

$$\Pr[\ \exists t \text{ s.t. for } \Psi_t \max_{1 \le i < m} (|S_i|) \le \gamma] \ge 1 - m^{-\kappa} \tag{5.18}$$

Since all candidates are PRFs, we prove Equation 5.18 assuming that any candidate Ψ_i distributes $|S| \le m$ items into the m bins uniformly and independently at random (otherwise, Ψ_i would not be indistinguishable from a truly random function) [69]. As discussed earlier, balls and bins analysis [107] implies

$$p_{\mathsf{fail}} = \Pr[\max_{1 \le i < m} (|S_i|) > \gamma] \le m \cdot (e/\gamma)^{\gamma},$$

which can be further simplified as $m \cdot (e/\gamma)^{\gamma} \le \frac{1}{m}$ for $\gamma \ge \frac{3 \ln m}{\ln \ln m}$ and large enough m [107]; also, the κ candidates are chosen independently. Hence, at least one of the candidates is a distributor for S with probability at least $1 - m^{-\kappa}$. This completes proof of Claim (i). Claim (ii) follows directly from the fact that Ψ_t is an ODPRF_2^2: From a PPT point of view



```
FindDistributor (public parameters $\tau$, $\kappa$ and $\gamma$)
Alice                                                        Bob
On input $S^{\mathcal{A}}$                                   On input $S^{\mathcal{B}}$

........ Phase 1: Randomly generate candidates and verify local validity ........

1 :  $\alpha_1, \alpha_2, \ldots, \alpha_\kappa \xleftarrow{\$} \{0,1\}^\tau$          $\beta_1, \beta_2, \ldots, \beta_\kappa \xleftarrow{\$} \{0,1\}^\tau$
2 :  for $1 \le i \le \kappa$ do
       (in parallel )
                 $S^{\mathcal{A}}, \alpha_i$ → [ $\Psi^{\mathcal{A}:\mathcal{B}}$ ] ← $\beta_i$
                 $\psi_i^{\mathcal{A}}$ ←
       $\psi_i^{\mathcal{A}} = \{(x, \Psi_i(x))\}_{x \in S^{\mathcal{A}}}$
          $1 \le i \le \kappa$
                                                             for $1 \le i \le \kappa$ do
                                                               (in parallel )
                 $\alpha_i$ → [ $\Psi^{\mathcal{B}:\mathcal{A}}$ ] ← $S^{\mathcal{B}}, \beta_i$
                                → $\psi_i^{\mathcal{B}}$
                                                             $\psi_i^{\mathcal{B}} = \{(x, \Psi_i(x))\}_{x \in S^{\mathcal{B}}}$
                                                                $1 \le i \le \kappa$
3 :  Create vector $\mathbf{V}^{\mathcal{A}}$ based on $\psi_i^{\mathcal{A}}$s          Create vector $\mathbf{V}^{\mathcal{B}}$ based on $\psi_i^{\mathcal{B}}$s
     $v_i^{\mathcal{A}} = 1$ iff $\Psi_i$ is locally valid             $v_i^{\mathcal{B}} = 1$ iff $\Psi_i$ is locally valid

........ Phase 2: Pick a valid candidate uniformly at random ........

4 :  $\mathbf{V}^{\mathcal{A}}$                                  $\mathbf{V}^{\mathcal{B}}$
                 $\mathbf{V}^{\mathcal{A}}$ → [ Pointwise product                  ] ← $\mathbf{V}^{\mathcal{B}}$
                 $[\![\mathbf{Z}]\!]^{\mathcal{A}}$ ← [ $\mathbf{Z} = \mathbf{V}^{\mathcal{A}} \odot \mathbf{V}^{\mathcal{B}}$ ] → $[\![\mathbf{Z}]\!]^{\mathcal{B}}$
5 :  $[\![\mathbf{Z}]\!]^{\mathcal{A}}$                          $[\![\mathbf{Z}]\!]^{\mathcal{B}}$
                 $[\![\mathbf{Z}]\!]^{\mathcal{A}}$ → [ PickAnyOne     ] ← $[\![\mathbf{Z}]\!]^{\mathcal{B}}$
                 $t$ ← [ (Protocol 5.3) ] → $t$
6 :  if $t \neq 0$ return $\psi_t^{\mathcal{A}}$                if $t \neq 0$ return $\psi_t^{\mathcal{A}}$
     else return $\perp$                                        else return $\perp$
```

Protocol 5.4. Finding a valid distributor function. Note that Step 1 is equivalent to κ invocations of Gen(1^τ) from Protocol 5.2.

Ψ_t distributes $|S|$ items uniformly and independently; also, each party learns only the image of their own private set under Ψ_t and does not know the other party's secret key for the candidate. A formal proof follows an argument similar to the proof of Lemma 5.1, and is straightforward (hence omitted). □

Lemma 5.10 (Security of Protocol 5.4)**.** *Protocol* FindDistributor *is secure with respect to sets $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$ in the semi-honest threat model.*

Proof. Security of Phase 1 in the protocol follows from the already argued security of protocols $\Psi^{\mathcal{A}:\mathcal{B}}$ and $\Psi^{\mathcal{B}:\mathcal{A}}$ (see Lemma 5.6); note that for $1 \leq i \leq \kappa$ each party learns only the image of their own private set under the candidate Ψ_i and does not know the other party's secret key for Ψ_i. Security of Phase 2 follows from the fact that it uses only secure multiplication in split form (Step 4) and protocol PickAnyOne (Step 5) which is itself secure due to Lemma 5.8. □

5.8 Overall Distribution-Resolution Scheme

We explain how Distribution-Resolution scheme (Figure 5.9) combines the schemes of sections 5.6 and 5.7 to construct an SPHF. We also discuss the inherent trade-off between parameters γ and κ. Finally, Theorem 5.8.1 states the correctness and security of our Distribution-Resolution scheme.

Let Ψ be the distributor that partitions S into m subproblems of size at most 2γ each. Moreover, let $\rho_0, \rho_1, \ldots, \rho_{m-1}$ be the resolvers obtained by FindSPHF for each subproblem. As depicted in Figure 5.9, SPHF $\rho = \langle \Psi, \rho_0, \rho_1, \ldots, \rho_{m-1} \rangle$ works as follows: There are 2γ hash values reserved for each subproblem S_i. Any $x \in S$ with $\Psi(x) = i$ resides in S_i and has, within this subproblem, an offset of $\rho_i(x) < 2\gamma$. Thus, the desired hash value $\rho(x)$ is the sum of two terms: The first term is $2\gamma \cdot i$ which corresponds to the hash values reserved for subproblems $S_0, \cdots, S_{i-1}$; the second term is $\rho_i(x)$, and represents the offset assigned to x among all 2γ reserved hash values for S_i. In summary,

$$\rho(x) = 2\gamma \cdot i + \rho_i(x), \text{ for } i = \Psi(x) \tag{5.19}$$

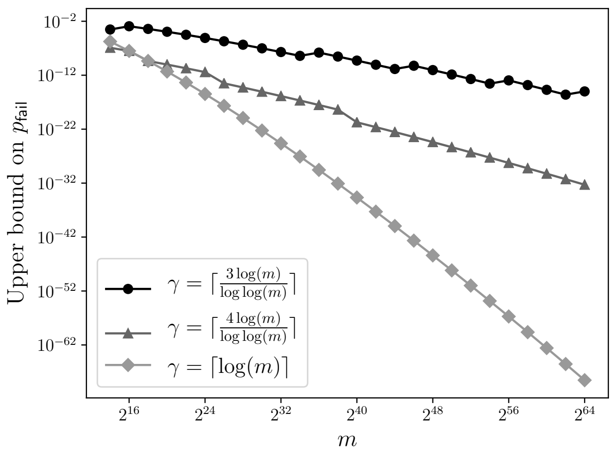


Figure 5.8. Upper bound $p_{\mathsf{fail}} \leq m \cdot (e/\gamma)^\gamma$ for various choices of γ as a function of m

The codomain cardinality of ρ is linear in γ, and naturally a larger γ results in a smaller p_{fail} for each randomly chosen distributor candidate. Figure 5.8 illustrates p_{fail} for three choices of γ as a function of m. Note that for all choices of γ, p_{fail} decreases as m grows; this suggests a trade-off between parameters γ and κ. For example, if $m = 2^{24}$ and it is desired to find a valid distributor with probability at least $1 - 10^{-50}$ (i.e., $(p_{\mathsf{fail}})^\kappa \leq 10^{-50}$), Alice and Bob may choose $\gamma = \lceil \log m \rceil = 24$ and $\kappa = 4$, or $\gamma = \lceil \frac{3 \log m}{\log \log m} \rceil = 16$ and $\kappa = 10$.

Theorem 5.8.1 (Correctness and Security)**.** *Let $\rho = \langle \Psi, \rho_0, \rho_1, \ldots, \rho_{m-1} \rangle$ be a function obtained by the Distribution-Resolution scheme on private inputs $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$:*

1. *ρ is an SPHF for $S = S^{\mathcal{A}} \cup S^{\mathcal{B}}$.*

2. *Construction of ρ is secure with respect to the private sets $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$ in the semi-honest threat model.*

Proof. For claim (1), we show that ρ satisfies Definition 5.1. Function ρ is injective on S by construction: First, S is partitioned into m subproblems of size at most 2γ; then, each subproblem is solved using FindSPHF and obtained SPHFs are combined using Equation 5.19. It remains to prove the security property in Definition 5.1; since equations 5.1 and 5.2 are symmetric, we prove only Equation 5.1, i.e., for $\varrho^{\mathcal{A}} = \{(x, \rho(x))\}_{x \in S^{\mathcal{A}}}$

$$|\Pr[y \in S^{\mathcal{B}} \mid \varrho^{\mathcal{A}}] - \Pr[y \in S^{\mathcal{B}}]| \leq \mathsf{negl}(\tau)$$

The resolvers $\rho_0, \rho_1, \ldots, \rho_{m-1}$ are *perfect SPHFs*, i.e., $\Pr[y \in S^{\mathcal{B}} | \varrho_i^{\mathcal{A}}] = \Pr[y \in S^{\mathcal{B}}]$ for all $0 \leq i \leq m-1$. This, together with the structure of $\rho(x)$ as in Equation 5.19, implies $\Pr[y \in S^{\mathcal{A}} | \varrho^{\mathcal{A}}] = \Pr[y \in S^{\mathcal{B}} | \psi^{\mathcal{A}}]$ (recall that $\psi^{\mathcal{A}} = \{(x, \Psi(x))\}_{x \in S^{\mathcal{A}}}$). In other words, any (hypothetical) difference between $\Pr[y \in S^{\mathcal{B}} | \varrho^{\mathcal{A}}]$ and $\Pr[y \in S^{\mathcal{B}}]$ must be caused by the distributor Ψ. Thus, it suffices to argue that such a difference is negligible:

$$|\Pr[y \in S^{\mathcal{B}} \mid \psi^{\mathcal{A}}] - \Pr[y \in S^{\mathcal{B}}]| \leq \mathsf{negl}(\tau) \tag{5.20}$$

Recall that Ψ is an ODPRF$_2^2$ and is indistinguishable from a truly random function except w.p. $\mathsf{negl}(\tau)$. If Equation 5.20 did not hold, then a PPT distinguisher could distinguish Ψ from a truly random function, which would contradict the fact that Ψ is a PRF.

For claim (2), recall that all m subproblems S_i are mutually disjoint and are dealt with separately via independent invocations of secure protocols proposed in Section 5.6 for the

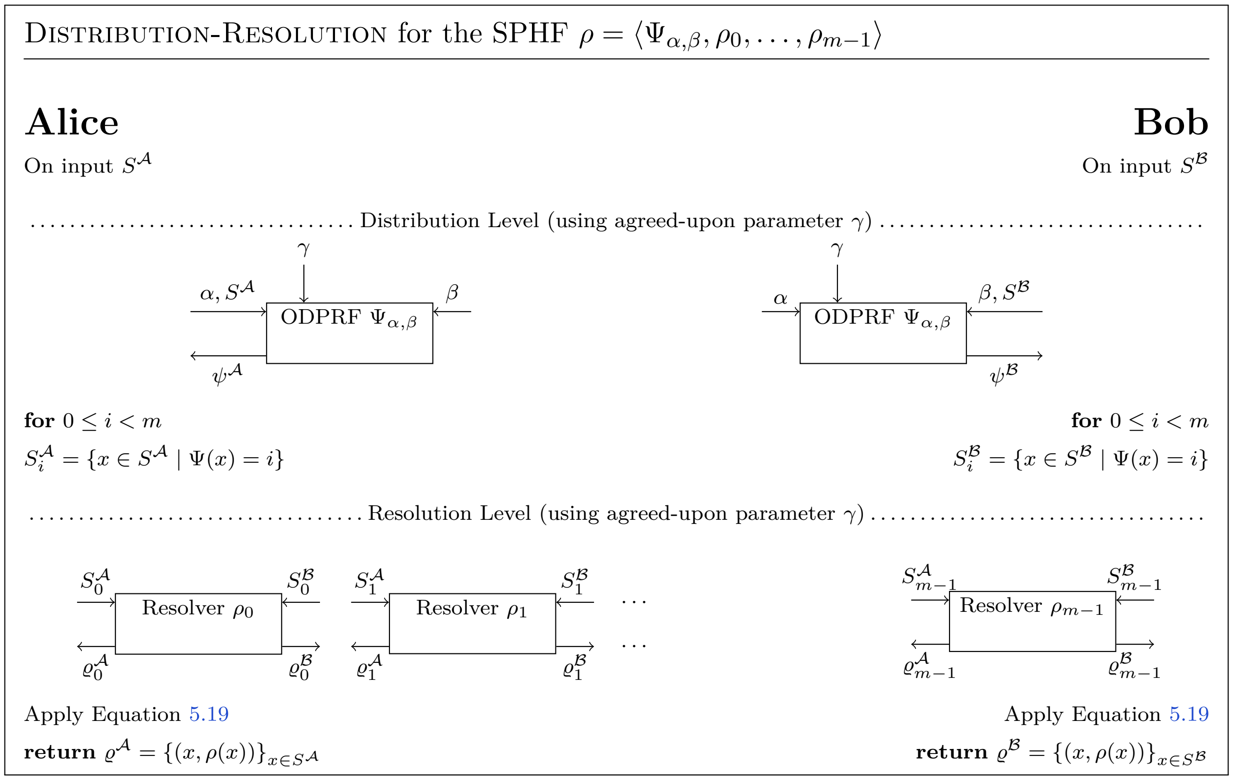

DISTRIBUTION-RESOLUTION for the SPHF $\rho = \langle \Psi_{\alpha,\beta}, \rho_0, \ldots, \rho_{m-1} \rangle$
Alice
On input $S^{\mathcal{A}}$
Bob
On input $S^{\mathcal{B}}$
Distribution Level (using agreed-upon parameter γ)
γ
$\alpha, S^{\mathcal{A}}$
ODPRF $\Psi_{\alpha,\beta}$
β
$\psi^{\mathcal{A}}$
γ
α
ODPRF $\Psi_{\alpha,\beta}$
$\beta, S^{\mathcal{B}}$
$\psi^{\mathcal{B}}$
for $0 \leq i < m$
$S_i^{\mathcal{A}} = \{x \in S^{\mathcal{A}} \mid \Psi(x) = i\}$
for $0 \leq i < m$
$S_i^{\mathcal{B}} = \{x \in S^{\mathcal{B}} \mid \Psi(x) = i\}$
Resolution Level (using agreed-upon parameter γ)
$S_0^{\mathcal{A}}$
Resolver ρ_0
$S_0^{\mathcal{B}}$
$\varrho_0^{\mathcal{A}}$
$\varrho_0^{\mathcal{B}}$
$S_1^{\mathcal{A}}$
Resolver ρ_1
$S_1^{\mathcal{B}}$
$\varrho_1^{\mathcal{A}}$
$\varrho_1^{\mathcal{B}}$
$\cdots$
$S_{m-1}^{\mathcal{A}}$
Resolver ρ_{m-1}
$S_{m-1}^{\mathcal{B}}$
$\varrho_{m-1}^{\mathcal{A}}$
$\varrho_{m-1}^{\mathcal{B}}$
Apply Equation 5.19
return $\varrho^{\mathcal{A}} = \{(x, \rho(x))\}_{x \in S^{\mathcal{A}}}$
Apply Equation 5.19
return $\varrho^{\mathcal{B}} = \{(x, \rho(x))\}_{x \in S^{\mathcal{B}}}$

Figure 5.9. Overview of the Distribution-Resolution approach for the SPHF construction. Black-box reductions in the same level denote independent secure computations in parallel.

FindSPHF approach. Hence, the overall security reduces to the security of the Distribution level (Lemma 5.10) that was already discussed in Section 5.7. □

5.9 Discussion: Use Cases and Implications in 2PC

Below, we describe two MPC applications for which input-size reduction and/or using SPHFs result in significant performance improvements. We also discuss the implicit connection of secure perfect hashing with the important and well-studied problem of Private Set Intersection (PSI).

5.9.1 Faster General Template Matching

Alice has an image $\mathbf{X} \in \Sigma^{n_1 \times n_1}$ and Bob owns a template $\mathbf{T} \in \Sigma^{n_2 \times n_2}$, where $n_2 < n_1$. They want to securely measure the similarity of $\mathbf{X}_{i,j} = \mathbf{X}[i \cdots i + n_2 - 1, j \cdots j + n_2 - 1]$ and $\mathbf{T}$ for all $1 \leq i, j \leq n_1 - n_2 + 1$. The problem has been extensively studied for various notions of similarity [87], [116]–[118]. The performance of many solutions to this problem depends on $|\Sigma|$, including those based on secure convolution or oblivious automata evaluation. We explain how to use SPHFs for faster secure Hamming similarity computation between $\mathbf{X}_{i,j}$ and $\mathbf{T}$ for all i and j. As illustrated in Figure 5.10, Alice and Bob intend to compute a *score matrix* $[\![\mathbf{C}]\!]$ containing the Hamming similarity of $\mathbf{X}_{i,j}$ and $\mathbf{T}$ such that

$$[\![c_{i,j}]\!] = \sum_{k=1}^{n_2} \sum_{l=1}^{n_2} (x_{i+k-1,j+l-1} == t_{k,l})$$

A convolution-based solution [116] for computing $[\![\mathbf{C}]\!]$ consists of one secure convolution in split form [41] per alphabet symbol $z \in \Sigma$ such that any occurrence of z in $\mathbf{X}$ and $\mathbf{T}$ is replaced with a bit 1 and all other symbols are replaced with 0. Each convolution computes a *partial* score matrix $[\![\mathbf{C}_z]\!]$ accounting for the contribution of z to $[\![\mathbf{C}]\!]$ with the overall contribution being $[\![\mathbf{C}]\!] = \sum_z [\![\mathbf{C}_z]\!]$. Computing $[\![\mathbf{C}]\!]$ takes $\mathcal{O}(|\Sigma| n_1^2 \log n_2)$ computation and communication ($\mathcal{O}(n_1^2 \log n_2)$ for each convolution) in $\mathcal{O}(1)$ rounds; the use of our approach essentially replaces the $|\Sigma|$ in the complexity with the size of a much smaller alphabet domain.

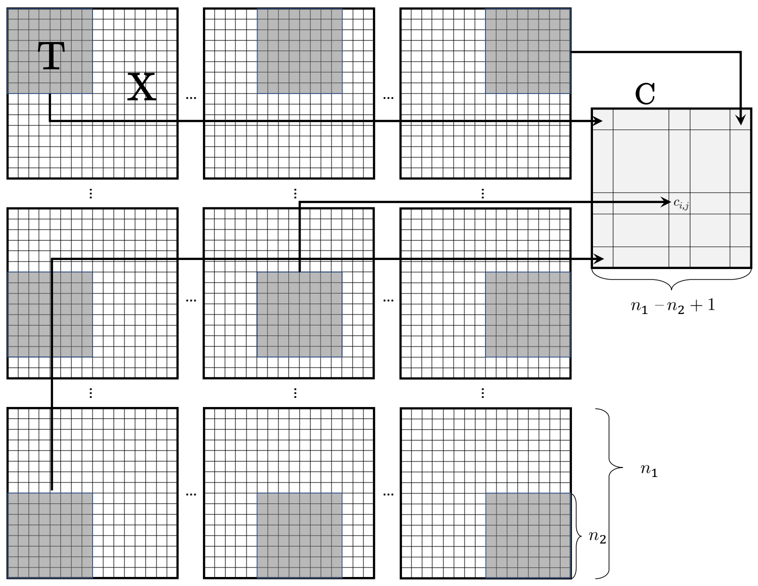


Figure 5.10. Hamming similarity of $\mathbf{X}_{i,j}$ and $\mathbf{T}$

For instance, let $\mathbf{X}$ and $\mathbf{T}$ be images with RGB encoding (i.e., $|\Sigma| = 2^{24}$), with $n_1 = 2^{10}$ and $n_2 = 2^6$ corresponding to a $2^{10} \times 2^{10}$ image and a $2^6 \times 2^6$ template. Unlike any non-secure setting, in the MPC framework, Alice and Bob cannot reveal to each other which alphabet symbols occur in $\mathbf{X}$ and $\mathbf{T}$ to simply ignore the non-occurring symbols; this forces them to compute $|\Sigma|$ convolutions, most of which pertain to non-occurring symbols. There are at most $n_1^2 + n_2^2 < 2^{21}$ distinct RGB codes contributing to $[\![\mathbf{C}]\!]$; thus, the above algorithm computes at least $2^{24} - 2^{21} \approx 15 \times 10^6$ redundant convolutions in split form, each of which contributes only 0s to the score matrix $[\![\mathbf{C}]\!]$. Our input-size reduction results in securely avoiding such redundant costs: Let $S^{\mathcal{A}}$ (resp. $S^{\mathcal{B}}$) be the set of occurring symbols in $\mathbf{X}$ (resp. $\mathbf{T}$). Alice and Bob (i) find an SPHF $\rho : \Sigma \to \{0, 1, \cdots, N-1\}$ for $S = S^{\mathcal{A}} \cup S^{\mathcal{B}}$, where $N = |S^{\mathcal{A}}| + |S^{\mathcal{B}}|$, (ii) replace each symbol $z \in \Sigma$ that appears in $\mathbf{X}$ and $\mathbf{T}$ with $\rho(z)$, and (iii) compute Hamming similarities using the reduced alphabet $\{0, 1, \cdots, N-1\}$ rather than the much larger Σ.

5.9.2 Search Queries in High-precision Scientific Data

High-precision scientific data typically has at least 64 bits of precision, therefore $|\Sigma| \geq 2^{64}$. Even though searches in such data may not be exact but approximate, e.g., within 12 bits based on a quantization that reduces alphabet size from 2^{64} to 2^{52} (i.e., compressing a range of values to a single value), the alphabet is still too large. Such a large alphabet results in a huge overhead for search queries, specifically in the MPC framework where the structure of items and how they relate to each other (e.g., their order) cannot be exploited for faster search queries. In these scenarios, SPHFs can be used to securely create a dictionary of items for better performance (e.g., faster search queries and saving memory space).

5.9.3 Implications of Secure Perfect Hashing for Private Set Intersection

Private Set Intersection (PSI) is one of the most well-studied MPC problems [31], [119]–[122]. PSI allows Alice and Bob to obtain the intersection of their private sets without revealing any information about items not in the intersection. Although naive hashing based solutions to PSI are very efficient, they are insecure if the input domain does not have high entropy or is not large [119]. Below, we discuss how our input-size reduction inherently solves PSI (whereas the input-size reduction of [31] must be followed by parallel applications of a circuit- or OT- based PSI protocol).

Let $\rho : \Sigma \rightarrow \{0, 1, \ldots, N-1\}$ be an SPHF for $S^{\mathcal{A}} \cup S^{\mathcal{B}}$, where $S^{\mathcal{A}}, S^{\mathcal{B}} \subseteq \Sigma$ are respectively Alice's and Bob's private input sets. Alice and Bob use ρ to obtain an indicator vector $[\![\mathbf{P}]\!]$ for membership in $S^{\mathcal{A}} \cap S^{\mathcal{B}}$:

1. Alice creates a length-N indicator vector $\mathbf{Z}^{\mathcal{A}}$ for membership in $S^{\mathcal{A}}$ such that for $1 \leq i \leq N$:
$$z_i^{\mathcal{A}} = \begin{cases} 1 & \text{if } \exists x \in S^{\mathcal{A}} \text{ s.t. } i = 1 + \rho(x) \\ 0 & \text{otherwise} \end{cases} \tag{5.21}$$
Similarly, Bob creates a length-N indicator vector $\mathbf{Z}^{\mathcal{B}}$ for membership in $S^{\mathcal{B}}$ such that for $1 \leq i \leq N$:
$$z_i^{\mathcal{B}} = \begin{cases} 1 & \text{if } \exists x \in S^{\mathcal{B}} \text{ s.t. } i = 1 + \rho(x) \\ 0 & \text{otherwise} \end{cases} \tag{5.22}$$
Equations 5.21 and 5.22 use "$i = \rho(x) + 1$" rather than "$i = \rho(x)$" becasuse in our notation all arrays are 1-indexed, while hash values are in the range $\{0, 1, \cdots, N-1\}$.

2. Alice and Bob compute the joint indicator vector $[\![\mathbf{P}]\!] = [\![\mathbf{Z}^{\mathcal{A}} \odot \mathbf{Z}^{\mathcal{B}}]\!]$ for membership in $S^{\mathcal{A}} \cap S^{\mathcal{B}}$.

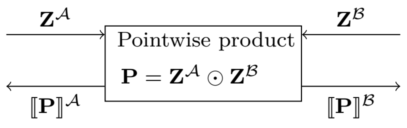
3. At this point, Alice and Bob may, depending on what they are trying to achieve, (i) learn $S^{\mathcal{A}} \cap S^{\mathcal{B}}$ through revealing $\mathbf{P}$ to each other by exchanging their shares of it, (ii) learn the intersection cardinality $|S^{\mathcal{A}} \cap S^{\mathcal{B}}| = \sum_{i=1}^{N} p_i$ without revealing the intersection, or (iii) proceed with any other subsequent computation of interest on $[\![\mathbf{P}]\!]$ without revealing anything about the intersection.

PRIVATE-SET-INTERSECTION (Public parameter: N)

Alice | **Bob**

On input $\varrho^{\mathcal{A}} = \{(x, \rho(x))\}_{x \in S^{\mathcal{A}}}$ | On input $\varrho^{\mathcal{B}} = \{(x, \rho(x))\}_{x \in S^{\mathcal{B}}}$

1 : Create $\mathbf{Z}^{\mathcal{A}}$ as in Equation 5.21 | Create $\mathbf{Z}^{\mathcal{B}}$ as in Equation 5.22


$\mathbf{Z}^{\mathcal{A}}$
$\mathbf{Z}^{\mathcal{B}}$
Pointwise product
$\mathbf{P} = \mathbf{Z}^{\mathcal{A}} \odot \mathbf{Z}^{\mathcal{B}}$
$[\![\mathbf{P}]\!]^{\mathcal{A}}$
$[\![\mathbf{P}]\!]^{\mathcal{B}}$

2 : **return** $[\![\mathbf{P}]\!]^{\mathcal{A}}$ | **return** $[\![\mathbf{P}]\!]^{\mathcal{B}}$

Protocol 5.5. Private Set Intersection built on Secure Perfect Hashing

This PSI solution is as secure as the underlying SPHF because the only joint computation in Protocol 5.5 consists of N secure multiplication in split form.

Lemma 5.11 (Correctness and Security of Protocol 5.5)**.** *Let $\rho : \Sigma \to \{0, 1, \cdots, N-1\}$ be an SPHF for $S = S^{\mathcal{A}} \cup S^{\mathcal{B}}$. Protocol* PRIVATE-SET-INTERSECTION*, on Alice's and Bob's respective private inputs $\varrho^{\mathcal{A}} = \{(x, \rho(x)) \mid x \in S^{\mathcal{A}}\}$ and $\varrho^{\mathcal{B}} = \{(x, \rho(x)) \mid x \in S^{\mathcal{B}}\}$, securely computes the intersection of $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$ in split form. Moreover, the protocol is unconditionally secure in the ABB model against a semi-honest adversary.*

Proof. Correctness of the protocol follows from the injectivity of ρ on $S^{\mathcal{A}} \cup S^{\mathcal{B}}$ together with the construction of private indicator vectors $\mathbf{Z}^{\mathcal{A}}$ and $\mathbf{Z}^{\mathcal{B}}$ as in equations 5.21 and 5.22:

$$\begin{gathered} z_i^{\mathcal{A}} = 1 \Leftrightarrow\ \exists x \in S^{\mathcal{A}} \text{ s.t. } i = 1 + \rho(x) \quad \wedge \quad z_i^{\mathcal{B}} = 1 \Leftrightarrow\ \exists x \in S^{\mathcal{B}} \text{ s.t. } i = 1 + \rho(x) \\ \Big\Downarrow \\ p_i = z_i^{\mathcal{A}} \cdot z_i^{\mathcal{B}} = 1 \Leftrightarrow \exists x \in S^{\mathcal{A}} \cap S^{\mathcal{B}} \text{ s.t. } i = 1 + \rho(x) \end{gathered}$$

Security of Protocol 5.5 follows from the fact that the only step with joint computations involves N invocations of secure multiplication on (distinct) pairs of inputs, with outputs in split form; hence, each party receives only a uniform vector over $\mathbb{F}_q^N$ that, in the absence of the other party's output, is statistically independent of $\mathbf{P}$. $\square$

Complexity. Protocol 4.8 requires one round of communication; moreover, its computation and communication complexities are both $\mathcal{O}(N) = \mathcal{O}(|S^{\mathcal{A}} \cup S^{\mathcal{B}}|)$.

6. SUMMARY

In this dissertation, we have presented approaches and techniques for improving the performance of secure two-party computation of functionalities that depend on equality of comparands. In pursuit of this goal, we considered performance improvement from two distinct aspects. First, we proposed lightweight and provably secure protocols for computing two fundamental problems of interest: *Private equality testing* and *secure wildcard pattern matching*. Both of these problems are of vital importance in secure two-party computation; they have many applications and are often used as basic building blocks for secure computation of other functionalities. Our second point of view concentrates on lowering the impact of inputs' sizes on any subsequent protocol's overall cost for computing a functionality of interest. Accordingly, we presented the first formal attempt at formulating and solving the problem of preprocessing for input-size reduction in a secure two-party setting.

Private Equality Testing

Chapter 3 of this thesis investigated the boolean problem of private equality testing (the socialist millionaire problem) over the finite field $\mathbb{F}_q$ of integers modulo a prime q. This functionality allows two parties to securely check the equality of their respective secret integers $a \in \mathbb{F}_q$ and $b \in \mathbb{F}_q$ without revealing any other information about these secrets. We presented a protocol that computes the value of the desired predicate in split form; hence neither party learns the outcome (unless they agree to do so), but they can use it for further computations. Moreover, our solution is lightweight in the sense that all internal computations are additions and multiplications over the small finite field $\mathbb{F}_3$ and only three multiplications over $\mathbb{F}_q$ are used in the final step of the protocol to convert the obtained test result in $\mathbb{F}_3$ to its counterpart in $\mathbb{F}_q$.

Secure Wildcard Pattern Matching

In Chapter 4, we proposed a template algorithm to solve the wildcard pattern matching problem (over any finite alphabet), using a single convolution computation. Its sub-quadratic complexity, together with the linearity of convolution, make this scheme an appropriate tool for use in the secure two-party framework. We also presented protocols for the secure implementation of our algorithm. Accordingly, we gave two sets of protocols for securely computing all three versions (search, counting, and decision) of secure two-party wildcard pattern matching. All of our protocols require sub-quadratic computation complexity and linear communication complexity in a constant number of rounds. Moreover, our protocols use only lightweight operations and avoid expensive cryptographic primitives such as homomorphic encryption and public-key operations.

In Section 4.7, we went beyond the semi-honest threat model. We pointed out that the conventional search version of the problem has a definitional drawback that would defeat the purpose of using any secure protocol in the presence of a stronger adversary such as *augmented semi-honest* and *malicious*. This definitional drawback allows the pattern owner (in our case, Bob) to learn the other party's input text in its entirety by providing a judiciously crafted pattern string. We proposed a fix for this issue by generalizing the problem to use an output-filtering function that restricts what Bob learns. All three traditional versions of the problem are special cases of our general formulation. Moreover, we proposed protocols for two other versions of secure two-party wildcard pattern matching.

Secure Two-party Input-size Reduction

Chapter 5 of this thesis presented the first formal attempt at formulating and solving the secure input-size reduction problem as a generic preprocessing that aims for more efficient secure two-party equality-based protocols without affecting their outputs. To do so, we formalized the notion of secure perfect hash functions in the two-party framework and proposed efficient constructions that obtain such functions. In addition to solving the input-size reduction, this also brings the advantages of traditional perfect hashing (i.e., less memory space, faster memory access) to the multiparty framework. To overcome the inherent obstacles of efficient secure perfect hashing, we used the notion of Oblivious Distributed Pseudorandom Functions and gave a practical two-party construction for it. Furthermore, we discussed the performance improvements made possible by this approach in image template matching and in searches done on high-precision data. Finally, we also discussed the implications of secure perfect hashing for Private Set Intersection, which is one of the fundamental problems in the secure two-party framework.

REFERENCES

[1] A. C. Yao, "Protocols for secure computations," in *23rd Annual Symposium on Foundations of Computer Science (FOCS 1982)*, IEEE, 1982, pp. 160–164.

[2] W. Du, M. J. Atallah, *et al.*, "Privacy-preserving cooperative scientific computations.," in *CSFW*, Citeseer, vol. 1, 2001, p. 273.

[3] W. Du and M. J. Atallah, "Privacy-preserving cooperative statistical analysis," in *Seventeenth Annual Computer Security Applications Conference*, IEEE, 2001, pp. 102–110.

[4] W. Du and M. J. Atallah, "Secure multi-party computation problems and their applications: A review and open problems," in *Proceedings of the 2001 Workshop on New Security Paradigms*, ACM, 2001, pp. 13–22.

[5] W. Du, Y. S. Han, and S. Chen, "Privacy-preserving multivariate statistical analysis: Linear regression and classification," in *Proceedings of the 2004 SIAM International Conference on Data Mining*, SIAM, 2004, pp. 222–233.

[6] D. Bogdanov, L. Kamm, S. Laur, P. Pruulmann-Vengerfeldt, R. Talviste, and J. Willemson, "Privacy-preserving statistical data analysis on federated databases," in *Annual Privacy Forum*, Springer, 2014, pp. 30–55.

[7] D. Bogdanov, M. Niitsoo, T. Toft, and J. Willemson, "High-performance secure multiparty computation for data mining applications," *International Journal of Information Security*, vol. 11, no. 6, pp. 403–418, 2012.

[8] W. Priesnitz Filho and C. N. da Cruz Ribeiro, "State of the art of secure multiparty computation for privacy preserving data mining," *Revista GEINTEC-Gestão, Inovação e Tecnologias*, vol. 7, no. 4, pp. 4131–4148, 2017.

[9] F. A. N. Pathak and S. B. S. Pandey, "An efficient method for privacy preserving data mining in secure multiparty computation," in *2013 Nirma University International Conference on Engineering (NUiCONE)*, IEEE, 2013, pp. 1–3.

[10] S. Jha, L. Kruger, and V. Shmatikov, "Towards practical privacy for genomic computation," in *2008 IEEE Symposium on Security and Privacy (SP 2008)*, IEEE, 2008, pp. 216–230.

[11] E. Check Hayden, "Extreme cryptography paves way to personalized medicine," *Nature News*, vol. 519, no. 7544, p. 400, 2015.

[12] H. Cho, D. J. Wu, and B. Berger, "Secure genome-wide association analysis using multiparty computation," *Nature Biotechnology*, vol. 36, no. 6, pp. 547–551, 2018.

[13] B. Hie, H. Cho, and B. Berger, “Realizing private and practical pharmacological collaboration,” *Science*, vol. 362, no. 6412, pp. 347–350, 2018.

[14] K. A. Jagadeesh, D. J. Wu, J. A. Birgmeier, D. Boneh, and G. Bejerano, “Deriving genomic diagnoses without revealing patient genomes,” *Science*, vol. 357, no. 6352, pp. 692–695, 2017.

[15] E. A. Abbe, A. E. Khandani, and A. W. Lo, “Privacy-preserving methods for sharing financial risk exposures,” *American Economic Review*, vol. 102, no. 3, pp. 65–70, 2012.

[16] D. Bogdanov, L. Kamm, B. Kubo, R. Rebane, V. Sokk, and R. Talviste, “Students and taxes: A privacy-preserving study using secure computation,” *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 3, pp. 117–135, 2016.

[17] D. Bogdanov, R. Talviste, and J. Willemson, “Deploying secure multi-party computation for financial data analysis,” in *International Conference on Financial Cryptography and Data Security*, Springer, 2012, pp. 57–64.

[18] M. D. Flood, J. Katz, S. J. Ong, and A. Smith, “Cryptography and the economics of supervisory information: Balancing transparency and confidentiality,” 2013.

[19] A. C.-C. Yao, “How to generate and exchange secrets,” in *27th Annual Symposium on Foundations of Computer Science (FOCS 1986)*, IEEE, 1986, pp. 162–167.

[20] D. Chaum, C. Crépeau, and I. Damgard, “Multiparty unconditionally secure protocols,” in *Proceedings of the twentieth annual ACM Symposium on Theory of Computing*, 1988, pp. 11–19.

[21] S. Micali, O. Goldreich, and A. Wigderson, “How to play any mental game,” in *Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC*, ACM, 1987, pp. 218–229.

[22] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella, *et al.*, “Fairplay-secure two-party computation system.,” in *USENIX Security Symposium*, San Diego, CA, USA, vol. 4, 2004, p. 9.

[23] A. Ben-David, N. Nisan, and B. Pinkas, “Fairplaymp: A system for secure multi-party computation,” in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, 2008, pp. 257–266.

[24] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic, “Sok: General purpose compilers for secure multi-party computation,” in *2019 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2019, pp. 1220–1237.

[25] J. A. Montenegro, M. J. Fischer, J. Lopez, and R. Peralta, “Secure sealed-bid online auctions using discreet cryptographic proofs,” *Mathematical and Computer Modelling*, vol. 57, no. 11-12, pp. 2583–2595, 2013.

[26] L. Dery, T. Tassa, and A. Yanai, “Fear not, vote truthfully: Secure multiparty computation of score based rules,” *Expert Systems with Applications*, vol. 168, p. 114 434, 2021.

[27] M. Zarezadeh, H. Mala, and B. T. Ladani, “Secure parameterized pattern matching,” *Information Sciences*, vol. 522, pp. 299–316, 2020.

[28] C. Hazay and Y. Lindell, *Efficient secure two-party protocols: Techniques and constructions*. Springer Science & Business Media, 2010.

[29] L. Shundong, W. Daoshun, D. Yiqi, and L. Ping, “Symmetric cryptographic solution to yao’s millionaires’ problem and an evaluation of secure multiparty computations,” *Information Sciences*, vol. 178, no. 1, pp. 244–255, 2008.

[30] L. Shundong, W. Chunying, W. Daoshun, and D. Yiqi, “Secure multiparty computation of solid geometric problems and their applications,” *Information Sciences*, vol. 282, pp. 401–413, 2014.

[31] B. Pinkas, T. Schneider, G. Segev, and M. Zohner, “Phasing: Private set intersection using permutation-based hashing,” in *24th USENIX Security Symposium*, 2015, pp. 515–530.

[32] C. Dong and G. Loukides, “Approximating private set union/intersection cardinality with logarithmic complexity,” *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 11, pp. 2792–2806, 2017.

[33] M. von Maltitz and G. Carle, “A performance and resource consumption assessment of secret sharing based secure multiparty computation,” in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, Springer, 2018, pp. 357–372.

[34] C. Zhao, S. Zhao, M. Zhao, Z. Chen, C.-Z. Gao, H. Li, and Y.-a. Tan, “Secure multi-party computation: Theory, practice and applications,” *Information Sciences*, vol. 476, pp. 357–372, 2019.

[35] M. Qaosar, A. Zaman, M. A. Siddique, C. Li, and Y. Morimoto, “Secure k-skyband computation framework in distributed multi-party databases,” *Information Sciences*, vol. 515, pp. 388–403, 2020.

[36] X. Wei, L. Xu, M. Zhao, and H. Wang, “Secure extended wildcard pattern matching protocol from cut-and-choose oblivious transfer,” *Information Sciences*, vol. 529, pp. 132–140, 2020.

[37] C. Zhao, S. Zhao, B. Zhang, S. Jing, Z. Chen, and M. Zhao, "Oblivious dfa evaluation on joint input and its applications," *Information Sciences*, vol. 528, pp. 168–180, 2020.

[38] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams, "Secure two-party computation is practical," in *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, 2009, pp. 250–267.

[39] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft, "Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation," in *Theory of Cryptography Conference*, Springer, 2006, pp. 285–304.

[40] B. Schoenmakers and P. Tuyls, "Efficient binary conversion for paillier encrypted values," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2006, pp. 522–537.

[41] J. Darivandpour and M. J. Atallah, "Efficient and secure pattern matching with wildcards using lightweight cryptography," *Computers & Security*, vol. 77, pp. 666–674, 2018.

[42] J. Darivandpour, D. V. Le, and M. J. Atallah, "Secure two-party input-size reduction: Challenges, solutions and applications," *Information Sciences*, vol. 567, pp. 256–277, 2021.

[43] T. Nishide and K. Ohta, "Multiparty computation for interval, equality, and comparison without bit-decomposition protocol," in *International Workshop on Public Key Cryptography*, Springer, 2007, pp. 343–360.

[44] C.-H. Yu and B.-Y. Yang, "Probabilistically correct secure arithmetic computation for modular conversion, zero test, comparison, mod and exponentiation," in *International Conference on Security and Cryptography for Networks*, Springer, 2012, pp. 426–444.

[45] T. Toft, "Sub-linear, secure comparison with two non-colluding parties," in *International Workshop on Public Key Cryptography*, Springer, 2011, pp. 174–191.

[46] H. Lipmaa and T. Toft, "Secure equality and greater-than tests with sublinear online complexity," in *International Colloquium on Automata, Languages, and Programming*, Springer, 2013, pp. 645–656.

[47] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications," in *International Colloquium on Automata, Languages, and Programming*, Springer, 2008, pp. 486–498.

[48] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," in *International Conference on Cryptology and Network Security*, Springer, 2009, pp. 1–20.

[49] T. K. Saha and T. Koshiba, "Private equality test using ring-lwe somewhat homomorphic encryption," in *2016 3rd Asia-Pacific World Congress on Computer Science and Engineering (APWC on CSE)*, IEEE, 2016, pp. 1–9.

[50] C. Gentry, S. Halevi, C. Jutla, and M. Raykova, "Private database access with he-over-oram architecture," in *International Conference on Applied Cryptography and Network Security*, Springer, 2015, pp. 172–191.

[51] H. Lipmaa, "Verifiable homomorphic oblivious transfer and private equality test," in *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, 2003, pp. 416–433.

[52] G. Couteau, "New protocols for secure equality test and comparison," in *International Conference on Applied Cryptography and Network Security*, Springer, 2018, pp. 303–320.

[53] M. Nateghizad, Z. Erkin, and R. L. Lagendijk, "Efficient and secure equality tests," in *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, IEEE, 2016, pp. 1–6.

[54] F. Karakoç, M. Nateghizad, and Z. Erkin, "Set-ot: A secure equality testing protocol based on oblivious transfer," in *Proceedings of the 14th International Conference on Availability, Reliability and Security*, 2019, pp. 1–9.

[55] T. Turban, "A secure multi-party computation protocol suite inspired by shamir's secret sharing scheme," M.S. thesis, Institutt for Telematikk, 2014.

[56] J. Baron, K. El Defrawy, K. Minkovich, R. Ostrovsky, and E. Tressler, "5pm: Secure pattern matching," in *International Conference on Security and Cryptography for Networks*, Springer, 2012, pp. 222–240.

[57] M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba, "Privacy-preserving wildcards pattern matching using symmetric somewhat homomorphic encryption," in *Australasian Conference on Information Security and Privacy*, Springer, 2014, pp. 338–353.

[58] T. K. Saha and T. Koshiba, "An enhancement of privacy-preserving wildcards pattern matching," in *International Symposium on Foundations and Practice of Security*, Springer, 2016, pp. 145–160.

[59] M. S. Riazi, E. M. Songhori, and F. Koushanfar, "Prisearch: Efficient search on private data," in *Proceedings of the 54th Annual Design Automation Conference 2017*, ACM, 2017, p. 14.

[60] O. Goldreich, S. Goldwasser, and S. Micali, “On the cryptographic applications of random functions,” in *Workshop on the Theory and Application of Crypto. Techniques*, Springer, 1984, pp. 276–288.

[61] O. Goldreich, *Foundations of Cryptography: Volume 2, Basic Applications.* Cambridge university press, 2009.

[62] A. Shamir, “How to share a secret,” *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[63] D. Beaver, “Efficient multiparty protocols using circuit randomization,” in *Annual International Cryptology Conference*, Springer, 1991, pp. 420–432.

[64] P. Pullonen, “Actively secure two-party computation: Efficient beaver triple generation,” M.S. thesis, University of Tartu, 2013.

[65] D. Rathee, T. Schneider, and K. Shukla, “Improved multiplication triple generation over rings via rlwe-based ahe,” in *International Conference on Cryptology and Network Security*, Springer, 2019, pp. 347–359.

[66] N. Döttling, S. Ghosh, J. B. Nielsen, T. Nilges, and R. Trifiletti, “Tinyole: Efficient actively secure two-party computation from oblivious linear function evaluation,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 2263–2276.

[67] I. Damgård and J. B. Nielsen, “Universally composable efficient multiparty computation from threshold homomorphic encryption,” in *Annual International Cryptology Conference*, Springer, 2003, pp. 247–264.

[68] R. Canetti, “Universally composable security: A new paradigm for cryptographic protocols,” in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, IEEE, 2001, pp. 136–145.

[69] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 2nd ed. Chapman and Hall/CRC, 2014.

[70] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold, “Keyword search and oblivious pseudorandom functions,” in *Theory of Cryptography Conference*, Springer, 2005, pp. 303–324.

[71] S. Jarecki and X. Liu, “Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection,” in *Theory of Cryptography Conference*, Springer, 2009, pp. 577–594.

[72] M. Naor, B. Pinkas, and O. Reingold, "Distributed pseudo-random functions and kdcs," in *International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 1999, pp. 327–346.

[73] B. Porat, *A Course in Digital Signal Processing.* Wiley, 1997.

[74] J. Arndt, *Matters Computational: ideas, algorithms, source code.* Springer Science & Business Media, 2010.

[75] A. V. Aho and J. E. Hopcroft, *The Design and Analysis of Computer Algorithms.* Pearson Education India, 1974.

[76] D. Vergnaud, "Efficient and secure generalized pattern matching via fast fourier transform," in *International Conference on Cryptology in Africa*, Springer, 2011, pp. 41–58.

[77] C. Hazay and Y. Lindell, "Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries," *Theory of Cryptography*, pp. 155–175, 2008.

[78] C. Hazay and T. Toft, "Computationally secure pattern matching in the presence of malicious adversaries," in *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, 2010, pp. 195–212.

[79] V. Kolesnikov, M. Rosulek, and N. Trieu, "Swim: Secure wildcard pattern matching from ot extension," in *International Conference on Financial Cryptography and Data Security*, Springer, 2018, pp. 222–240.

[80] M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba, "Secure pattern matching using somewhat homomorphic encryption," in *Proceedings of the 2013 ACM Workshop on Cloud Computing Security Workshop*, ACM, 2013, pp. 65–76.

[81] C. Hazay and T. Toft, "Computationally secure pattern matching in the presence of malicious adversaries," *Journal of Cryptology*, vol. 27, no. 2, pp. 358–395, 2014.

[82] P. Clifford and R. Clifford, "Simple deterministic wildcard matching," *Information Processing Letters*, vol. 101, no. 2, pp. 53–54, 2007.

[83] M. J. Fischer and M. S. Paterson, "String-matching and other products.," Massachusetts Inst of Tech Cambridge Project Mac, Tech. Rep., 1974.

[84] J. R. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik, "Privacy preserving error resilient dna searching through oblivious automata," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, ACM, 2007, pp. 519–528.

[85] K. B. Frikken, “Practical private dna string searching and matching through efficient oblivious automata evaluation,” in *IFIP Annual Conference on Data and Applications Security and Privacy*, Springer, 2009, pp. 81–94.

[86] K. Abrahamson, “Generalized string matching,” *SIAM Journal on Computing*, vol. 16, no. 6, pp. 1039–1051, 1987.

[87] S. R. Kosaraju, *Efficient string matching*, 1987.

[88] M. J. Atallah, F. Chyzak, and P. Dumas, “A randomized algorithm for approximate string matching,” *Algorithmica*, vol. 29, no. 3, pp. 468–486, 2001.

[89] K. Yessenov, “Dirichlet’s theorem on primes in arithmetic progressions,” *Web: http://people. csail. mit. edu/kuat/courses/dirichlet. pdf*, 2006.

[90] R. L. Rivest *et al.*, “Chaffing and winnowing: Confidentiality without encryption,” *CryptoBytes (RSA laboratories)*, vol. 4, no. 1, pp. 12–17, 1998.

[91] W. Du and M. T. Goodrich, “Searching for high-value rare events with uncheatable grid computing,” in *International Conference on Applied Cryptography and Network Security*, Springer, 2005, pp. 122–137.

[92] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms.* MIT press, 2009.

[93] Z. J. Czech, G. Havas, and B. S. Majewski, “Perfect hashing,” *Theoretical Computer Science*, vol. 182, no. 1-2, pp. 1–143, 1997.

[94] C. Chang and C. Chang, “An ordered minimal perfect hashing scheme with single parameter,” *Information Processing Letters*, vol. 27, no. 2, pp. 79–83, 1988.

[95] G. Jaeschke, “Reciprocal hashing: A method for generating minimal perfect hashing functions,” *Commun. ACM*, vol. 24, no. 12, pp. 829–833, Dec. 1981, ISSN: 0001-0782.

[96] V. G. Winters, “Minimal perfect hashing in polynomial time,” *BIT*, vol. 30, no. 2, pp. 235–244, Jun. 1990.

[97] M. L. Fredman, J. Komlós, and E. Szemerédi, “Storing a sparse table with o(1) worst case access time,” *Journal of the ACM*, vol. 31, no. 3, pp. 538–544, 1984.

[98] D. Belazzougui, F. C. Botelho, and M. Dietzfelbinger, “Hash, displace, and compress,” in *Algorithms - ESA 2009*, A. Fiat and P. Sanders, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 682–693.

[99] F. C. Botelho, R. Pagh, and N. Ziviani, "Practical perfect hashing in nearly optimal space," *Information Systems*, vol. 38, no. 1, pp. 108–131, 2013.

[100] A. Limasset, G. Rizk, R. Chikhi, and P. Peterlongo, "Fast and scalable minimal perfect hashing for massive key sets," *arXiv preprint arXiv:1702.03154*, 2017.

[101] Y. Nawaz, F. Olumofin, and S. H. Yuan, *Secure perfect hash function*, U.S. Pat. App. 14/733,000, Dec. 2016.

[102] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," *Journal of Computer and System Sciences*, vol. 18, no. 2, pp. 143–154, 1979.

[103] F. C. Botelho, R. Pagh, and N. Ziviani, "Simple and space-efficient minimal perfect hash functions," in *Workshop on Algorithms and Data Structures*, Springer, 2007, pp. 139–150.

[104] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *JACM*, vol. 33, no. 4, pp. 792–807, 1986.

[105] Y. Chen, R. Venkatesan, M. Cary, R. Pang, S. Sinha, and M. H. Jakubowski, "Oblivious hashing: A stealthy software integrity verification primitive," in *Information Hiding*, F. A. P. Petitcolas, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 400–414.

[106] P. Puzio, R. Molva, M. Önen, and S. Loureiro, "Perfectdedup: Secure data deduplication," in *Data Privacy Management, and Security Assurance*, Springer, 2015, pp. 150–166.

[107] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms & Data Analysis*. Cambridge Univ. Press, 2017.

[108] S. Micali and R. Sidney, "A simple method for generating and sharing pseudo-random functions, with applications to clipper-like key escrow systems," in *Annual International Cryptology Conf.*, Springer, 1995, pp. 185–196.

[109] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient batched oblivious prf with applications to private set intersection," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 818–829.

[110] J. Burns, D. Moore, K. Ray, R. Speers, and B. Vohaska, "Ec-oprf: Oblivious pseudo-random functions using elliptic curves.," *IACR Cryptology ePrint Archive*, 2017.

[111] Y. Lindell, K. Nissim, and C. Orlandi, "Hiding the input-size in secure two-party computation," in *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, 2013, pp. 421–440.

[112] M. J. Atallah and W. Du, “Secure multi-party computational geometry,” in *Workshop on Algorithms and Data Structures*, Springer, 2001, pp. 165–179.

[113] G. Wang, T. Luo, M. T. Goodrich, W. Du, and Z. Zhu, “Bureaucratic protocols for secure two-party sorting, selection, and permuting,” in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ACM, 2010, pp. 226–237.

[114] K. V. Jónsson, G. Kreitz, and M. Uddin, “Secure multi-party sorting and applications.,” *IACR Cryptology ePrint Archive*, vol. 2011, p. 122, 2011.

[115] K. E. Batcher, “Sorting networks and their applications,” in *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, ACM, 1968, pp. 307–314.

[116] K. Abrahamson, “Generalized string matching,” *SIAM J. on Computing*, vol. 16, no. 6, pp. 1039–1051, 1987.

[117] A. Amir and G. M. Landau, “Fast parallel and serial multidimensional approximate array matching,” in *Sequences*, Springer, 1990, pp. 3–24.

[118] M. J. Atallah, “Faster image template matching in the sum of the absolute value of differences measure,” *IEEE Transactions on Image Processing*, vol. 10, no. 4, pp. 659–663, 2001.

[119] B. Pinkas, T. Schneider, and M. Zohner, “Faster private set intersection based on ot extension,” in *23rd USENIX Security Symposium*, 2014, pp. 797–812.

[120] A. Abadi, S. Terzis, R. Metere, and C. Dong, “Efficient delegated private set intersection on outsourced private datasets,” *IEEE Transactions on Dependable and Secure Computing*, 2017.

[121] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, “Spot-light: Lightweight private set intersection from sparse ot extension,” in *Annual International Cryptology Conference*, Springer, 2019, pp. 401–431.

[122] A. Kavousi, J. Mohajeri, and M. Salmasizadeh, “Improved secure efficient delegated private set intersection,” *arXiv preprint arXiv:2004.03976*, 2020.