



NORTHWESTERN UNIVERSITY

Computer Science Department

Technical Report

Number: NU-CS-2026-36

September, 2026

Practical Private Set Operators in the Client-Server Setting

Yunqing Sun

Abstract

Private set operators are functions over private inputs that multiple parties can jointly evaluate while revealing only the prescribed output. Two such operators are intersection and join-and-compute, realized respectively by private set intersection (PSI), outputting the intersection of two sets, and private join and compute (PJC), outputting aggregates such as cardinality, sum, and inner product over matching records. Classical PSI and PJC target one-shot two-party settings where each party holds its full input. Real deployments rarely fit this model: servers maintain persistent datasets reused across many clients, and inputs are often split across multiple data owners. Existing protocols fall short: they lack cross-execution consistency, require per-execution server reprocessing, or incur substantial overhead for distributed inputs. This thesis develops efficient and provably secure protocols for private set operators in practical client-server settings, through three schemes together with new cryptographic primitives:

1. Inspired by password-checkup applications, we study client-output PSI in which the server publishes a one-time, linear-size encoding of its set, after which each client executes PSI with the server at cost linear only in its own set, with simulation-based security against malicious adversaries. A key ingredient is an efficient oblivious verifiable unpredictable function (OVUF).
2. We introduce committed vector oblivious linear evaluation (C-VOLE), which generates VOLE correlations on a pre-committed vector and serves as a unifying tool for zero-knowledge proofs

of committed values and actively secure multi-party computation. Built on a tailored LPN-based commitment, our matching C-VOLE protocols exploit the commitment structure to minimize the cost of binding the committed vector to the VOLE correlation, and efficiently instantiate a maliciously secure server-output PSI protocol.

3. Beyond intersection, we study computation over matching records from distributed datasets, motivated by applications such as privacy-preserving ad conversion measurement. We propose the first efficient approximate PJC protocol with communication sublinear in the input size. Its core is a new adaptation of the Alon-Matias-Szegedy (AMS) sketch, redesigned for efficient evaluation under fully homomorphic encryption via structured randomness.

Keywords

Private Set Intersection; Private Join and Compute; Secure Multiparty Computation; Oblivious Verifiable Unpredictable Function; Committed-Vector Oblivious Linear Evaluation; Secure Approximation

NORTHWESTERN UNIVERSITY

Practical Private Set Operators in the Client-Server Setting

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Yunqing Sun

EVANSTON, ILLINOIS

September 2026

© Copyright by Yunqing Sun 2026

All Rights Reserved

Abstract

Practical Private Set Operators in the Client-Server Setting

Private set operators are functions over private inputs that multiple parties can jointly evaluate while revealing only the prescribed output. Two such operators are intersection and join-and-compute, realized respectively by private set intersection (PSI), outputting the intersection of two sets, and private join and compute (PJC), outputting aggregates such as cardinality, sum, and inner product over matching records. Classical PSI and PJC target one-shot two-party settings where each party holds its full input. Real deployments rarely fit this model: servers maintain persistent datasets reused across many clients, and inputs are often split across multiple data owners. Existing protocols fall short: they lack cross-execution consistency, require per-execution server reprocessing, or incur substantial overhead for distributed inputs.

This thesis develops efficient and provably secure protocols for private set operators in practical client-server settings, through three schemes together with new cryptographic primitives:

1. Inspired by password-checkup applications, we study *client-output* PSI in which the server publishes a one-time, linear-size encoding of its set, after which each client executes PSI with the server at cost linear only in its own set, with simulation-based security against malicious adversaries. A key ingredient is an efficient *oblivious verifiable unpredictable function* (OVUF).
2. We introduce *committed vector oblivious linear evaluation* (C-VOLE), which generates VOLE correlations on a pre-committed vector and serves as a unifying tool for zero-knowledge proofs of committed values and actively secure multi-party computation. Built on a tailored LPN-based commitment, our matching C-VOLE protocols exploit the commitment structure to minimize the cost of binding the committed vector to the VOLE correlation, and efficiently instantiate a maliciously secure *server-output* PSI protocol.

3. Beyond intersection, we study computation over matching records from distributed datasets, motivated by applications such as privacy-preserving ad conversion measurement. We propose the first efficient approximate PJC protocol with communication sublinear in the input size. Its core is a new adaptation of the Alon–Matias–Szegedy (AMS) sketch, redesigned for efficient evaluation under fully homomorphic encryption via *structured randomness*.

Acknowledgment

I would like to express my deepest gratitude to my advisor, Professor Xiao Wang. He brought me into cryptography research and supported me through every step of my PhD. Beyond the technical training, the way he thinks about problems and the way he treats students is what I will take beyond this thesis. It is him who made my PhD a happy and rewarding experience.

I thank the members of my thesis defense committee for their time and for the feedback that improved this thesis. They are Prof. Peihan Miao, Prof. Jennie Rogers, Prof. V.S. Subrahmanian.

Many thanks to my collaborators on the work in this thesis, for patiently answering my questions and for the work they put into every project: Xinran Cai, Wei Dong, Jonathan Katz, Hanlin Liu, Mariana Raykova, Phillip Schoppmann, Chenkai Weng, Kang Yang, and Yu Yu. From every paper, I gained ideas, techniques, and ways of thinking I could not have found alone.

I am lucky to have spent time alongside kind and warm labmates and friends, whose company has made the everyday a pleasure.

Most of all, thanks to my parents and my husband for their unwavering belief, support, and love.

ABBREVIATIONS

2PC	Two-party computation
AMS	Alon–Matias–Szegedy (sketch)
AWS	Amazon Web Services
BB	Bulletin board
BLS	Barreto–Lynn–Scott (pairing-friendly elliptic curve)
C-VOLE	Committed vector oblivious linear evaluation
C2A	Commitment-to-authentication
CDN	Content delivery network
COT	Correlated oblivious transfer
CSP	Cryptographic service provider
DBDHI	Decisional bilinear Diffie–Hellman inversion
DDH	Decisional Diffie–Hellman
DHI	Diffie–Hellman inversion
DH	Diffie–Hellman
DP	Differential privacy
EA	Expand–accumulate (code)
EC	Expand–convolute (code)
EC2	Amazon Elastic Compute Cloud
EMP	Efficient multi-party protocol (toolkit)
FHE	Fully homomorphic encryption
FERRET	Fast extension for correlated OT
GC	Garbled circuit
HE	Homomorphic encryption
KOS	Keller–Orsini–Scholl (OT extension)
LAN	Local area network
LPN	Learning parity with noise
LowMC	Low multiplicative complexity (block cipher)
MoM	Median of means
MPVOLE	Multi-point vector oblivious linear evaluation

MPC	Multiparty computation
MtA	Multiplicative-to-additive (conversion)
MVZK	Multi-verifier zero-knowledge
NR	Naor–Reingold (pseudorandom function)
OKVS	Oblivious key–value store
OPRF	Oblivious pseudorandom function
OT	Oblivious transfer
OVUF	Oblivious verifiable unpredictable function
PJC	Private join and compute
PKI	Public key infrastructure
PPT	Probabilistic polynomial time
PRF	Pseudorandom function
PRG	Pseudorandom generator
PSI	Private set intersection
PSI-sum	Private set intersection sum
ROM	Random oracle model
RSA	Rivest–Shamir–Adleman
RVOLE	Random vector oblivious linear evaluation
SEAL	Simple Encrypted Arithmetic Library
SWHE	Somewhat homomorphic encryption
ThSWHE	Threshold somewhat homomorphic encryption
UC	Universal composability
U-MtA	Unbalanced multiplicative-to-additive (conversion)
VOLE	Vector oblivious linear evaluation
VOPRF	Verifiable oblivious pseudorandom function
VRF	Verifiable random function
VUF	Verifiable unpredictable function
WAN	Wide area network
ZK	Zero-knowledge
ZKP	Zero-knowledge proof
ZKPoK	Zero-knowledge proof of knowledge

Table of Contents

Abstract	3
Acknowledgment	5
ABBREVIATIONS	6
Table of Contents	8
List of Figures	13
List of Tables	16
Chapter 1. Introduction	18
1.1. Related Works	21
1.1.1. Overview of Existing PSI Protocols in the Client-Server Setting	21
1.1.2. OPRF-based PSI protocols	22
1.1.3. VOLE-based PSI protocol	23
1.1.4. PJC protocols	23
Chapter 2. Background	26
2.1. Notation	26
2.2. Preliminaries	27
2.2.1. Verifiable Unpredictable Functions	27
2.2.2. Learning Parity with Noise	28

2.2.3. Commitment Scheme	29
2.2.4. Somewhat Homomorphic Encryption	30
2.2.5. AMS Sketch Algorithm	31
2.2.6. Differential Privacy	31
2.3. Security Model and Functionalities	32
Chapter 3. Client–Server PSI from Oblivious Verifiable Unpredictable Function	39
3.1. Protocol Intuition	40
3.2. Protocol and Security Analysis	43
3.2.1. PSI from OVUF	44
3.2.1.1. Distributing the Server Encoding	46
3.2.2. Oblivious VUF	47
3.2.2.1. Encoding for Coalesced Multiplication	47
3.2.2.2. Imperfect MtA Protocol	49
3.2.2.3. An Oblivious VUF from Imperfect MtA	52
3.2.2.4. Complexity Analysis	58
3.3. Performance Evaluation	59
3.3.1. Benchmark Setup	59
3.3.2. Efficiency of Server’s Encoding	59
3.3.3. Efficiency of Online Computation	60
3.3.4. Comparison with Other Protocols	62
Chapter 4. Committed-Vector Oblivious Linear Evaluation and its Applications	65
4.1. Protocol Intuition	68
4.2. Protocol and Security Analysis	74
4.2.1. Commitment from LPN	74
4.2.1.1. LPN Commitment	75
4.2.1.2. Hybrid Commitment	77

	10
4.2.2. Committed-VOLE	78
4.2.2.1. Proof of Security	80
4.2.2.2. Complexity Analysis	83
4.2.3. PSI from Committed-VOLE	83
4.3. Other Applications	85
4.3.1. Commit-and-Prove VOLE-ZK	85
4.3.2. Multi-Party VOLE in MPC and MVZK	86
4.4. Performance Evaluation	87
4.4.1. Parameter Selection of Hybrid Commitment	87
4.4.2. Efficiency of Committed-VOLE	88
4.4.3. Comparison with Prior Work	90
4.4.3.1. Comparison with C2A Conversion.	90
4.4.3.2. Comparison with CVOLE with Input-revealing Identifiable-abort (IRIA).	91
4.4.4. Efficiency of C-VOLE based PSI and Comparison	92
4.4.4.1. Comparison with VOLE based PSI.	92
4.4.4.2. Comparison with Prior Work in Client–Server PSI.	94
Chapter 5. Communication-Efficient Private Join and Compute over Distributed Input Sets	96
5.1. Protocol Intuition	98
5.1.1. A New Framework for Distributed PJC	98
5.1.2. Why Classic Constructions Fail	100
5.1.3. Warm-Up Solution: HE-Friendly AMS Sketch	102
5.1.4. AMS Sketch with Imperfect Randomness	103
5.1.5. Supporting Differentially Private Output	106
5.2. AMS Sketch with Inadmissible Distribution	107
5.2.1. Δ -Inadmissible Distribution	107
5.2.2. AMS Sketch	109

	11
5.2.3. Inner Product Approximation	112
5.3. PJC Protocol for Distributed Input Sets	113
5.4. Obtaining Differential Privacy for Free	115
5.5. Performance Evaluation	117
5.5.1. Benchmark Accuracy	118
5.5.2. Protocol Efficiency	119
5.5.2.1. Server Initialization	120
5.5.2.2. Client Local Sketching	121
5.5.2.3. Server Aggregation and Output	122
5.5.2.4. Efficiency under Large Polynomial Modulus Degree.	122
5.5.3. Comparison with Distributed-PJC Baselines	123
Chapter 6. Conclusion	126
Bibliography	127
Appendix A. Appendix for Chapter 3	139
A.1. Batched OVUF with Improved Efficiency	139
A.1.1. Unbalanced Imperfect MtA	139
A.1.2. OVUF with Improved Efficiency	140
Appendix B. Appendix for Chapter 4	148
B.1. Hybrid Commitment Parameters based on EA-Code	148
B.1.1. Conservative parameters	148
B.1.2. Aggressive parameters	149
B.2. Primal-LPN based C-VOLE	149
B.2.1. Performance of Primal-LPN based C-VOLE	153
B.3. Performance of C-VOLE with Aggressive EA-code	155
B.4. Protocol of MPVOLE	157

	12
B.5. Protocol of VOLE	164
B.6. Proof of Theorem 9	165
Appendix C. Appendix for Chapter 5	169
C.1. Proof of Theorem 10	169
C.2. Proof of Throrem 12	174
C.3. Our 4-Wise Independent Construction	176
C.4. Performance under Large Polynomial Modulus Degree	178

List of Figures

1.1	The two-party private set intersection functionality.	18
2.1	The correlated OT functionality.	33
2.2	The random VOLE functionality.	33
2.3	The VOLE functionality.	34
2.4	The multi-point VOLE functionality.	35
2.5	Two-party threshold SWHE ideal functionality.	36
2.6	The bulletin board functionality.	36
2.7	The coin-tossing functionality.	36
2.8	The two-party computation functionality.	36
2.9	The private set intersection functionality.	37
2.10	The private set intersection functionality.	37
2.11	The ideal functionality for private join and compute over distributed input sets. . . .	38
3.1	The OVUF functionality.	44
3.2	PSI protocol in the $(\mathcal{F}_{\text{OVUF}}, \mathcal{F}_{\text{BB}})$ -hybrid model.	44
3.3	The MtA protocol in the $\mathcal{F}_{\text{COT}}$ -hybrid model.	51
3.4	OVUF protocol in the $(\mathcal{F}_{\text{COT}}, \mathcal{F}_{\text{BB}})$ -hybrid model with sub-protocol Π_{MtA}	53

3.5	Performance of our protocol. We show the performance of both phases. The one-time offline local encoding time for the server set is depicted in (a). The interactive online encoding time for the client set is shown in (b) under LAN network and (c) under WAN network. Both (b) and (c) utilize different OT methods (KOS/FERRET) and number of threads (1/4/32) for comparison.	60
3.6	Our performance under different network settings. We show our performance of time consumption as bandwidth varies in (a) and as latency varies in (b). (a) uses client set size 10^3 to compare performance under different OT methods (KOS/FERRET) and different thread numbers (1/4/32). (b) takes OT method KOS to compare performance as set sizes vary. Figure (c) shows our protocol's bandwidth consumption as the set size varies.	60
3.7	Server cost comparison with Blazing-PSI [14]. All experiments are run on AWS instance. Costs are estimated based on AWS instance pricing and network pricing.	63
4.1	The committed-VOLE functionality.	74
4.2	The committed-VOLE protocol based on dual-LPN-based hybrid commitment	80
4.3	Secure PSI protocol from Committed-VOLE.	84
4.4	Performance of C-VOLE protocol in LAN and WAN setting.	89
5.1	System model of Private Join and Compute over Distributed Input Sets. Two non-colluding servers, the Aggregator and the Cryptographic Service Provider (CSP), serve many clients. ① During initialization, the two servers jointly generate encrypted sketching randomness; the CSP can then go offline. ② Each client P_i locally computes an encrypted sketch of its input vector ($\mathbf{x}_i$ for set S_1 , or $\mathbf{y}_i$ for set S_2) and sends it to the Aggregator in a single message. ③ The Aggregator homomorphically aggregates all client sketches into $\llbracket \text{Sketch}(S_1) \rrbracket$ and $\llbracket \text{Sketch}(S_2) \rrbracket$. ④ Finally, the two servers jointly compute the approximate inner product $z \approx \langle S_1, S_2 \rangle$ via threshold decryption and 2PC.	96
5.2	Private join and compute over distributed input sets, secure against semi-honest adversaries in the $(\mathcal{F}_{\text{Rand}}, \mathcal{F}_{\text{ThSWHE}}, \mathcal{F}_{2\text{PC}})$ -hybrid model.	114
5.3	Accuracy Benchmark of Algorithm 1. (a) Empirical error across (n, d) combinations with $k = 400$, $t = 11$, $N = 2^{24}$, and $\text{supp}(\mathbf{a}) \cup \text{supp}(\mathbf{b}) = 1.5 \cdot 2^{18}$. (b,c) Empirical errors of Algorithm 1 against the standard AMS sketch on uniform and real-world data, with $d = 7$, $n \in \{2^6, 2^7, 2^8\}$; $\text{supp}(\mathbf{a}) \cup \text{supp}(\mathbf{b}) = 1.5 \cdot 2^{18}$ for uniform data and $\text{supp}(\mathbf{a}) \cup \text{supp}(\mathbf{b}) = 102,660$ for real-world data.	118

5.4	Amortized per-element cost for client local sketching. Tested under varying $d = \text{HW}(\mathbf{A}_i)$ and set size m	121
A.1	The U-MtA protocol in $\mathcal{F}_{\text{COT}}$ -hybrid.	140
A.2	The OVUF2 protocol in $(\mathcal{F}_{\text{COT}}, \mathcal{F}_{\text{BB}})$ -hybrid model with sub-protocol Π_{MtA} and $\Pi_{\text{U-MtA}}$	141
B.1	The committed-VOLE protocol based on primal-LPN-based hybrid commitment.	150
B.2	Performance of the C-VOLE protocol based on the primal-LPN assumption in LAN and WAN settings.	155
B.3	Performance of the C-VOLE protocol based on the dual-LPN assumption using aggressive EA-code in LAN and WAN settings.	157
B.4	Multi-point VOLE protocol.	160

List of Tables

3.1	Performance of unbalanced PSI with server set X and client set Y . Our protocol and [16] were tested with 32 threads, with [16] using 256 GB RAM for a 2^{28} server set. [12] is obtained based on numbers from their paper, which is based on faster hardware than our testbed. [20] is tested with 32 threads for the offline phase and 1 thread for the online phase. [78] is tested with 1 thread, except the case of $ X = 2^{28}$ where the online performance are extrapolated.	64
4.1	Dual-LPN-based hybrid commitment parameter selection for x of different lengths. Achieves 128-bit security.	88
4.2	Time consumption compared with [86]. With different network settings, we tested C-VOLE of 128-bit data block to compare with C2A of 64-bit data block. Both evaluations are with single thread.	90
4.3	Bandwidth consumption compared with [86]. we tested the bandwidth consumption of C-VOLE and C2A under varying input blocks. For each C-VOLE input block, we used a 128-bit data block, while for C2A, a 64-bit data block was employed for comparison.	90
4.4	Time consumption compared with [14]. (m, n) represents the server's set size m , and client's set size n . PSI Init is a one-time computation and PSI online is executed between the server and each individual client. The experiments are tested with 16 threads.	93
4.5	Communication overhead compared with [14]. (m, n) represents the server's set size m , and client's set size n . The experiments are executed with 16 threads. . . .	93
4.6	Time consumption compared with [6]. (m, n) represents the server's set size m , and the client's set size n . PSI Init is a one-time computation and PSI Online is executed between the server and each individual client. All experiments are executed with 16 threads. The online computation time for [6] with a client size of 2^{20} is derived from other tested results based on linear properties.	94

4.7	Communication overhead compared with [6]. (m, n) represents the server's set size m , and client's set size n . The online communication time for [6] with a client size of 2^{20} is derived from other tested results based on linear properties.	95
5.1	Approximation accuracy comparison under different k . Accuracy is measured as median over 1000 executions.	104
5.2	Initialization costs for different n	120
5.3	Server aggregation and output costs. Broken down by step. The encrypted sketch aggregation includes 2000 clients.	122
5.4	PJC over distributed inputs. We report per-client and total server cost under a large- w setting and a small- w setting; each client holds m elements. For the PJC-decomposition baselines [31], [32], [118], each client executes one pairwise PJC against every client on the opposite side; the server's role is summing pairwise inner products and is negligible. For the server-based generic 2PC baseline, client-side cost is negligible (preparing and sending secret shares). Time values are in seconds (s), minutes (m), hours (h), days (d), or years (y) as appropriate. Values marked with * are extrapolated.	124
B.1	Primal-LPN-based hybrid commitment parameters selection of x with different lengths. Achieves 128-bit security.	154
B.2	Dual-LPN-based hybrid commitment parameters selection of x with different lengths. Using aggressive parameters for EA-code. Commitment achieves 128-bit security level.	156
C.1	Server initialization costs under polynomial modulus degree 16384 for different seed lengths n	179
C.2	Amortized per-element client sketching cost (ms/elem.) under polynomial modulus degree 16384.	179
C.3	Server aggregation and output costs under polynomial modulus degree 16384. . . .	179

CHAPTER 1

Introduction

Private set intersection (PSI) is a secure multi-party computation primitive that enables two parties to compute the intersection of their private sets without revealing any additional information. Figure 1.1 presents the standard two-party PSI functionality. Two parties, holding private sets X and Y , respectively, securely compute the intersection $X \cap Y$, which is revealed to both parties while no additional information about the private inputs is leaked.

Beyond revealing the intersection itself, many PSI variants support private set computations over the intersection, such as cardinality, summation, and inner-product computation, while revealing only the corresponding outputs. These private set intersection and computation protocols have found numerous applications in practice, including genome testing [1], botnet detection [2], online advertising [3], compromised credential checking [4], and contact discovery [5].

Functionality $\mathcal{F}_{\text{PSI}}$
Upon receiving (input, X) from party P_1 and (input, Y) from party P_2 , where $X, Y \subseteq U$, the functionality computes the intersection $X \cap Y$ and outputs it to both parties.

Figure 1.1: The two-party private set intersection functionality.

However, many real-world applications are not well captured by the classical one-shot two-party PSI functionality shown in Figure 1.1. In practice, private set intersection and computation are often deployed in large-scale client–server or multi-client settings, where data is contributed by many parties across multiple protocol executions and servers maintain persistent datasets or state across sessions.

For example, in password checkup services, a server maintains a large database of compromised credentials, while clients query the database with their own credentials to determine whether

any have been compromised. Similarly, in contact discovery applications, servers maintain large databases of registered contact information, while clients seek to identify which of their contacts appear in the system. More generally, distributed private join and private data analysis applications involve datasets contributed by many clients, while one or more servers perform private set computations over aggregated data. These deployment settings introduce security, composability, and efficiency requirements that are not adequately captured by the classical two-party PSI functionality.

One might naively expect that these applications can be realized by simply executing a standard PSI or private set computation protocol repeatedly between individual pairs of parties. However, this approach suffers from both security and efficiency limitations. From a security perspective, independently invoking a secure PSI protocol does not guarantee consistency across executions. For example, a malicious server may selectively use different datasets for different clients, such as withholding compromised credentials from targeted users in a password checkup service. From an efficiency perspective, repeatedly processing large datasets for every pairwise execution can be prohibitively expensive. Similarly, in distributed private join computation and private data analysis, naively performing pairwise computations across client datasets leads to quadratic communication and computation costs in the number of participants, making such approaches poorly scalable in large deployments.

Moving from the classical two-party PSI setting to real-world client–server deployments is therefore not simply a matter of repeatedly running a two-party protocol. The underlying functionality changes fundamentally, introducing new security and efficiency requirements. Addressing these challenges requires rethinking both the cryptographic functionalities and protocol designs, as well as developing new cryptographic primitives and techniques tailored to practical client–server applications.

This thesis studies private set computation in the client–server setting, with a focus on achieving strong security guarantees and communication efficiency. Beginning with private set intersection (PSI), we progressively extend our techniques and functionalities to more general private join

computation tasks arising in distributed privacy-preserving data analysis.

First, inspired by password checkup applications, we study client-output PSI in Chapter 3 [6]. In our setting, the server publishes a linear-size offline encoding of its dataset once, after which many clients can efficiently execute lightweight online sessions whose complexity is linear in the size of the client’s set and locally recover the intersection. The main challenge is to design a reusable server encoding that remains efficient while being verifiably bound to a single committed server dataset across all executions. To address this challenge, we introduce oblivious verifiable unpredictable function (OVUF) and provide efficient constructions for it. This yields a new maliciously secure OVUF-based approach for reusable client–server PSI. The OVUF primitive may also be of independent interest.

Second, we consider server-output PSI in Chapter 4 [7]. We extend state-of-the-art two-party PSI techniques to the client–server setting by introducing a new commitment scheme together with matching committed-vector oblivious linear evaluation (C-VOLE) protocols. Our resulting construction achieves malicious security and the desired consistency guarantees while maintaining online communication complexity linear in the size of the client’s set. Beyond PSI, the C-VOLE techniques developed in this chapter have broader applicability to VOLE-based zero-knowledge proofs (ZKPs) and, more generally, maliciously secure multi-party computation (MPC).

Finally, we move beyond computing the intersection itself and study computation over the intersection, motivated by privacy-preserving data analysis applications. We refer to this general problem as private join computation (PJC). As before, we consider settings where input datasets are contributed by multiple clients and jointly processed by two servers. We take an initial step in this direction by introducing, to the best of our knowledge, the first treatment of secure approximate computation for distributed PJC under the semi-honest model with sublinear communication efficiency, in Chapter 5.

We implement and prototype all proposed constructions and conduct extensive experimental evaluations. The results demonstrate that our protocols achieve practical efficiency and provide favorable communication and runtime performance in the client–server setting.

1.1. Related Works

We first provide a brief overview of prior work on two-party PSI protocols and discuss their limitations in the client–server setting. We then give a more detailed discussion of oblivious pseudorandom function (OPRF)-based and vector oblivious linear evaluation (VOLE)-based PSI protocols, which serve as the two main starting points for our PSI constructions in the client–server setting. Finally, we review existing private join and compute (PJC) protocols and discuss their limitations in the distributed-input setting.

1.1.1. Overview of Existing PSI Protocols in the Client-Server Setting

Existing PSI protocols fall short in the client-server setting, as they either fail to achieve malicious security or lack efficiency in this unbalanced setting.

1. There have been works focusing on the unbalanced setting based on oblivious PRF [8], [9], [10], [11] to achieve online communication only linear in the size of the client’s set. Kiss et al. [12] systematically evaluated this approach for unbalanced PSI, but none of the protocols can achieve fully malicious security against a corrupted sender (which is the server in our case).
2. The state-of-the-art malicious PSI protocols [13], [14] based on vector oblivious linear evaluation require online communication linear to the size of both parties’ sets. It requires more than 16 bytes of communication per element in the parties’ sets; for a server holding 10^8 elements, this translates to around 1.6 GB of communication when interacting with each client. Furthermore, even with a full malicious secure protocol, a server might use different set for different clients.
3. There are also works incorporate homomorphic encryption [12], [15], [16] for unbalanced PSI. By incorporating OPRF, the protocols [12] can achieve malicious security against clients and the attack from a malicious server can only be launched in certain ways. However, none of these protocols can protect against a malicious server without using heavy zero-knowledge proof techniques [11].

4. Other works like Laconic PSI [17], [18], [19] also achieve small communications; however, their goal is to minimize the overhead of the server and, as a result, the receiver’s computational cost is linear to the sender’s large set.

To summarize, while many PSI protocols allow the server to reuse work done in an offline phase [12], [15], [16], [20], existing solutions with this property do not achieve (full) security against malicious attackers. On the other hand, while several recent works have shown actively secure PSI protocols [13], [14], [21], [22], [23], [24], all such protocols require the server to do work linear in the size of its input in each execution.

1.1.2. OPRF-based PSI protocols

Prior OPRF-based PSI works [13], [23], [24] relies on the following idea: The server begins by generating a private key sk . Then, in an offline phase, the server with set $X = \{x_i\}$ computes a deterministic encoding $EX = \{ex_i = \text{Enc}(sk; x_i)\}$ of its set. To compute the intersection with a set $Y = \{y_i\}$ held by some client, the server runs an interactive protocol with the client (with complexity linear in $|Y|$) that allows the client to learn $EY = \{\text{Enc}(sk; y_i)\}$. Once the client learns EX it can compute $EX \cap EY$, from which it can deduce the elements in the intersection. In prior work, the encoding of an element was done by setting $\text{Enc}(sk; x_i) = F_{sk}(x_i)$ for a pseudorandom function F . The server can locally compute this encoding, while the client can compute this encoding by interacting with the server in an OPRF sub-protocol.

This suffices to achieve semi-honest security: informally, the client learns nothing beyond the intersection (and the size of the server’s set), since each encoding outside the intersection is random; the server learns nothing about the client’s set due to the obliviousness of the OPRF protocol. Unfortunately, the above does not appear to allow for proving security against a malicious server. In that setting, it must be possible to extract the server’s (effective) input from its published encoding EX . There is no obvious way to do this in the above protocol. In particular, for a single value ex published by the server, a simulator has no way to even tell whether ex is a (correct) encoding of some element or a garbage value that will not match anything.

1.1.3. VOLE-based PSI protocol

Vector oblivious linear evaluation (VOLE)-based PSI protocols [13], [14] has been the state-of-the-art in the actively secure 2-party PSI setting. It relies on the following idea: The server's input set is encoded into a vector $\mathbf{p}$, such that $\text{Decode}(\mathbf{p}, x) = H_1(x)$ for $x \in X$. Then, the server feeds its input set into the VOLE functionality to generate a VOLE correlation of $\mathbf{p}$ with a corresponding client. The VOLE correlation is denoted as $\langle \mathbf{p} \rangle$, where $M[\mathbf{p}] = K[\mathbf{p}] + \mathbf{p} \cdot \Delta$. In this setup, $\mathbf{p}$ and $M[\mathbf{p}]$ are given to the server, while $K[\mathbf{p}]$ and the global key Δ are given to the client. The server identifies any client's set elements in the intersection using an OPRF defined by $\langle \mathbf{p} \rangle$. Specifically, the client encodes its set elements by $F(x) = H_2(\text{Decode}(K[\mathbf{p}], x) + \Delta \cdot H_1(x), x)$, where H_1 and H_2 are two hash functions, and sends the encodings to the server. The server encodes its set elements by $X' := \{H_2(\text{Decode}(M[\mathbf{p}], x), x) | x \in X\}$ and checks for matches with the client's encoding. The two encodings match if the client's set element is located in the intersection, which satisfies $\text{Decode}(\mathbf{p}, x) = H_1(x)$. In this case, VOLE is utilized as a black-box component in the VOLE-PSI protocol.

Given a maliciously secure instantiation of the VOLE functionality, this scheme achieves fully active security. However, it remains insufficient for a malicious PSI protocol in the client-server setting. A malicious server can change its input set for each execution or modify it between repeated executions with the same client, potentially leaking information about the client's input. Moreover, in each execution, the server must generate a new encoding of its set and establish a fresh VOLE correlation from scratch, which can incur significant communication overhead.

1.1.4. PJC protocols

Private Join and Compute (PJC) [25], [26] is an important cryptographic primitive that enables two parties to perform data analytics over matching records without revealing their individual data. It was originally proposed by Google [27] for privacy-preserving ad conversion measurement between advertisers and websites. A basic form of PJC is private set intersection sum (PSI-sum) [28], [29], [30], which reveals the sum of values associated with records in the intersection. A more ex-

pressive variant is PSI-inner product [25], [26], [31], [32], [33], which privately joins two datasets on a key and computes the inner product of the aligned values.

All of the above PJC protocols consider two parties, each holding its entire dataset. A typical example from ad conversion measurement involves two parties: 1) an advertisement vendor (AV), which holds a dataset of $(customer_id, duration)$ pairs, where *duration* denotes the amount of time a customer was exposed to an ad; 2) a transaction data provider (TDP), which holds a dataset of $(customer_id, spending)$ pairs, where *spending* denotes the amount spent by the customer after viewing the ad. The desired computation can be expressed as the following SQL query:

```
SELECT SUM(TDP.spending * AV.duration)
FROM AV
INNER JOIN TDP
ON AV.cid = TDP.cid;
```

However, applying existing PJC protocols in this setting relies on a strong assumption: there exist two parties that actually hold all advertisement records and all transaction records, respectively. In practice, this assumption is often unrealistic. For example, after seeing an ad, a customer may purchase a product from Amazon, Target, or other retailers. In that case, the transaction data is partitioned across multiple mutually untrusted parties that are unwilling to share their customer records with one another. Similarly, advertisement records may also be split across multiple vendors or jurisdictions, where data sharing is restricted or prohibited.

It is therefore desirable to have a PJC protocol where the two input sets are distributed over many clients. However, extending existing PJC protocols to this setting is non-trivial, since they inherently assume that each party holds its entire input. There are two natural candidate solutions, but both fall short:

- **PJC Decomposition.** Since $\langle \sum_i x_i, \sum_j y_j \rangle = \sum_{i,j} \langle x_i, y_j \rangle$, we can decompose PJC over distributed input sets into a quadratic number of smaller PJC computations. Thus, one can support

PJC over distributed input sets by each client running a PJC with every client for the other set. However, each pairwise PJC reveals its output, so this approach leaks every pairwise inner product to the servers; this is costly and leaks far more information than the target aggregate.

- **Server-based Generic 2PC.** Alternatively, we can let all clients secret-share their inputs to two non-colluding servers, who can then run a generic 2PC to aggregate and compute the final PJC values. The cost of 2PC would be quasi-linear to the sum of the sizes of all client inputs, which also leads to prohibitive cost.

CHAPTER 2

Background

2.1. Notation

Define statistical and computational security parameters λ and κ . We let $\text{negl}(\cdot)$ denote a negligible function, and use $\log$ to denote logarithms in base 2. We use $H_\infty(\gamma)$ to denote the min-entropy of a random variable γ . $\mathbb{F}$ denotes a finite field, and we let $[n] = \{1, \dots, n\}$. We write $a \leftarrow \mathbb{F}$ to indicate that a is sampled uniformly from $\mathbb{F}$.

We use bold upper-case letters like $\mathbf{A} \in \mathbb{F}^{m \times n}$ for matrices, where m denotes the number of rows and n the number of columns. We use bold lower-case letters like $\mathbf{a}$ for row vectors, where $\mathbf{a}[i]/a_i$ denotes the i th component of $\mathbf{a}$ (with $\mathbf{a}[1]/a_1$ the first entry). $\mathbf{a}^T$ denotes the transposed vector of $\mathbf{a}$. We write $\mathbf{a} \circ \mathbf{b}$ for the Hadamard product of two vectors, and use $\parallel$ to denote concatenation. $\text{HW}(\mathbf{a})$ denotes the Hamming weight of $\mathbf{a}$, $\text{supp}(\mathbf{a}) := \{i \mid \mathbf{a}[i] \neq 0\}$ denotes its support, and $\mathbf{1}_n$ denotes the all-ones vector of length n . For $b \in \mathbb{Z}_q$, we use $\text{Bits}(b)$ to denote the bit decomposition of b . Define a regular- w vector $\mathbf{e}$ as $\mathbf{e} = (\mathbf{e}^1 \parallel \dots \parallel \mathbf{e}^w) \in \mathbb{F}^{w \cdot 2^L}$, where $\mathbf{e}^i \in \mathbb{F}^{2^L}$ is a unary vector. We also denote $\mathbf{e}$ as a multi-point vector and $\mathbf{e}^i$ as a single-point vector. For a vector $\mathbf{a} = (a_1, \dots, a_n)$, we use $\mathbf{K}[\mathbf{a}]$ to represent $\mathbf{K}[a_1] \parallel \dots \parallel \mathbf{K}[a_n]$, and let $\mathbf{M}[\mathbf{a}] = \mathbf{M}[a_1] \parallel \dots \parallel \mathbf{M}[a_n]$.

We use $\llbracket a \rrbracket$ to denote the ciphertext from the encryption of a under a homomorphic encryption scheme. For $a, b \in \{0, 1\}$, homomorphic XOR of $\llbracket a \rrbracket, \llbracket b \rrbracket$ is denoted as $\llbracket a \oplus b \rrbracket = \llbracket a \rrbracket + \llbracket b \rrbracket - 2 \cdot \llbracket a \rrbracket \cdot \llbracket b \rrbracket$. For $a \in \{0, 1\}$, we define its signed encoding as $b = 2a - 1 \in \{-1, 1\}$; then $\llbracket b \rrbracket$ is homomorphically evaluated as $2 \cdot \llbracket a \rrbracket - \llbracket 1 \rrbracket$. Each of these operations can be naturally extended to vectors by applying them element-wise.

2.2. Preliminaries

2.2.1. Verifiable Unpredictable Functions

A verifiable random function (VRF) is a keyed function whose output is verifiable given a public key and an associated proof; informally, the output should be indistinguishable from random without the proof. A verifiable unpredictable function (VUF) is a weaker primitive, where all that is required is for the output to be unpredictable. Note, however, that in contrast to a VRF, the output of a VUF can be verified without any additional proof. We only rely on VUFs in this thesis.

Definition 1. A VUF consists of algorithms $(\text{Gen}, F, \text{Vrfy})$ where

- Gen takes as input 1^λ , and outputs a key pair (sk, pk) .
- F takes as input a secret key sk and an element x and outputs y .
- Vrfy takes as input a public key pk and elements x, y and outputs a bit.

It is required that for all (sk, pk) output by Gen and all x in the domain of F , we have $\text{Vrfy}(\text{pk}, x, F_{\text{sk}}(x)) = 1$.

Moreover, the following security properties hold:

Uniqueness: There do not exist (pk, x, y_1, y_2) with $y_1 \neq y_2$ and

$$\text{Vrfy}(\text{pk}, x, y_1) = 1 = \text{Vrfy}(\text{pk}, x, y_2).$$

Unpredictability: For any efficient algorithm $\mathcal{A}$, the following is negligible:

$$\Pr [(\text{sk}, \text{pk}) \leftarrow \text{Gen}(1^\lambda); (x, y) \leftarrow \mathcal{A}^{F_{\text{sk}}(\cdot)}(\text{pk}) : y = F_{\text{sk}}(x)] ,$$

where $\mathcal{A}$ does not query its oracle on x .

For technical reasons, we also require that it is possible to identify whether a key pair (sk, pk) is valid or not; for a valid key pair $\text{Vrfy}(\text{pk}, x, F_{\text{sk}}(x)) = 1$ for all x , while for an invalid key pair $\text{Vrfy}(\text{pk}, x, F_{\text{sk}}(x)) = 0$ for all x .

We recall the VUF proposed by Dodis and Yampolskiy [34], based on prior work of Boneh and Boyen [35]. Let $\mathbb{G}, \mathbb{G}_T$ be cyclic groups of prime order q , with $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ an efficiently computable pairing.¹ The Dodis-Yampolskiy VUF is defined as follows:

1. Gen: Choose $\text{sk} \leftarrow \mathbb{Z}_q$ and output $(\text{sk}, \text{pk} = g^{\text{sk}})$.
2. $F_{\text{sk}}(x) = g^{1/(\text{sk}+x)}$. (We define $F_{\text{sk}}(-\text{sk}) = 1$.)
3. $\text{Vrfy}(\text{pk}, x, y)$ outputs 1 iff $e(g^x \cdot \text{pk}, y) = e(g, g)$, or $\text{pk} = g^{-x}$ and $y = 1$.

The security of this construction for small domain relies on variations of (bilinear) Diffie-Hellman assumptions. Subsequent work [36] shows its security for general input domains in the generic group model.

2.2.2. Learning Parity with Noise

The Learning Parity with Noise (LPN) problem is defined as recovering a secret vector given a noisy system of linear equations. It has found many applications in cryptography, as it has strong security guarantees. The hardness definition of primal-LPN and dual-LPN [37] are presented in Definition 2 and Definition 3 respectively as follows:

Definition 2. (*Primal LPN*) Let $\mathcal{C}$ be a probabilistic code generation algorithm such that $\mathcal{C}(N, k, \mathbb{F})$ outputs a matrix $\mathbf{A} \in \mathbb{F}^{k \times N}$. We say that the decisional primal-LPN(N, k, t) problem is (T, ε) -computationally hard for every probabilistic distinguisher $\mathcal{D}$ running in time T if

$$|\Pr[\mathcal{D}(\mathbf{A}, \mathbf{A} \cdot \mathbf{x} + \mathbf{e}) = 1] - \Pr[\mathcal{D}(\mathbf{A}, \mathbf{u}) = 1]| \leq \varepsilon,$$

where $\mathbf{x} \leftarrow \mathbb{F}^N$, $\mathbf{u} \leftarrow \mathbb{F}^k$, and $\mathbf{e} \leftarrow \mathbb{F}^k$ with weight t .

¹Our implementation uses a Type-III pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ for efficiency, but for simplicity we describe our protocols using Type-I pairings.

Definition 3. (Dual LPN) Let $\mathcal{C}^\perp$ be a probabilistic code generation algorithm such that $\mathcal{C}^\perp(N, k, \mathbb{F})$ outputs a matrix $\mathbf{H} \in \mathbb{F}^{k \times N}$. We say that the decisional dual-LPN(N, k, t) problem is (T, ε) -computationally hard for every probabilistic distinguisher $\mathcal{D}$ running in time T if

$$|Pr[\mathcal{D}(\mathbf{H}, \mathbf{H} \cdot \mathbf{e}) = 1] - Pr[\mathcal{D}(\mathbf{H}, \mathbf{u}) = 1]| \leq \varepsilon,$$

where $\mathbf{e} \leftarrow \mathbb{F}^N$ with weight t , and $\mathbf{u} \leftarrow \mathbb{F}^k$.

2.2.3. Commitment Scheme

A commitment scheme is defined as a sender committing to a chosen value while keeping it hidden from others and opening the value later to let others verify it.

1. **ParamGen:** The sender samples public parameter k from a parameter space K and publish k to other parties.
2. **Commit:** The sender picks message $x \leftarrow \mathbb{F}^m$ and computes $\{\text{com}_x, \text{decom}_x\} \leftarrow \text{Com}(x)$, where $\text{decom}_x \in \mathbb{F}^*$. The sender sends com_x to the receiver.
3. **Open:** The sender sends x and decom_x to the receiver. The receiver computes $0/1 \leftarrow \text{Open}(\text{com}_x, \text{decom}_x, x)$

A commitment scheme with computational hiding and statistical binding is defined as follows.

1. **Computational hiding:** with overwhelming probability over the choice of k , for every $x, x' \in \mathbb{F}^m$ and $\{\text{com}_x, \text{decom}_x\} \leftarrow \text{Com}(x)$, $\{\text{com}'_x, \text{decom}'_x\} \leftarrow \text{Com}(x')$, the distributions of com_x and com'_x are computationally indistinguishable.
2. **Statistical binding:** with overwhelming probability over the choice of k , the following equation holds with probability $\text{negl}(\lambda)$,

$$\text{Open}(\text{com}_x, \text{decom}_x, x) \wedge \text{Open}(\text{com}_x, \text{decom}'_x, x') = 1$$

2.2.4. Somewhat Homomorphic Encryption

Somewhat Homomorphic Encryption (SWHE) is a form of encryption that allows limited-depth computation over encrypted data without access to the secret key [38]. A SWHE scheme is described by the following randomized algorithms:

- **Key Generation:** $(\text{params}, \text{sk}, \text{pk}, \text{ek}) \leftarrow \text{KeyGen}(1^\lambda, 1^L)$. The key generation algorithm takes as input a security parameter λ and a maximum multiplication depth L . It outputs public parameters params , a secret key sk for decryption, a public key pk for encryption, and an evaluation key ek for homomorphic evaluation. Here, params is an implicit input to the following algorithms, omitted for simplicity.
- **Encryption:** $\llbracket m \rrbracket \leftarrow \text{Enc}(\text{pk}, m)$. The encryption algorithm takes as input a public key $\text{pk} \in \mathcal{K}$ and a message $m \in \mathcal{M}$, and outputs a ciphertext $\llbracket m \rrbracket \in \mathcal{C}$.
- **Decryption:** $m \leftarrow \text{Dec}(\text{sk}, \llbracket m \rrbracket)$. The decryption algorithm takes as input a secret key sk and a ciphertext $\llbracket m \rrbracket$, and outputs $m \in \mathcal{M}$.

The SWHE scheme needs to satisfy the following properties.

1. **Message homomorphism.** For any $(\text{params}, \text{sk}, \text{pk}, \text{ek}) \leftarrow \text{KeyGen}(1^\lambda, 1^L)$, any messages $m_i \in \mathcal{M}$ with its ciphertexts $\llbracket m_i \rrbracket \leftarrow \text{Enc}(\text{pk}, m_i)$ for each $i \in [\ell]$, where $\ell \geq 1$, the following holds with probability $1 - \text{negl}(\lambda)$: for any polynomial-size circuit f of depth at most L ,

$$\text{Dec}(\text{sk}, \tilde{f}(\llbracket m_1 \rrbracket, \dots, \llbracket m_\ell \rrbracket)) = f(m_1, \dots, m_\ell) .$$

Here, $\tilde{f}$ denotes the homomorphic evaluation of the circuit f on ciphertexts, which can be performed in time $\text{poly}(\lambda)$.

2. **Ciphertexts pseudorandom.** For any secret key sk kept private, the distribution of ciphertexts is computationally indistinguishable from the uniform distribution over the ciphertext space.

2.2.5. AMS Sketch Algorithm

The AMS sketch is a logarithmic-size sketch for estimating the second frequency moment of a stream. It is an optimized and specialized instance of the estimator for the k th frequency moment, as proposed by [39].

Theorem 1 (The AMS Sketch [39]). *Let $\mathbf{x}, \mathbf{y} \in \mathbb{Z}^N$, $\mathbf{r} \in \{-1, +1\}^N$ be a random four-wise independent vector, $Z_x = \sum_{i \in [N]} \mathbf{r}[i] \mathbf{x}[i]$, $Z_y = \sum_{i \in [N]} \mathbf{r}[i] \mathbf{y}[i]$. Then $\mathbb{E}[Z_x \cdot Z_y] = \langle \mathbf{x}, \mathbf{y} \rangle$ and $\text{Var}[Z_x \cdot Z_y] = O(\|\mathbf{x}\|_2^2 \|\mathbf{y}\|_2^2)$.*

To bound the estimation variance, the median-of-means technique is applied, which maintains $O\left(\frac{1}{\varepsilon^2} \ln \frac{1}{\delta}\right)$ independent estimators to achieve an (ε, δ) -approximation.

Theorem 2 (Median of Mean [40]). *Let $\{X_{i,j}\}_{i \in [t], j \in [k]}$ be i.i.d. random variables such that $\mathbb{E}[X_{i,j}] = Q$. Define MoM operator as*

$$\text{MoM}(\{X_{i,j}\}_{i \in [t], j \in [k]}) \stackrel{\text{def}}{=} \text{Median}_{i \in [t]} \left(\frac{1}{k} \sum_{j \in [k]} X_{ij} \right).$$

Let $t = O\left(\log \frac{1}{\delta}\right)$ and $k = O\left(\frac{\text{Var}[X]}{\varepsilon^2 (\mathbb{E}[X])^2}\right)$. Then

$$\Pr \left[\left| \text{MoM}(\{X_{i,j}\}_{i \in [t], j \in [k]}) - Q \right| \geq \varepsilon Q \right] \leq \delta.$$

2.2.6. Differential Privacy

Unlike homomorphic encryption, which protects the privacy of data during computation, *differential privacy* [41] protects the privacy of individuals in the released output of a computation. Informally, it requires that the output distribution of an analysis does not change significantly when any single individual's record is added to or removed from the dataset. Consequently, the presence or absence of any one record is statistically hidden, providing a rigorous information-theoretic privacy guarantee. Here, two datasets D and D' are called *neighbors*, denoted by $D \sim D'$, if they differ by one record.

Theorem 3 (ε -differential privacy). *A randomized algorithm $\mathcal{M}$ with range $\mathcal{R}$ is ε -differential privacy (DP) if, for any neighboring datasets $D \sim D'$, and for all measurable subsets $S \subseteq \mathcal{R}$,*

$$\Pr[\mathcal{M}(D) \in S] \leq e^\varepsilon \Pr[\mathcal{M}(D') \in S] .$$

A commonly used DP mechanism is the *Laplace mechanism*. Given a query $Q : \mathcal{D} \rightarrow \mathcal{R}$, its *global sensitivity* (GS_Q) is defined as

$$\text{GS}_Q \stackrel{\text{def}}{=} \max_{D, D' : D \sim D'} |Q(D) - Q(D')|.$$

Then, the Laplace mechanism $\mathcal{M}(D) = Q(D) + \eta$ satisfies ε -DP, where η is drawn from a Laplace distribution with scale GS_Q/ε , i.e., $\eta \sim \text{Lap}(\text{GS}_Q/\varepsilon)$. DP enjoys some properties, one of the most commonly used ones is *composition theory*:

Lemma 1 (Composition theory). *Let $\{\mathcal{M}_1, \dots, \mathcal{M}_k\}$ be a series of ε -DP mechanisms. Then $\mathcal{M}_1(D), \mathcal{M}_2(D), \dots, \mathcal{M}_k(D)$ is $k\varepsilon$ -DP.*

2.3. Security Model and Functionalities

We prove security of our protocols in the UC framework [42], assuming static corruptions. Below we introduce the functionalities used across the thesis. We define the PSI functionalities in the client–server setting and the PJC functionality with distributed inputs below. We defer other proposed cryptographic primitive functionalities, such as OVUF and C-VOLE, to the corresponding chapters for clarity. We omit session IDs for readability. Some of our protocols rely on a programmable random oracle, which can be formalized as a functionality within the generalized UC framework [43]; we do not do this explicitly here. We assume authenticated channels, but do not require private channels.

Correlated Oblivious Transfer. Correlated oblivious transfer (COT) is a variant of oblivious transfer, shown in Figure 2.1. In the functionality we used here, party P_A and party P_B obtain

additive shares of either 0 or the correlation value τ_i chosen by P_A , depending on P_B 's choice bit w_i . P_A learns nothing about the P_B 's choice vector, while P_B learns only its corresponding additive shares.

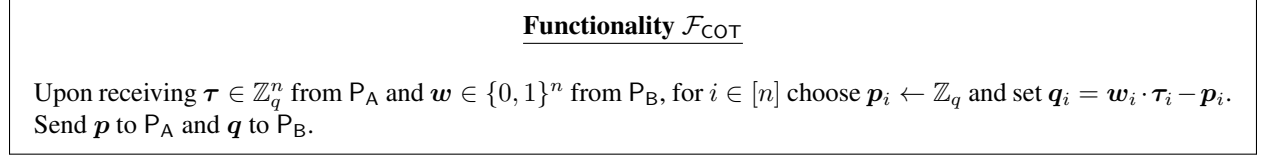


Figure 2.1: The correlated OT functionality.

Random Vector Oblivious Linear Evaluation. Random Vector Oblivious Linear Evaluation (RVOLE) functionality is a two party functionality that samples a vector $x \leftarrow \mathbb{F}^m$ to P_A , a field element $\Delta \leftarrow \mathbb{F}$ and a vector $K[x] \leftarrow \mathbb{F}^m$ to P_B , and delivers $M[x] = K[x] + x \cdot \Delta \in \mathbb{F}^m$ to P_A . RVOLE can be efficiently generated by LPN-based approaches [44], [45], [46], [47]. We take the functionality of random VOLE depicted in Figure 2.2 as the fundamental building block to construct committed VOLE.

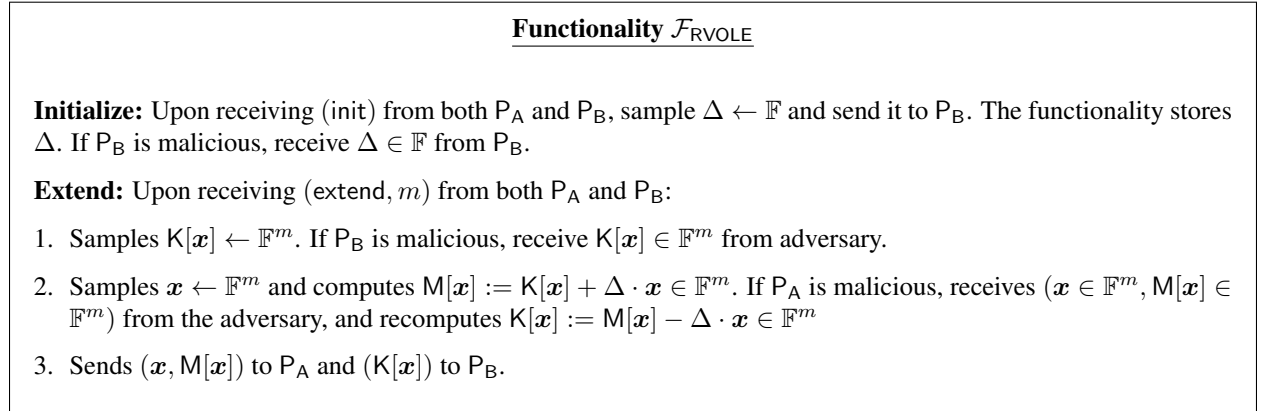


Figure 2.2: The random VOLE functionality.

Vector Oblivious Linear Evaluation. Vector Oblivious Linear Evaluation (VOLE) is generating VOLE relation of a designated vector rather than a random vector. The ideal functionality is shown in Figure 2.3. The functionality samples field element $\Delta \leftarrow \mathbb{F}$ to P_B for initialization.

Then, for P_A 's input vector $\mathbf{x} \in \mathbb{F}^m$, the functionality samples $K[\mathbf{x}] \leftarrow \mathbb{F}^m$ to P_B and delivers $M[\mathbf{x}] = K[\mathbf{x}] + \mathbf{x} \cdot \Delta \in \mathbb{F}^m$ to P_A . VOLE is typically instantiated from Random VOLE (RVOLE). It serves as a building block in the instantiation of $\mathcal{F}_{C\text{-VOLE}}$.

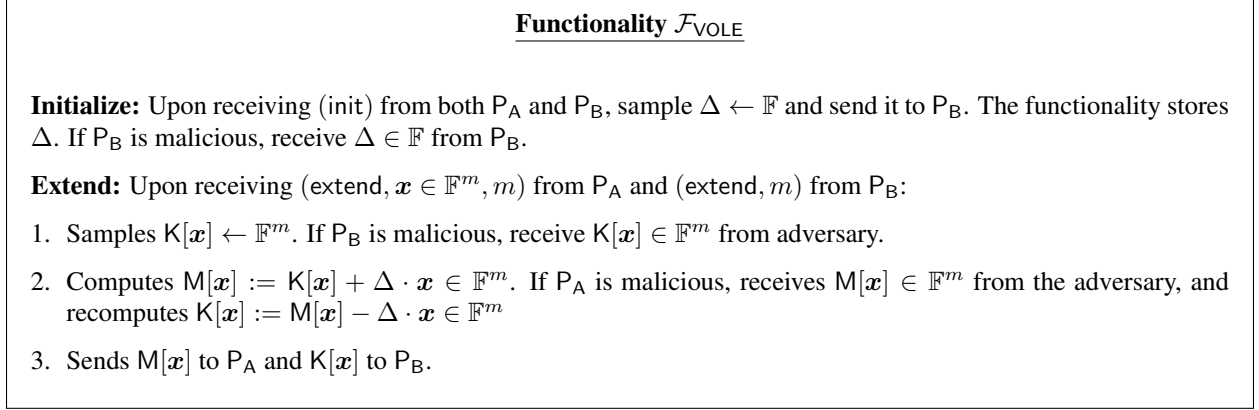


Figure 2.3: The VOLE functionality.

Multi-point Vector Oblivious Linear Evaluation. Multi-point Vector Oblivious Linear Evaluation (MPVOLE) is a variant of VOLE that generates the VOLE relation of a multi-point vector, a regular sparse vector formed by concatenating multiple single-point (unary) vectors. Its ideal functionality is shown in Figure 2.4 and consists of two phases: the functionality samples field element $\Delta \leftarrow \mathbb{F}$ to P_B for initialization; Then, for P_A 's input multi-point vector $e_u \in \mathbb{F}^{t \cdot 2^h}$, where $t \cdot 2^h = n$, the functionality samples $K[e_u] \leftarrow \mathbb{F}^n$ to P_B and delivers $M[e_u] = K[e_u] + e_u \cdot \Delta \in \mathbb{F}^n$ to P_A . It serves as an important building block in instantiating $\mathcal{F}_{C\text{-VOLE}}$.

Threshold Somewhat Homomorphic Encryption The functionality $\mathcal{F}_{\text{ThSWHE}}$ in Figure 2.5 models an ideal *threshold SWHE* system between two parties. Compared to a standard SWHE scheme, the main difference lies in the decryption procedure. In the threshold setting, neither party can independently decrypt a ciphertext $\llbracket m \rrbracket$. Instead, both parties jointly decrypt $\llbracket m \rrbracket$ such that each party obtains one additive share of m . We realize this using a distributed decryption procedure (e.g., [48]).

Functionality $\mathcal{F}_{\text{MPVOLE}}$

Initialize: Upon receiving (init) from both P_A and P_B , sample $\Delta \leftarrow \mathbb{F}$ to P_B . The functionality stores Δ . If P_B is malicious, receive $\Delta \in \mathbb{F}$ from P_B .

Extend: Upon receiving (extend, $e_u \in \mathbb{F}^{t \cdot 2^h}, h, t$) from P_A where e_u is a regular sparse vector with t non-zero elements and $n = t \cdot 2^h$, and (extend, h, t) from P_B .

1. Samples $K[e_u] \leftarrow \mathbb{F}^n$. If P_B is malicious, receives $K[e_u] \in \mathbb{F}^n$ from adversary.
2. Computes $M[e_u] := K[e_u] + \Delta \cdot e_u \in \mathbb{F}^n$. If P_A is malicious, receives $M[e_u] \in \mathbb{F}^n$ from the adversary, and recomputes $K[e_u] := M[e_u] - \Delta \cdot e_u \in \mathbb{F}^n$.
3. Sends $M[e_u]$ to P_A and $K[e_u]$ to P_B .

Selective failure: In the malicious setting, if P_B is corrupted, the adversary is allowed to make the following selective failure query for each extend call:

1. Wait for the adversary to input t sets $I_1, \dots, I_t \subseteq [0, n)$. Let $\{\alpha_i\} \in [0, n)$ be the index of the non-zero entry of i th single-point block e_u^i .
2. For all $\alpha_i \in I_i$ for $i \in [t]$, send success to P_B and continue. Otherwise, send abort to both parties and abort.

Figure 2.4: The multi-point VOLE functionality.

Bulletin Board. We use a bulletin board functionality that allows the server to post messages that can be read by all clients. This functionality is used for distribution of the server’s public key as well as the server’s encoding. See Figure 2.6. In practice, the server’s public key would be distributed through standard PKI mechanisms, and we envision that the server’s encoding would be distributed through content-distribution networks.

Coin-tossing. The coin-tossing functionality $\mathcal{F}_{\text{Rand}}$ allows the parties to jointly sample a public random value from a specified efficiently sampleable set, as shown in Figure 2.7. The functionality can be securely computed in the random oracle model (see, e.g., [49], [50]).

2-party Computation This functionality $\mathcal{F}_{2\text{PC}}$ in Figure 2.8 defines the ideal execution of two-party computation (2PC). Two parties, P_A and P_B , provide inputs x and y , respectively, and a trusted functionality computes $z = f(x, y)$. In the honest setting, both parties receive the output z . If one party is corrupted, the adversary controlling it receives z and may either send it to the honest party or output $\perp$.

Functionality $\mathcal{F}_{\text{ThSWHE}}$

Public parameters: Security parameter λ , maximum multiplication depth L , message share space $\mathcal{M}$. Let KeyGen and Dec denote the key generation and decryption algorithms of an FHE scheme, respectively.

1. **KeyGen:** Upon receiving (init) from both servers, the functionality invokes $(\text{params}, \text{sk}, \text{pk}, \text{ek}) \leftarrow \text{KeyGen}(1^\lambda, 1^L)$, stores sk , and sends (pk, ek) to both parties.
2. **Decryption:** Upon receiving $(\llbracket m \rrbracket)$ from both parties, the functionality fetches sk , decrypts $m = \text{Dec}(\text{sk}, \llbracket m \rrbracket)$, samples uniform $m_1, m_2 \leftarrow \mathcal{M}$ under the constraint that $m_1 + m_2 = m$, and sends m_1 to one party and m_2 to the other party.

Figure 2.5: Two-party threshold SWHE ideal functionality.

Functionality $\mathcal{F}_{\text{BB}}$

There is a server S and clients $C_1, \dots$

Send: Upon receiving msg from S , send msg to $\mathcal{A}$ and store msg . Ignore subsequent messages from S .

Fetch: Upon receiving fetch from C_j , if msg is stored then send it to C_j .

Figure 2.6: The bulletin board functionality.

Functionality $\mathcal{F}_{\text{Rand}}$

Upon receiving (Rand, S) from both parties, where S is an efficiently sampleable set, sample $r \leftarrow S$, and then output r to all parties.

Figure 2.7: The coin-tossing functionality.

Functionality $\mathcal{F}_{2\text{PC}}$

Participants: Two parties P_A and P_B .

Public parameters: A function $f(\cdot, \cdot)$.

Input Phase: Upon receiving input x from P_A and y from P_B , the functionality records these values. If a party does not provide input, it uses a default value $\perp$.

Computation Phase: Once both inputs are received, compute $z \leftarrow f(x, y)$.

Output Phase: Sends (output, z) to P_A ; P_B receives no output.

Figure 2.8: The two-party computation functionality.

Client-Output Private Set Intersection. Private set intersection (PSI) allows two parties to jointly compute the intersection of their private sets without revealing any additional information (except the sizes of their sets). In Figure 2.9, we describe the ideal functionality of client-output PSI

in the client–server setting, which allows a server to compute intersections with multiple clients, while the client get the intersection. The functionality ensures that the server uses the same set with every client.

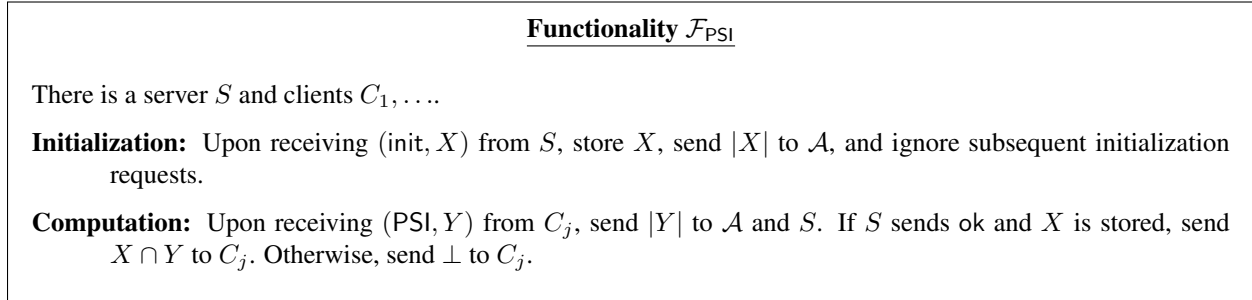


Figure 2.9: The private set intersection functionality.

Server-Output Private Set Intersection. In the client–server setting, we also consider a variant in which a server interacts separately with multiple clients and receives the intersection in each execution. In Figure 2.10, the server acts as the receiver of the intersection, while the clients provide their private sets as inputs. The server initializes the functionality with its private set. Thereafter, whenever a client submits its private set, the functionality computes the intersection with the stored server set and returns the result to the server.

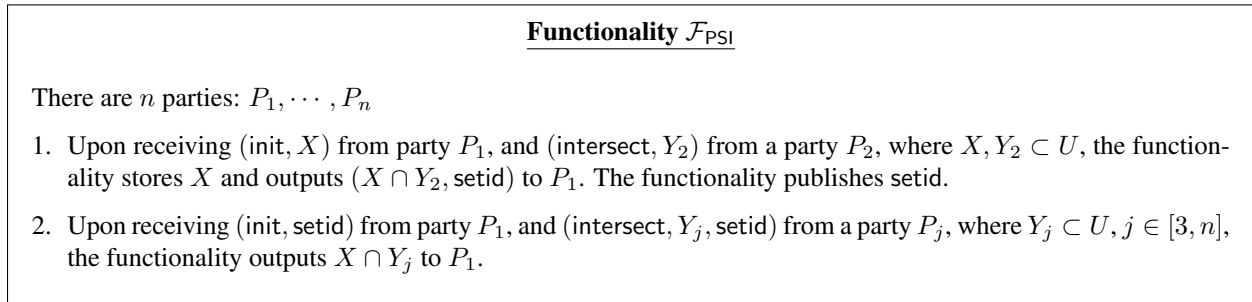


Figure 2.10: The private set intersection functionality.

PJC over Distributed Inputs. This functionality $\mathcal{F}_{\text{PJC}}$ (see Figure 2.11) defines the ideal process for private join and compute over distributed input sets. It involves two servers, Agg and CSP, and w parties $P_1, \dots, P_w$. Each party submits a multiset to the functionality, which adds it to the

designated multiset union. Finally, the functionality computes the frequency vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ of the two multiset unions, applies a randomized function F to obtain an approximation $z \leftarrow F(\mathbf{v}_1, \mathbf{v}_2)$, and sends z to Agg.

Functionality $\mathcal{F}_{\text{PJC}}$

Participants: Two servers Agg, CSP, and w parties $P_1, \dots, P_w$.

Public function: A randomized function F that outputs an inner product approximation of the input vectors.

1. **Initialization.** Upon receiving (Initialization) from Agg and CSP, initialize sets $\mathcal{S}_1 := \emptyset$ and $\mathcal{S}_2 := \emptyset$.
2. **Input.** Upon receiving from P_i input (input, $p, \mathcal{S}$), where $p \in \{1, 2\}$ and $\mathcal{S}$ is a multiset over domain $[N]$, the functionality updates $\mathcal{S}_p := \mathcal{S}_p \uplus \mathcal{S}$ as multi-set union. Send p to Agg.
3. **Output.** Upon receiving (Output) from Agg and CSP, the functionality computes a frequency vector $\mathbf{v}_1$ (resp. $\mathbf{v}_2$) of set $\mathcal{S}_1$ (resp. $\mathcal{S}_2$), computes and sends to Agg an approximation $z \leftarrow F(\mathbf{v}_1, \mathbf{v}_2)$.

Figure 2.11: The ideal functionality for private join and compute over distributed input sets.

CHAPTER 3

Client–Server PSI from Oblivious Verifiable Unpredictable Function

In this chapter, inspired by the password-checkup setting, we introduce a client–server PSI protocol based on an oblivious verifiable unpredictable function (OVUF). We begin by giving an intuition of the main ideas underlying our construction. We then present the protocol and provide a detailed security analysis. Finally, we describe our implementation and evaluate the protocol’s performance experimentally.

Actively secure PSI in the client-server setting. We design an actively secure PSI protocol that is particularly suitable for the multi-client setting. In our protocol the server encodes its set once during an offline phase, and can then repeatedly execute the online phase of the protocol with multiple clients. Thus, the server’s initial encoding is *reusable*, so only needs to be computed once. Moreover, clients do not need to know the server’s encoding when running the online phase, but can retrieve it *asynchronously* even after the online phase is complete. This offers flexibility, potentially allowing the encoding to be distributed via content-distribution or peer-to-peer networks. (See further discussion in Chapter.) Alternately, the client can reduce bandwidth and retrieve only relevant portions of the encoding (after completing the online phase of the protocol) using verifiable private information retrieval [51], [52].

An efficient oblivious VRF. Similar to prior work, our PSI protocol relies on a subprotocol for oblivious pseudorandom function (OPRF) evaluation. To achieve security against a malicious server (as explained further in Chapter) we strengthen this to oblivious evaluation of a *verifiable* pseudorandom function (OVRF). (For technical reasons, it is actually more convenient for us to work with verifiable unpredictable functions, or VUFs.) While OPRFs are well-studied, we are not aware of any prior work constructing (efficient) OVUFs/OVRFs. Some prior works consider veri-

fiable OPRFs; however, efficient constructions [53], [54], [55], [56], [57], [58] are not extractable (thus not applicable here), while extractable constructions are all far from being practical [59], [60], [61], [62]. We also show an efficient OVUF/OVRF protocol based on the (non-oblivious) VRF of Dodis and Yampolskiy [34]. Our protocol relies on techniques for converting secret shares from multiplicative to additive form (aka MtA conversion), something considered by several prior works in other contexts [63], [64], [65]. To further improve efficiency, we rely on an “imperfect” MtA protocol that allows a cheating server to cause a client to output an incorrect result. We show that this suffices in our setting.

Practical efficiency. We implemented our PSI protocol using state-of-the-art building blocks, and our experimental results show that our protocol is highly efficient. For example, at the 128-bit computational / 40-bit statistical security level, for a server holding 10^8 elements and a client holding 10^3 elements, the size of the server’s encoding is 800MB; an execution of the protocol uses 60MB of communication, runs in under 5s in a WAN network with 120 Mbps bandwidth, and costs only 0.017 USD when utilizing network caching infrastructures, a $5\times$ saving compared to a state-of-the-art malicious PSI protocol without consistency guarantee.

3.1. Protocol Intuition

As we discussed in Chapter 1.1.2, the OPRF-based PSI schemes suffices to achieve security against a malicious client, but not a malicious server. Because malicious servers can use different encodings for different clients, they can easily distinguish between clients with different inputs. We address this issue by using a *verifiable* random function (VRF) [66]. A VRF is associated with both a private key sk and a public key pk ; informally, $F_{sk}(\cdot)$ should look random even given pk , but a VRF has the extra property that $x' = F_{sk}(x)$ can be verified as correct given x and pk by running a verification procedure $\text{Vrfy}(pk, x, x')$. We modify the protocol given above by having the server publish pk , and setting $\text{En}(sk; x_i) = H(x, F_{sk}(x))$ for H a hash function modeled as a random oracle. (This means we now need a sub-protocol for oblivious evaluation of a VRF, which

we present in the following chapter.)

To see how this allows for extraction of the server’s input, consider again a single encoded value ex published by the server. The simulator can look for a corresponding H -query $H(x, x')$ with output ex ; if a unique such query exists then ex can only possibly correspond to x . (If there is no H -query with output ex , then the simulator knows that ex does not correspond to any element.) Crucially, the simulator can then check whether ex is indeed a (correct) encoding of x by checking whether $\text{Vrfy}(\text{pk}, x, x') = 1$. We remark that using a VRF also allows clients to verify that their encoding EY is computed correctly during the online phase of the protocol, something that is also critical for security against a malicious server. Finally, we observe that using a VRF is overkill, and it suffices to rely on a verifiable *unpredictable* function (VUF); we thus construct an oblivious VUF (OVUF) sub-protocol in the construction.¹

Constructing an OVUF. Our starting point is the VUF of Dodis and Yampolskiy [34] based on a bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$.² Let g be a generator of $\mathbb{G}$. In this VUF, the server’s public key is $\text{pk} = g^{\text{sk}} \in \mathbb{G}$, and evaluation is defined as $F_{\text{sk}}(x) = g^{1/(\text{sk}+x)}$. Verification is done by checking if $e(F_{\text{sk}}(x), \text{pk} \cdot g^x) = e(g, g)$.

We now sketch a protocol for oblivious evaluation of this function, run between a server S holding sk and a client C holding input y . At a high level, our protocol works as follows:

1. The parties choose random values ϕ_1 and ϕ_2 , respectively, viewed as an additive sharing of a random value.
2. The parties run a multiplicative-to-additive share-conversion protocol, where S uses sk and C uses ϕ_2 ; as a result, S and C obtain A_1 and A_2 , respectively, such that $A_2 + A_1 = \phi_2 \cdot \text{sk}$.
3. Similarly, the parties obtain B_1 and B_2 such that $B_2 + B_1 = \phi_1 \cdot y$.

¹It is easy to turn a VUF F into a VRF F' in the random-oracle model by defining $F'_{\text{sk}}(x) = H(F_{\text{sk}}(x))$. Nevertheless, relying on a VUF provides a cleaner abstraction for our protocol. OVUFs may also be easier to construct than OVRFs.

²For efficiency, we use Type-III pairings where the groups in the domain are different; here, we describe things in the Type-I setting for simplicity.

4. S sends $\phi_1 \cdot \text{sk} + A_1 + B_1$ and C sends $\phi_2 \cdot y + A_2 + B_2$. The parties then add these values to obtain $v = (\text{sk} + y)(\phi_1 + \phi_2)$.
5. S sends $g^{\phi_1/v}$ to C , who computes $g^{\phi_1/v} \cdot g^{\phi_2/v} = g^{1/(\text{sk}+y)}$.

Note that C can verify the final result using the verifiability property of the VUF and the server's public key.

The bottleneck in the above is the subroutine for multiplicative-to-additive (MtA) share conversion. Actively secure MtA protocols have been a key building block in the context of threshold ECDSA, and there have been proposals for constructing them using oblivious transfer (OT) [67], [68], Paillier encryption [69], [70], [71], and Castagnos-Laguillaumie encryption [65]; see Xue et al. [64] for a more detailed survey. In this chapter, we focus on constructions from OT as they are the most computationally efficient.

We adapt the MtA approach used by Doerner et al. [67] that can be viewed as a malicious version of an idea by Gilboa [72]. For two parties with a and b as input, the high-level idea is to use OT to generate additive secret sharings of $a \cdot b_i$, where b_i is the i th bit of b . The two parties can then compute an additive secret sharing of $a \cdot b$ as a linear combination of the shares of the $\{ab_i\}$. To achieve security against malicious behavior, Doerner et al. made two changes: (1) each OT will select two sets of values, where the second set of values is used solely for checking correctness of the output; (2) to prevent selective-failure attacks, they encode the bit b_i as a longer string of choice bits instead of using just b_i itself. These changes lead to an overhead of $4\text{--}5\times$ in communication as compared to the underlying semi-honest protocol.

We observe that since the final output in our application can be verified anyway, we can save half the communication by not doing checking in the MtA protocol itself. Removing the check in the MtA protocol complicates the proof of security. In particular, we are unable to define an appropriate functionality that our “imperfect” MtA sub-protocol realizes, and so instead we prove security of the entire OVUF directly.

It is still possible for a malicious server to cheat by using an incorrect value of sk in the protocol. This can even lead to a concrete attack: to determine whether the client's input is some value y ,

a malicious server can use $sk^* = -y$ in step 2 of the protocol and then see whether $v = 0$ in step 4. To ensure that this does not happen, we add an extra verification step after the second step. Essentially, we want to verify that the server holds A_1 such that $A_1 + A_2 = \phi_2 \cdot sk$, where A_2, ϕ_2 are known to the client and $sk = \log_g pk$. To do this, we have the server send g^{A_1} , and the client checks if this is equal to $pk^{\phi_2} \cdot g^{-A_2}$. (When the server is honest, this does not reveal anything to the client that it did not already know; on the other hand, if the client is honest then A_2 is uniform and so a cheating server will be caught with overwhelming probability.) Using hashing, this check can be batched when evaluating the VUF at multiple points; thus, the check incurs negligible (amortized) communication and only a few exponentiations.

3.2. Protocol and Security Analysis

This section presents our PSI protocol and its security analysis. The protocol UC-realizes the client-output PSI functionality (Figure 2.9) from Section 2.3, using OVUF as the main building block. We define the OVUF functionality below and give an efficient instantiation in Section 3.2.2.3.

Oblivious verifiable unpredictable function. One natural way to formalize an oblivious VRF is via a functionality that internally generates a random function F ; when queried by a client with input (eval, x) , the functionality returns $F(x)$ to the client if the server approves. Moreover, the functionality should allow any party to query (verify, x, y) to learn whether $y = F(x)$ (without notifying the server or requiring its approval). There are at least two problems with such an approach. First, it would need to be modified to handle a malicious server who may not choose a uniform key. While such a modification is possible, it complicates things. Second, it seems difficult to model an *unpredictable* (rather than *random*) function using this type of approach.

We therefore choose to model the OVUF functionality as a secure evaluation of a concrete VUF, as shown in Figure 3.1. We allow the client to request oblivious evaluation at multiple points (“batch evaluation”), as this can allow for better efficiency.

<u>Functionality $\mathcal{F}_{\text{OVUF}}$</u>
There is a server S and clients $C_1, \dots$. Let $(\text{Gen}, F, \text{Vrfy})$ be a VUF. Initialization: Upon receiving (init, pk) from S, store pk, send pk to $\mathcal{A}$, and ignore subsequent initialization requests. The queries below are ignored if pk is not stored. Key query: Upon receiving fetch from C_j, send pk to C_j. Evaluation: Upon receiving $(\text{eval}, (y_1, \dots, y_n))$ from C_j, send n to S and $\mathcal{A}$. When S responds with sk, check the validity of (sk, pk). If (sk, pk) is invalid, send $\perp$ to C_j. Otherwise, send $(F_{\text{sk}}(y_1), \dots, F_{\text{sk}}(y_n))$ to C_j.

Figure 3.1: The OVUF functionality.

3.2.1. PSI from OVUF

We now present the PSI protocol obtained from the OVUF primitive in Figure 3.2 and give its security proof in Theorem 4.

<u>Protocol Π_{PSI}</u>
The server S holds X and each client C_j holds Y_j. $H : \mathbb{Z}_q \times \mathbb{G} \rightarrow \{0, 1\}^\sigma$ is a hash function. Initialization: 1. S runs $(\text{sk}, \text{pk}) \leftarrow \text{Gen}$ and sends (init, pk) to $\mathcal{F}_{\text{OVUF}}$. 2. For each $x_i \in X$, the server computes $ex_i = H(x_i, F_{\text{sk}}(x_i))$. It then sends $EX = \{ex_i\}$ to $\mathcal{F}_{\text{BB}}$. Compute intersection: 1. C_j sends $(\text{eval}, y_1, \dots, y_n)$ to $\mathcal{F}_{\text{OVUF}}$. Upon receiving n from $\mathcal{F}_{\text{OVUF}}$, S sends sk to $\mathcal{F}_{\text{OVUF}}$. Then $\mathcal{F}_{\text{OVUF}}$ sends $(F_{\text{sk}}(y_1), \dots, F_{\text{sk}}(y_n))$ to C_j. (If $\mathcal{F}_{\text{OVUF}}$ sends $\perp$, then C_j aborts.) 2. C_j computes $ey_i = H(y_i, F_{\text{sk}}(y_i))$ and lets $EY = \{ey_i\}$. 3. C_j sends fetch to $\mathcal{F}_{\text{BB}}$, and receives EX in return. It then outputs $\{y_i : ey_i \in EX \cap EY\}$.

Figure 3.2: PSI protocol in the $(\mathcal{F}_{\text{OVUF}}, \mathcal{F}_{\text{BB}})$ -hybrid model.

Theorem 4. Assume the VUF used by $\mathcal{F}_{\text{OVUF}}$ is secure. If H is modeled as a random oracle, then Π_{PSI} UC-realizes $\mathcal{F}_{\text{PSI}}$ in the $(\mathcal{F}_{\text{OVUF}}, \mathcal{F}_{\text{BB}})$ -hybrid model.

Proof. Let $\mathcal{A}$ be a PPT adversary that may corrupt the server and any number of clients. We construct a simulator $\mathcal{S}$ with access to functionality $\mathcal{F}_{\text{PSI}}$ that runs $\mathcal{A}$ as a subroutine. Note that

there is nothing to simulate if a corrupted server interacts with a corrupted client. When an honest server interacts with a corrupted client, the only communication observed by the adversary is pk and EX ; thus, that case is covered in the same way as in the case of an honest server interacting with a corrupted client.

Corrupted server with some honest clients. $\mathcal{S}$ runs $\mathcal{A}$, simulating H by returning random responses to $\mathcal{A}$'s queries. Then:

1. Let (init, pk) be the message $\mathcal{A}$ sends to $\mathcal{F}_{\text{OVUF}}$, and let $EX = \{ex_i\}$ be the message $\mathcal{A}$ sends to $\mathcal{F}_{\text{BB}}$.
2. Initialize $X = \emptyset$. Then for each $ex \in EX$, do:
 - (a) If $\mathcal{A}$ did not make any H -query with output ex , do nothing.
 - (b) If $\mathcal{A}$ made an H -query with output ex , let $H(x, x')$ be the first such query. Add x to X iff $\text{Vrfy}(\text{pk}, x, x') = 1$.

Send X to $\mathcal{F}_{\text{PSI}}$ on behalf of S .

3. Upon receiving n from $\mathcal{F}_{\text{PSI}}$, send n to $\mathcal{A}$ on behalf of $\mathcal{F}_{\text{OVUF}}$. If $\mathcal{A}$ does not respond, or responds with sk for which (sk, pk) is invalid, send **abort** to $\mathcal{F}_{\text{PSI}}$. Otherwise, send **ok** to $\mathcal{F}_{\text{PSI}}$.

It is not hard to see that the simulation is statistically close to an execution of Π_{PSI} in the $(\mathcal{F}_{\text{OVUF}}, \mathcal{F}_{\text{BB}})$ -hybrid world. The uniqueness of the VUF ensure that the simulator does not include wrong elements: if the adversary could find values (x, x') that pass the verification but not a VUF input-output pair, the extraction would include such an element incorrectly.

Corrupted clients with an honest server. $\mathcal{S}$ runs $\mathcal{A}$, simulating H by returning random responses to $\mathcal{A}$'s queries. Then:

1. Run $(\text{sk}, \text{pk}) \leftarrow \text{Gen}$. If a corrupted client queries **fetch** to $\mathcal{F}_{\text{OVUF}}$, send pk in response.
2. $\mathcal{S}$ receives n from $\mathcal{F}_{\text{PSI}}$. It chooses $ex_1, \dots, ex_n \leftarrow \{0, 1\}^\sigma$ and sets $EX = \{ex_i\}$. If any client (corrupted or not) queries $\mathcal{F}_{\text{BB}}$, it sends EX in response. $\mathcal{S}$ also maintains a table T indexed by $[n]$, initially empty.

3. Whenever $\mathcal{A}$ sends $(\text{eval}, y_1, \dots, y_m)$ to $\mathcal{F}_{\text{OVUF}}$ on behalf of a corrupted client, do
 - (a) Send $Y = \{y_i\}$ to $\mathcal{F}_{\text{PSI}}$, and receive in return a set $Z \subseteq Y$. Send $(F_{\text{sk}}(y_1), \dots, F_{\text{sk}}(y_m))$ to $\mathcal{A}$.
 - (b) For each $z \in Z$ do: If there is an i with $T[i] = z$, do nothing. Otherwise, choose a uniform empty entry $T[i]$, set $T[i] = z$, and program H so that $H(z, F_{\text{sk}}(z)) = ex_i$.

It is again not hard to see that the simulation is statistically close to an execution of Π_{PSI} in the $(\mathcal{F}_{\text{OVUF}}, \mathcal{F}_{\text{BB}})$ -hybrid world. In particular, the simulation relies on the fact that corrupted clients do not query $(z, F_{\text{sk}}(z))$ to RO ahead of time, which reduces to the unpredictability of the underlying VUF. $\square$

3.2.1.1. Distributing the Server Encoding

Here we discuss several solutions that could be used in practice to distribute the server encoding.

Network caching. Network caching technologies like content distribution network (CDN) are good at distributing content cheaply and quickly. This is the standard technique to distribute common website and streaming services. Our service encoding can take advantage of CDN networks since the server encoding is identical for all clients. Note that prior works on malicious PSI cannot take advantage of CDN since the communication with each client is different.

Verifiable private information retrieval. One can also use verifiable PIR [52], [73] to allow the clients getting only a subset of encodings relevant to their own PSI. Unlike normal PIR, verifiable PIR publishes a digest of the data, which ensures that anyone with the digest can verify that the PIR results are consistent with a global database, something needed to prevent attacks from a corrupted server. However, state-of-the-art verifiable PIR has a digest size of around 600MB for a database of 800MB [52] and thus the current savings are small. With more advances in their efficiency, we believe this solution could be highly valuable.

Other solutions. There are other potential solutions with some trade offs between security and efficiency. First of all, one could directly fetch the needed encodings through a TOR network to

hide their identity, which requires assumptions of trusting TOR. Bucketization is another solution that provides better efficiency with reduced privacy. In detail, one can use a hash function to partition all encodings into buckets and reveal which buckets the clients are looking. Indeed, this solution has been used by Google and Cloudflare for credential checking, but there are also demonstration of attacks for various bucketization techniques [74].

3.2.2. Oblivious VUF

Now we present an OVUF protocol for the Dodis-Yampolskiy VUF, with security against malicious adversaries. Our protocol works in the $(\mathcal{F}_{\text{BB}}, \mathcal{F}_{\text{COT}})$ -hybrid model with a sub-protocol named imperfect multiplicative to additive shares Π_{MtA} . In Chapter 3.2.2.1, we review a randomized encoding scheme. Then, in Chapter 3.2.2.2, we introduce the sub-protocol Π_{MtA} , which leverages the encoding scheme. The OVUF protocol, described in Chapter 3.2.2.3, is constructed based on Π_{MtA} . Then, we give a complexity analysis of the proposed OVUF protocol in Chapter 3.2.2.4. We leave the discussion of further optimization in Appendix A.1.

3.2.2.1. Encoding for Coalesced Multiplication

We provide a brief recap of the randomized encoding scheme described by Doerner et al. [67]. However, we prove some slightly different properties of the encoding where we also take the randomness of the encoding vector $\mathbf{g}^{\text{R}}$. This is valid in our protocol because, as shown in Figure 3.3, we sample $\mathbf{g}^{\text{R}}$ only after the adversary chooses where to cheat.

Single encoding. Define coefficient vector $\mathbf{g} = \mathbf{g}^{\text{G}} \parallel \mathbf{g}^{\text{R}}$, where $\mathbf{g}^{\text{G}} \in \mathbb{Z}_q^{\log q}$, $\mathbf{g}_i^{\text{G}} = 2^{i-1}$, and $\mathbf{g}^{\text{R}} \in \mathbb{Z}_q^{\log q + 2\kappa}$.

$\text{Encode}(\mathbf{g}^{\text{R}} \in \mathbb{Z}_q^{\log q + 2\kappa}, \beta \in \mathbb{Z}_q)$

1. Sample $\gamma \leftarrow \{0, 1\}^{\log q + 2\kappa}$.
 2. Output $\text{Bits}(\beta - \langle \mathbf{g}^{\text{R}}, \gamma \rangle) \parallel \gamma$.
-

Lemma 2. Given uniform $\gamma \leftarrow \{0, 1\}^{\log q + 2\kappa}$ and $\mathbf{g}^R \leftarrow \mathbb{Z}_q^{\log q + 2\kappa}$, $h_{\mathbf{g}^R}(\gamma) := \langle \mathbf{g}^R, \gamma \rangle$ is statistically close to uniform distribution with a statistical distance of at most $2^{-\kappa}$.

Proof. Let $\varepsilon = 2^{-\kappa}$, then we have $\log q = \log q + 2\kappa - 2\log(\frac{1}{\varepsilon})$. We can define a set of functions $\mathcal{H}$, such that for each $h \in \mathcal{H}$, it is the form of $\mathbb{Z}_q^{\log q + 2\kappa} \times \{0, 1\}^{\log q + 2\kappa} \rightarrow \mathbb{Z}_q$. Each function, parameterized by $\mathbf{g}^R$ as $h_{\mathbf{g}^R}(\gamma) := \langle \mathbf{g}^R, \gamma \rangle$ is a 2-universal hash function [63] with output bit length $\log q$. For input $\gamma \leftarrow \{0, 1\}^{\log q + 2\kappa}$, its entropy is $H_\infty(\gamma) = \log q + 2\kappa$. Thus, we have

$$\log q = H_\infty(\gamma) - 2\log\left(\frac{1}{\varepsilon}\right) \quad (3.1)$$

Equation 3.1 satisfies the Leftover Hash Lemma, that the output bit length of the 2-universal hash function equals the entropy of input minus $2\log(\frac{1}{\varepsilon})$. Thus, for any $\mathbf{g}^R$ uniformly distributed over $\mathbb{Z}_q^{\log q + 2\kappa}$ and independent of γ , we have

$$\sigma[(h_{\mathbf{g}^R}(\gamma), \gamma), (U, \gamma)] \leq \varepsilon$$

where U is uniform distributed over $\{0, 1\}^{\log q}$ and independent of $\mathbf{g}^R$. Thus, the statistical distance of $h_{\mathbf{g}^R}(\gamma)$ and U is at most $2^{-\kappa}$. $\square$

Batch encoding. When encoding more than one element, it is possible to perform better than encoding each element independently.

BatchEncode($\mathbf{g}^R \in \mathbb{Z}_q^{\log q + 2\kappa}, \{\beta^1, \dots, \beta^n\} \in \mathbb{Z}_q^n$)

1. Sample $\gamma^1 \leftarrow \{0, 1\}^{\log q}, \dots, \gamma^n \leftarrow \{0, 1\}^{\log q}, \gamma^{n+1} \leftarrow \{0, 1\}^{2\kappa}$
 2. Output $Bits(\beta^1 - \langle \mathbf{g}^R, \gamma^1 || \gamma^{n+1} \rangle) || \gamma^1 || \dots || Bits(\beta^n - \langle \mathbf{g}^R, \gamma^n || \gamma^{n+1} \rangle) || \gamma^n || \gamma^{n+1}$.
-

Lemma 3. Given uniform $\gamma = \gamma^1 || \dots || \gamma^{n+1} \leftarrow \{0, 1\}^{n \log q + 2\kappa}$ and $\mathbf{g}^R \leftarrow \mathbb{Z}_q^{\log q + 2\kappa}$, $h_{\mathbf{g}^R}(\gamma) := \langle \mathbf{g}^R, \gamma^1 || \gamma^{n+1} \rangle || \dots || \langle \mathbf{g}^R, \gamma^n || \gamma^{n+1} \rangle$ is within statistical distance $2^{-\kappa}$ of uniform.

Proof. Let $\varepsilon = 2^{-\kappa}$, then we have $n \log q = n \log q + 2\kappa - 2\log(\frac{1}{\varepsilon})$. We can define a set of functions $\mathcal{H}$, such that for each $h \in \mathcal{H}$, it is the form of $\mathbb{Z}_q^{\log q + 2\kappa} \times \{0, 1\}^{n \log q + 2\kappa} \rightarrow \{0, 1\}^{\log q}$. Each

function, parameterized by $\mathbf{g}^R$ as $h_{\mathbf{g}^R}(\gamma) := \langle \mathbf{g}^R, \gamma^1 || \gamma^{n+1} \rangle || \dots || \langle \mathbf{g}^R, \gamma^n || \gamma^{n+1} \rangle$ is a 2-universal hash function [63] with output bit length $n \log q$. For input $\gamma \leftarrow \{0, 1\}^{n \log q + 2\kappa}$, it has information entropy $H_\infty(\gamma) = n \log q + 2\kappa$. Thus,

$$n \log q = H_\infty(\gamma) - 2 \log\left(\frac{1}{\varepsilon}\right) \quad (3.2)$$

Equation 3.2 satisfies the Leftover Hash Lemma, that the output bit length of the 2-universal hash function equals the entropy of input minus $2 \log(\frac{1}{\varepsilon})$. Thus, for any $\mathbf{g}^R$ uniformly distributed over $\mathbb{Z}_q^{\log q + 2\kappa}$ and independent of γ , we have

$$\sigma[(h_{\mathbf{g}^R}(\gamma), \gamma), (U, \gamma)] \leq \varepsilon$$

where U is uniform distributed over $\{0, 1\}^{n \log q}$ and independent of $\mathbf{g}^R$. Thus, the statistical distance of $h_{\mathbf{g}^R}(\gamma)$ and U is at most ε , which equals to $2^{-\kappa}$. $\square$

3.2.2.2. Imperfect MtA Protocol

The imperfect multiplicative to additive (MtA) shares protocol transforms multiplicative shares to additive shares. It is imperfect because a malicious sender can execute attacks that lead to incorrect additive secret shares, depending on the receiver's input. This protocol is specially designed for our efficient DY-based oblivious VUF protocol because it does not directly instantiate the MtA functionality due to the lack of correctness guarantee. Therefore, we do not model it as a functionality. The correctness will be checked for free as part of the OVUF protocol.

We use oblivious transfer based constructions to achieve this MtA. For the semi-honest version, given value $a \in \mathbb{Z}_q$ on the sender side and $b \in \mathbb{Z}_q$ on the receiver side, the sender execute $\log q$ iterations of $\mathcal{F}_{\text{COT}}$ with a as input in each i th iteration, while the receiver inputs b_i , representing the i th bit of the binary representation of b . The procedure and its correctness are detailed below:

1. For $i \in [\log q]$, the receiver inputs b_i to $\mathcal{F}_{\text{COT}}$, while the sender inputs a . $\mathcal{F}_{\text{COT}}$ sends q_i to receiver and p_i to sender, such that $q_i + p_i = a \cdot b_i$.

2. Define $d = \sum_{i \in [\log q]} 2^{i-1} \mathbf{q}_i$, $c = \sum_{i \in [\log q]} 2^{i-1} \mathbf{p}_i$. Then

$$\begin{aligned} d + c &= \sum_{i \in [\log q]} 2^{i-1} \mathbf{q}_i + \sum_{i \in [\log q]} 2^{i-1} \mathbf{p}_i \\ &= a \sum_{i \in [\log q]} 2^{i-1} \mathbf{b}_i = a \cdot b. \end{aligned}$$

However, a malicious sender could potentially execute attacks to the semi-honest protocol above. Specifically, it samples an error vector $\mathbf{e} \in \mathbb{Z}_q^n$ and inputs $a + \mathbf{e}_i$ to $\mathcal{F}_{\text{COT}}$ in its i th iteration. It results $d + c = a \cdot b + \sum_{i \in [\log q]} 2^{i-1} \mathbf{e}_i \mathbf{b}_i$. Given $\mathbf{e}_i$, the correctness of MtA transformation depends on the receiver's input b . Specifically, the transformation is correct when $\sum_{i \in [\log q]} 2^{i-1} \mathbf{e}_i \mathbf{b}_i = 0$. Prior works incorporate consistency checks and encoding to resist such malicious behaviors [67]. The consistency check, for input a in different iterations, leaks information. Encoding is involved to further protect privacy. In our construction, MtA is used in OVUF protocol in Chapter 3.2.2.3. Since the verifiability of OVUF implicitly gives the same property as a consistency check, we only incorporate the encoding algorithm in [67] to give an imperfect MtA protocol. To enhance efficiency, we give a batch version in Figure 3.3. In this scenario, two parties hold collections of n elements, denoted as $\mathbf{a} \in \mathbb{Z}_q^n$ and $\mathbf{b} \in \mathbb{Z}_q^n$, respectively. There is a receiver that employs $\text{BatchEncode}(\mathbf{g}^R, \mathbf{b})$ algorithm to encode each element of its input into a batched binary representation. $\mathbf{g}^R \in \mathbb{Z}_q^{\log q + 2s}$ is randomly chosen by the receiver and sent to the sender after executing $\mathcal{F}_{\text{COT}}$. To run $\mathcal{F}_{\text{COT}}$ correctly in each iteration, the sender inputs $\mathbf{a}_i$ and the receiver inputs corresponding $\mathbf{b}_i$ in its batch encoded bit representation form.

We show correctness of Figure 3.3 in its single encoded version:

1. Define $\mathbf{w} = \text{Encode}(\mathbf{g}^R, b) \in \{0, 1\}^{2 \log q + 2\kappa}$, which is the encoding of b . For $i \in [t + 2\kappa]$, $t = 2 \log q$, the receiver inputs $\mathbf{w}_i$ to $\mathcal{F}_{\text{COT}}$, while the sender inputs a . $\mathcal{F}_{\text{COT}}$ sends $\mathbf{q}_i$ to receiver and $\mathbf{p}_i$ to sender, such that $\mathbf{q}_i + \mathbf{p}_i = \mathbf{w}_i \cdot a$.
2. For $\mathbf{g} = \mathbf{g}^G || \mathbf{g}^R$, define $d = \sum_{i \in [t]} \mathbf{g}_i \mathbf{q}_i + \sum_{i \in [2\kappa]} \mathbf{g}_{t+i} \mathbf{q}_{t+i}$ and $c = \sum_{i \in [t]} \mathbf{g}_i \mathbf{p}_i + \sum_{i \in [2\kappa]} \mathbf{g}_{t+i} \mathbf{p}_{t+i}$.

We have

$$\begin{aligned}
d + c &= \sum_{i \in [t]} \mathbf{g}_i \mathbf{q}_i + \sum_{i \in [2\kappa]} \mathbf{g}_{t+i} \mathbf{q}_{t+i} + \sum_{i \in [t]} \mathbf{g}_i \mathbf{p}_i + \sum_{i \in [2\kappa]} \mathbf{g}_{t+i} \mathbf{p}_{t+i} \\
&= \sum_{i \in [t]} \mathbf{g}_i (\mathbf{q}_i + \mathbf{p}_i) + \sum_{i \in [2\kappa]} \mathbf{g}_{t+i} (\mathbf{q}_{t+i} + \mathbf{p}_{t+i}) \\
&= a \left(\sum_{i \in [t]} \mathbf{g}_i \mathbf{w}_i + \sum_{i \in [2\kappa]} \mathbf{g}_{t+i} \mathbf{w}_{t+i} \right) \\
&= a \cdot b
\end{aligned}$$

For a malicious sender executing the attacks described above, the relation will be resulted as $d + c = a \cdot b + \sum_{i \in [t]} \mathbf{g}_i \mathbf{e}_i \mathbf{w}_i + \sum_{i \in [2\kappa]} \mathbf{g}_{t+i} \mathbf{e}_{t+i} \mathbf{w}_{t+i}$ with respect to the value of $\mathbf{w} = \text{Encode}(\mathbf{g}^R, b) \in \mathbb{Z}_q^{t+2\kappa}$. For $\mathbf{w} = \text{BatchEncode}(\mathbf{g}^R, \mathbf{b}) \in \mathbb{Z}_q^{nt+2\kappa}$, malicious behavior of sender will result in $\mathbf{d}_i + \mathbf{c}_i = \mathbf{a}_i \cdot \mathbf{b}_i + \mathbf{f}_i$, where

$$\mathbf{f}_i := \sum_{j \in [t]} \mathbf{g}_j \mathbf{w}_{(i-1)t+j} \mathbf{e}_{(i-1)t+j} + \sum_{k \in [2\kappa]} \mathbf{g}_{t+k} \mathbf{w}_{nt+k} \mathbf{e}_{nt+(k-1)n+i} \quad (3.3)$$

We will show how to catch this incorrectness in Chapter 3.2.2.3 below with respect to the detailed OVUF protocol.

<u>Protocol Π_{MtA}</u>
Inputs: P_0 holds $\mathbf{a} \in \mathbb{Z}_q^n$. P_1 holds $\mathbf{b} \in \mathbb{Z}_q^n$. Protocol: 1. P_1 samples $\mathbf{g}^R \leftarrow \mathbb{Z}_q^{\log q + 2\kappa}$. P_1 encodes $\mathbf{b}$ by computing $\mathbf{w} := \text{BatchEncode}(\mathbf{g}^R, \mathbf{b}) \in \{0, 1\}^{nt+2\kappa}$, where $t = 2 \log q$. 2. For $i \in [n], j \in [t]$, P_1 inputs $\mathbf{w}_{(i-1)t+j}$ to $\mathcal{F}_{\text{COT}}$. P_0 inputs $\mathbf{a}_i \in \mathbb{F}_q$ to $\mathcal{F}_{\text{COT}}$. P_0 receives $\mathbf{p}_{i,j} \in \mathbb{F}_q$ from $\mathcal{F}_{\text{COT}}$. P_1 receives $\mathbf{q}_{i,j} \in \mathbb{F}_q$ from $\mathcal{F}_{\text{COT}}$. 3. For $k \in [2\kappa]$, P_1 inputs $\mathbf{w}_{nt+k}$ to $\mathcal{F}_{\text{COT}}$. P_0 inputs $\mathbf{a} \in \mathbb{F}_q^n$ to $\mathcal{F}_{\text{COT}}$. P_0 receives $\{\mathbf{p}'_{1,k}, \dots, \mathbf{p}'_{n,k}\} \in \mathbb{F}_q^n$ from $\mathcal{F}_{\text{COT}}$. P_1 receives $\{\mathbf{q}'_{1,k}, \dots, \mathbf{q}'_{n,k}\} \in \mathbb{F}_q^n$ from $\mathcal{F}_{\text{COT}}$. 4. P_1 sends $\mathbf{g}^R$ to P_0. 5. For $j \in [t], k \in [2\kappa], i \in [n]$, P_0 computes $\mathbf{c}_i = \sum_{j \in [t]} \mathbf{g}_j \cdot \mathbf{p}_{i,j} + \sum_{k \in [2\kappa]} \mathbf{g}_{t+k} \cdot \mathbf{p}'_{i,k}$ and P_1 computes $\mathbf{d}_i = \sum_{j \in [t]} \mathbf{g}_j \cdot \mathbf{q}_{i,j} + \sum_{k \in [2\kappa]} \mathbf{g}_{t+k} \cdot \mathbf{q}'_{i,k}$ such that $\mathbf{d}_i + \mathbf{c}_i = \mathbf{a}_i \cdot \mathbf{b}_i$.

Figure 3.3: The MtA protocol in the $\mathcal{F}_{\text{COT}}$ -hybrid model.

3.2.2.3. An Oblivious VUF from Imperfect MtA

Based on the imperfect MtA protocol from Chapter 3.2.2.2, we construct an OVUF protocol as follows.

1. The server first uses pk to initiate the bulletin board $\mathcal{F}_{BB}$. Then, upon receiving pk from $\mathcal{F}_{BB}$, the client checks whether $g^{-y_i} = pk$ for each y_i in its input set $(y_1, \dots, y_n)$. If so, the client replaces these y_i values with random values to avoid corner cases in the protocol.
2. Given input value sk on the server side and input vector $(y_1, \dots, y_n) \in \mathbb{Z}_q^n$ on the client side, both parties uniformly choose random vectors $\phi_1 \in \mathbb{Z}_q^n$ and $\phi_2 \in \mathbb{Z}_q^n$ respectively.
3. The inputs and random vectors are specifically ordered as $(sk, \phi_1^i) \in \mathbb{Z}_q^2, i \in [n]$ and $(\phi_2^i, y_i) \in \mathbb{Z}_q^2, i \in [n]$, which serves as input vector for Π_{MtA} in its i th iteration. By running Π_{MtA} on both sides for n times, both parties obtain additive secret shares A_1^i and B_1^i of $sk \cdot \phi_2^i$ and additive secret shares A_2^i and B_2^i of $\phi_1^i \cdot y_i$.
4. Then, the server raise g to A_1^i for each $i \in [n]$, where A_1^i is the secret share of $sk \cdot \phi_2^i$, and apply a hash function to these values. The server sends the hash result to the client, allowing it to check whether sk used by the server in each iteration is consistent with the pk initialized on $\mathcal{F}_{BB}$.
5. Then, both parties are able to locally compute $sk \cdot \phi_1^i + A_1^i + B_1^i$ and $\phi_2^i \cdot y_i + A_2^i + B_2^i$, respectively. The results are regarded as secret shares of $v_i = (\phi_1^i + \phi_2^i)(sk + y_i)$. Both parties exchanges the results to recover v_i .
6. Both parties are able to compute $g^{\phi_1^i/v_i}$ and $g^{\phi_2^i/v_i}$, respectively. Given $g^{\phi_1^i/v_i}$, the client computes $F_{sk}(y_i) = g^{\phi_1^i/v_i} \cdot g^{\phi_2^i/v_i}$ and verifies correctness of the protocol using the fetched pk .

The detailed scheme is shown in Figure 3.4. Its correctness can be directly verified. For security, we assume the client acts as a receiver in the execution of $\mathcal{F}_{COT}$ in sub-protocol Π_{MtA} , while the server acts as a sender. A malicious client might send the wrong y_i or u_i to the server.

Incorrect y_i can be extracted by $\mathcal{S}$ given g^R from the client. Incorrect u_i leads to abort with all but negligible probability, which can be simulated by $\mathcal{S}$ constructing message h_i to manipulate abort probability. A malicious server could execute selective failure attack in Π_{OVUF} and bias the secret shares of v_i to be $u_i + m_i = \text{diff}_i + (\phi_1^i + \phi_2^i)(\text{sk} + y_i) = \text{diff}_i + v_i$. diff_i resulted from the incorrectness stated in Chapter 3.2.2.2 that $\text{diff}_i = f_1^i + f_2^i$. f_1^i resulted from incorrect $\phi_1^i \cdot y_i$ and f_2^i resulted from incorrect $\text{sk} \cdot \phi_2^i$. In the server's perspective, g^R is received after the selective failure attack has been executed. For any element g_i^R uniformly distributed over $\mathbb{Z}_q$, diff_i is uniformly distributed over $\mathbb{Z}_q$. If the server sends m_i and h_i honestly, the verification of $F_{\text{sk}}(y_i)$ passes if and only if $\text{diff}_i = 0$, which is with negligible probability. If not, the verification of $F_{\text{sk}}(y_i)$ passes if and only if diff equals a specific number that results in correct $F_{\text{sk}}(y_i)$, which is negligible either. Thus, the server's malicious behavior can be simulated by $\mathcal{S}$ with all but negligible abort probability. The detailed proof of the security of the proposed Π_{OVUF} with sub-protocol Π_{MtA} in the hybrid of $(\mathcal{F}_{\text{BB}}, \mathcal{F}_{\text{COT}})$ is shown in Theorem 2.

<u>Protocol Π_{OVUF}</u>
Inputs and parameters: Hash function H modeled as RO. Client C_j holds $(y_1, \dots, y_n) \in \mathbb{Z}_q^n$. Initialization: S chooses $\text{sk} \in \mathbb{Z}_q$, sets $\text{pk} = g^{\text{sk}}$, and sends pk to $\mathcal{F}_{\text{BB}}$. Key query: Client C_j sends fetch to $\mathcal{F}_{\text{BB}}$ and receives pk. Evaluation: 1. C_j checks if $g^{-y_i} = \text{pk}$ for each $i \in [n]$. If it is, C_j inserts i to set I and sample uniform $y_i \leftarrow \mathbb{Z}_q$. 2. Server S chooses $\phi_1 \leftarrow \mathbb{Z}_q^n$; client C_j chooses $\phi_2 \leftarrow \mathbb{Z}_q^n$. 3. For $i \in [n]$, S holds vector $(\text{sk}, \phi_1^i) \in \mathbb{Z}_q^2$, C_j holds vector $(\phi_2^i, y_i) \in \mathbb{Z}_q^2$. Both parties run Π_{MtA} with the stated input vector above. Then, S receives $(A_1^i, B_1^i) \in \mathbb{Z}_q^2$, C_j receives $(A_2^i, B_2^i) \in \mathbb{Z}_q^2$, such that $A_2^i + A_1^i = \phi_2^i \cdot \text{sk}$, $B_2^i + B_1^i = \phi_1^i \cdot y_i$. 4. S computes $V_S = H(g^{A_1^1}, \dots, g^{A_1^n})$, and sends V_S to C_j. C_j computes $V_R = H(\text{pk}^{\phi_2^1}/g^{A_2^1}, \dots, \text{pk}^{\phi_2^n}/g^{A_2^n})$. C_j checks whether $V_R = V_S$ and aborts if they are not equal. 5. S sends m to C_j, where $m_i = \phi_1^i \cdot \text{sk} + A_1^i + B_1^i$. C_j computes $u_i = \phi_2^i \cdot y_i + A_2^i + B_2^i$ and sends it to S. Both parties compute $v = m + u$. 6. For each $i \in [n]$, S computes $h_i = g^{\phi_1^i/v_i}$. Then S sends h to C_j. C_j sets $F_{\text{sk}}(y_i) = 1, i \in I$. For each $i \in [n] \setminus I$, C_j computes $F_{\text{sk}}(y_i) = h_i \cdot g^{\phi_2^i/v_i}$. 7. C_j outputs $(F_{\text{sk}}(y_1), \dots, F_{\text{sk}}(y_n))$ if $e(g^{y_i} \cdot \text{pk}, F_{\text{sk}}(y_i)) = e(g, g)$ for each $i \in [n] \setminus I$. Otherwise it aborts.

Figure 3.4: OVUF protocol in the $(\mathcal{F}_{\text{COT}}, \mathcal{F}_{\text{BB}})$ -hybrid model with sub-protocol Π_{MtA} .

Theorem 5. *If H is modeled as a random oracle, then protocol Π_{OVUF} with sub-protocol Π_{MtA} shown in Figure 3.4 UC-realizes $\mathcal{F}_{\text{OVUF}}$ in $(\mathcal{F}_{\text{BB}}, \mathcal{F}_{\text{COT}})$ -hybrid model.*

Proof. Let $\mathcal{A}$ be a PPT adversary that allows to corrupt the server or the client. We construct a PPT simulator $\mathcal{S}$ with access to functionality $\mathcal{F}_{\text{OVUF}}$, which simulates the adversary's view. We consider the following two cases: malicious client and malicious server. The client acts as the receiver of $\mathcal{F}_{\text{COT}}$ in sub-protocol Π_{MtA} , while the server acts as the sender. We will prove that the joint distribution over the output of $\mathcal{A}$ and the honest party in the real world is indistinguishable from the joint distribution over the outputs of $\mathcal{S}$ and the honest party in the ideal world execution.

Corrupted client. Let $\mathcal{S}$ access to $\mathcal{F}_{\text{OVUF}}$ as an honest client and interact with $\mathcal{A}$ as an honest server. $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

0. $\mathcal{S}$ emulates $\mathcal{F}_{\text{BB}}$, once it receives fetch from $\mathcal{A}$. $\mathcal{S}$ sends fetch to $\mathcal{F}_{\text{OVUF}}$ and receives pk. $\mathcal{S}$ sends pk to $\mathcal{A}$.

2-3. For $i \in [n]$, $\mathcal{S}$ simulates the i th iteration of sub-protocol Π_{MtA} below.

(1)-(3) $\mathcal{S}$ emulates $\mathcal{F}_{\text{COT}}$ and receives $\mathbf{w} \in \mathbb{Z}_q^{2t+2\kappa}$ from $\mathcal{A}$. $\mathcal{S}$ samples $\mathbf{q} \leftarrow \mathbb{Z}_q^{2t}$, $\mathbf{q}' \leftarrow \mathbb{Z}_q^{4\kappa}$ and sends them to $\mathcal{A}$.

(4) $\mathcal{S}$ receives $\mathbf{g}^R$ from $\mathcal{A}$. $\mathcal{S}$ computes ϕ_2^i and y_i as follows:

$$\begin{aligned}\phi_2^i &= \sum_{j \in [t]} \mathbf{g}_j \mathbf{w}_j + \sum_{j \in [t+1, t+2\kappa]} \mathbf{g}_j \mathbf{w}_{t+j} \\ y_i &= \sum_{j \in [t]} \mathbf{g}_j \mathbf{w}_{t+j} + \sum_{j \in [t+1, t+2\kappa]} \mathbf{g}_j \mathbf{w}_{t+j}\end{aligned}$$

(5) $\mathcal{S}$ computes A_2^i, B_2^i as an honest P_1 does in step 5 in Π_{MtA} .

4. $\mathcal{S}$ samples V_S^* and sends it to $\mathcal{A}$. $\mathcal{S}$ emulates H and receives query q from $\mathcal{A}$. If $q = (\text{pk}^{\phi_2^1}/g^{A_2^1}, \dots, \text{pk}^{\phi_2^n}/g^{A_2^n})$, $\mathcal{S}$ sends $V_R^* = V_S^*$ to $\mathcal{A}$. Otherwise, $\mathcal{S}$ uniformly samples V_R^* and sends it to $\mathcal{A}$. $\mathcal{S}$ aborts if $\mathcal{A}$ aborts.

5. $\mathcal{S}$ sends $\mathbf{m}_i^* \leftarrow \mathbb{Z}_q$ to $\mathcal{A}$. $\mathcal{S}$ receives $\mathbf{u}$ from $\mathcal{A}$. $\mathcal{S}$ computes $\mathbf{v}$.

6. $\mathcal{S}$ sends $(\text{eval}, (y_1, \dots, y_n))$ to $\mathcal{F}_{\text{OVUF}}$ and waits to receive $(F_{\text{sk}}(y_1), \dots, F_{\text{sk}}(y_n))$. For each $i \in [n]$, $\mathcal{S}$ checks whether $\mathbf{u}_i = y_i \cdot \phi_2^i + A_2^i + B_2^i$. If it is, $\mathcal{S}$ simulates $\mathbf{h}_i^* = \frac{F_{\text{sk}}(y_i)}{g^{\phi_2^i/v_i}}$, and sends it to $\mathcal{A}$. Otherwise, $\mathcal{S}$ simulates $\mathbf{h}_i^* \leftarrow \mathbb{G}$ and sends it to $\mathcal{A}$.
7. $\mathcal{S}$ aborts if $\mathcal{A}$ aborts and outputs what $\mathcal{A}$ outputs.

We are going to show the simulated execution is indistinguishable from the real protocol execution.

Hybrid $\mathcal{H}_0$. Same as real-world execution in the $(\mathcal{F}_{\text{COT}}, \mathcal{F}_{\text{BB}})$ -hybrid model.

Hybrid $\mathcal{H}_1$. This hybrid is identical to $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{\text{BB}}$, $\mathcal{F}_{\text{COT}}$, the random oracle, and simulates the messages to $\mathcal{A}$ as follows:

For **step 0**, $\mathcal{S}$ emulates $\mathcal{F}_{\text{BB}}$. Upon receiving `fetch` from $\mathcal{A}$, $\mathcal{S}$ sends `fetch` to $\mathcal{F}_{\text{OVUF}}$ and receives `pk`. $\mathcal{S}$ sends `pk` to $\mathcal{A}$. In hybrid $\mathcal{H}_0$, $\mathcal{F}_{\text{BB}}$ was initialized by an honest server with `pk` and sends it to $\mathcal{A}$ upon receiving `fetch`. Thus, the `pk` sends by $\mathcal{S}$ is same as the one in hybrid $\mathcal{H}_0$.

For **step 2-3**, $\mathcal{S}$ emulates $\mathcal{F}_{\text{COT}}$, receives $\mathbf{w} \in \mathbb{Z}_q^{2t+2\kappa}$ from $\mathcal{A}$. $\mathcal{S}$ samples $\mathbf{q} \leftarrow \mathbb{Z}_q^{2t}$, $\mathbf{q}' \leftarrow \mathbb{Z}_q^{4\kappa}$ to $\mathcal{A}$, and receives $\mathbf{g}^R \in \mathbb{Z}_q^{\log q + 2\kappa}$ from $\mathcal{A}$. In Hybrid $\mathcal{H}_0$, $\mathbf{q}$ and $\mathbf{q}'$ are uniformly distributed according to $\mathcal{F}_{\text{COT}}$. Thus, the $\mathbf{q}, \mathbf{q}'$ sampled by $\mathcal{S}$ is indistinguishable from the one in Hybrid $\mathcal{H}_0$.

For **step 4**, $\mathcal{S}$ samples V_S^* to $\mathcal{A}$. Then, $\mathcal{S}$ emulates random oracle and returns $V_R^* = V_S^*$ to $\mathcal{A}$ for query $q = (\text{pk}^{\phi_1^1}/g^{A_1^1}, \dots, \text{pk}^{\phi_1^n}/g^{A_1^n})$, where A_2^i and ϕ_2^i are recovered by $\mathcal{S}$. For other queries, $\mathcal{S}$ samples V_R^* to $\mathcal{A}$. In hybrid $\mathcal{H}_0$, an honest server uses `sk` that corresponding to `pk` in Π_{MTA} and computes $V_S = H(g^{A_1^1}, \dots, g^{A_1^n})$. Since $A_1^i + A_2^i = \text{sk} \cdot \phi_2^i$ holds for an honest server, the received V_S equals to V_R if V_R is computed from $(\text{pk}^{\phi_2^1}/g^{A_2^1}, \dots, \text{pk}^{\phi_2^n}/g^{A_2^n})$ honestly. Thus, the simulated V_R^* equals to V_S^* if $\mathcal{A}$ query random oracle honestly, which is indistinguishable from Hybrid $\mathcal{H}_0$.

For **step 5**, $\mathcal{S}$ sends random $\mathbf{m}_i^* \leftarrow \mathbb{Z}_q$ to $\mathcal{A}$. In hybrid $\mathcal{H}_0$, an honest server computes $\mathbf{m}_i = \phi_1^i \cdot \text{sk} + A_1^i + B_1^i$ and sends it to $\mathcal{A}$. $\mathbf{m}_i$ satisfies the distribution that $\mathbf{m}_i + y_i \cdot \phi_2^i + A_2^i + B_2^i = \mathbf{v}_i = (\phi_1^i + \phi_2^i)(\text{sk} + y_i)$. Since ϕ_1^i is randomly sampled by an honest server, $\mathcal{A}$ has no idea about the distribution of $\mathbf{m}_i$. The simulated $\mathbf{m}_i^*$ is randomly uniform in $\mathbb{Z}_q$ as well, which is indistinguishable from hybrid $\mathcal{H}_0$. Thus, the view simulated by $\mathcal{S}$ is identical to hybrid $\mathcal{H}_0$.

For **step 6**, $\mathcal{S}$ sends $(\text{eval}, (y_1, \dots, y_n))$ to $\mathcal{F}_{\text{OVUF}}$ and waits for $(F_{\text{sk}}(y_1), \dots, F_{\text{sk}}(y_n))$. $\mathcal{S}$ checks whether the received $\mathbf{u}_i$ is computed from $y_i \cdot \phi_2^i + A_2^i + B_2^i$ correctly. If it is, $\mathcal{S}$ sends $\mathbf{h}_i^* =$

$\frac{F_{\text{sk}}(y_i)}{g^{\phi_2^i/(m_i^*+u_i)}}$ to $\mathcal{A}$, such that $\mathbf{h}_i^* \cdot g^{\phi_2^i/(m_i^*+u_i)} = F_{\text{sk}}(y_i)$. Otherwise, $\mathcal{S}$ samples $\mathbf{h}_i^* \leftarrow \mathbb{G}$ and sends it to $\mathcal{A}$. In hybrid $\mathcal{H}_0$, an honest server sends $\mathbf{h}_i = g^{\phi_1^i/(m_i+u_i)}$ to $\mathcal{A}$. If u_i is computed honestly from $y_i \cdot \phi_2^i + A_2^i + B_2^i$, then $\mathbf{h}_i \cdot g^{\phi_2^i/(m_i+u_i)} = F_{\text{sk}}(y_i)$. Thus, the view of $\mathbf{h}_i^*$ in hybrid $\mathcal{H}_0$ is identical to $\mathbf{h}_i$ in Hybrid $\mathcal{H}_0$. If u_i is not computed honestly, then $m_i + u_i \neq (\phi_1^i + \phi_2^i)(\text{sk} + y_i)$ and thus $\mathbf{h}_i \cdot g^{\phi_2^i/(m_i+u_i)} \neq F_{\text{sk}}(y_i)$. In hybrid $\mathcal{H}_1$, $\mathcal{S}$ simulates $\mathbf{h}_i^* \leftarrow \mathbb{G}$, we have $\mathbf{h}_i^* \cdot g^{\phi_2^i/(m_i^*+u_i)} = F_{\text{sk}}(y_i)$ w.p. $2^{-\log q}$, which is indistinguishable from hybrid $\mathcal{H}_0$. Thus, the view simulated by $\mathcal{S}$ is identical to hybrid $\mathcal{H}_0$.

Corrupted server. Let $\mathcal{S}$ access to the $\mathcal{F}_{\text{OVUF}}$ as an honest server and interact with $\mathcal{A}$ as an honest client. $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

0. $\mathcal{S}$ emulates $\mathcal{F}_{\text{BB}}$, once it receives the pk from $\mathcal{A}$, $\mathcal{S}$ stores pk and ignores subsequent messages from $\mathcal{A}$. $\mathcal{S}$ sends (init, pk) to $\mathcal{F}_{\text{OVUF}}$.

2-3. Upon receiving n from $\mathcal{F}_{\text{OVUF}}$, $\mathcal{S}$ simulates iterations of Π_{MtA} for each $i \in [n]$. The i th iteration of Π_{MtA} is simulated as follows:

(1)-(3) $\mathcal{S}$ emulates $\mathcal{F}_{\text{COT}}$ and receives a vector $\tau \in \mathbb{Z}_q^{2(t+2\kappa)}$ from $\mathcal{A}$. $\mathcal{S}$ samples $\mathbf{p} \leftarrow \mathbb{Z}_q^{2t}$, $\mathbf{p}' \leftarrow \mathbb{Z}_q^{4\kappa}$ and sends them to $\mathcal{A}$. $\mathcal{S}$ checks whether the received $\tau \in \mathbb{Z}_q^{2(t+2\kappa)}$ satisfies a pattern that for $k \in [2]$, all the bits τ_j , $j \in [(k-1)t+1, kt] \cup j = 2t+k+(l-1)2$, $l \in [2s]$ are the same. For $k=1$, if τ_j are the same, $\mathcal{S}$ extracts $\text{sk}' = \tau_j$. For $k=2$, if τ_j are the same, $\mathcal{S}$ extracts $\phi_1^i = \tau_j$.

(4) $\mathcal{S}$ samples $\mathbf{g}^R \leftarrow \mathbb{Z}_q^{\log q + 2\kappa}$ and sends $\mathbf{g}^R$ to $\mathcal{A}$.

(5) $\mathcal{S}$ computes A_1^i, B_1^i as an honest P_0 does in step 5 in Π_{MtA} .

4. $\mathcal{S}$ emulates random oracle H and receives query q from $\mathcal{A}$. $\mathcal{S}$ samples V_S to $\mathcal{A}$ and records (q, V_S) . Once $\mathcal{S}$ receives V_S from $\mathcal{A}$, $\mathcal{S}$ first checks whether sk' 's have been extracted and are the same in last step. $\mathcal{S}$ also checks whether $g^{\text{sk}'} = \text{pk}$. Then, $\mathcal{S}$ checks if the corresponded $q = (g^{A_1^1}, \dots, g^{A_1^n})$. If all these requirements are satisfied, $\mathcal{S}$ continue; Otherwise, $\mathcal{S}$ aborts.

5. $\mathcal{S}$ receives $\mathbf{m}$ from $\mathcal{A}$. $\mathcal{S}$ samples $\mathbf{u}^* \leftarrow \mathbb{Z}_q^n$ to $\mathcal{A}$. $\mathcal{S}$ computes $\mathbf{v}^* = \mathbf{m} + \mathbf{u}^*$.

6. $\mathcal{S}$ waits to receive $\mathbf{h}$.
7. For each i th iteration, if ϕ_1^i is extracted, $\mathcal{S}$ checks whether $\mathbf{h}_i = g^{\phi_1^i/v_i^*}$, $\mathbf{m}_i = \phi_1^i \cdot \text{sk}' + A_1^i + B_1^i$ and sends sk' to $\mathcal{F}_{\text{OVUF}}$. Otherwise, $\mathcal{S}$ aborts.

We are going to show the simulated execution is indistinguishable from the real protocol execution.

Hybrid $\mathcal{H}_0$. Same as real-world execution in $(\mathcal{F}_{\text{BB}}, \mathcal{F}_{\text{COT}})$ -hybrid model.

Hybrid $\mathcal{H}_1$. This hybrid is identical to $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{\text{BB}}$, $\mathcal{F}_{\text{COT}}$, the random oracle, and generates the messages to $\mathcal{A}$ as follows:

For Step 2-3, $\mathcal{S}$ emulates $\mathcal{F}_{\text{COT}}$ and waits to receive τ . Then, $\mathcal{S}$ sends $\mathbf{p} \leftarrow \mathbb{Z}_q^{2t}$ and $\mathbf{p}' \leftarrow \mathbb{Z}_q^{4\kappa}$ to $\mathcal{A}$. $\mathcal{S}$ samples $\mathbf{g}^R \leftarrow \mathbb{Z}_q^{\log q + 2\kappa}$ and sends it to $\mathcal{A}$. For an honest client in hybrid $\mathcal{H}_0$, it samples $\mathbf{p} \leftarrow \mathbb{Z}_q^{2t}$, $\mathbf{p}' \leftarrow \mathbb{Z}_q^{4\kappa}$, $\mathbf{g}^R \leftarrow \mathbb{Z}_q^{\log q + 2\kappa}$ as well, which is indistinguishable from this hybrid.

For Step 5, $\mathcal{S}$ receives $\mathbf{m}$ and samples $\mathbf{u}_i^* \leftarrow \mathbb{Z}_q$ to $\mathcal{A}$. In the hybrid $\mathcal{H}_0$, for i th iteration, if $\mathcal{A}$ sends τ correctly, $\mathbf{u}_i + \phi_1^i \cdot \text{sk} + A_1^i + B_1^i = \mathbf{v}_i = (\phi_1^i + \phi_2^i)(\text{sk} + y_i)$, which is uniformly distributed over $\mathbb{Z}_q$. Thus, $\mathbf{u}_i$ is uniformly distributed, same as the sampled one in this hybrid. If there exists an error e sampled by $\mathcal{A}$, an honest client computes $\mathbf{u}_i$ such that $\mathbf{u}_i + \phi_1^i \cdot \text{sk} + A_1^i + B_1^i = \mathbf{v}_i = (\phi_1^i + \phi_2^i)(\text{sk} + y_i) + \text{diff}_i$. $\text{diff}_i = \mathbf{f}_1^i + \mathbf{f}_2^i$, where $\mathbf{f}_1^i$ resulted from incorrect $\phi_1^i \cdot y_i$ and $\mathbf{f}_2^i$ resulted from incorrect $\text{sk} \cdot \phi_2^i$ as stated in Equation 3.3. Since e is defined by $\mathcal{A}$ before knowing $\mathbf{g}^R$, $\mathbf{g}^R$ appears uniformly random over $\mathbb{Z}_q^{\log q + 2\kappa}$ to $\mathcal{A}$ at this time. Thus, for any given $\mathbf{w}$ and e , $\mathbf{f}_i$ is uniformly distributed over $\mathbb{Z}_q$. Therefore, diff_i is uniformly distributed over $\mathbb{Z}_q$. $\mathbf{u}_i$ is uniformly distributed over $\mathbb{Z}_q$, which is identically distributed as the simulated $\mathbf{u}_i^*$ in this hybrid.

Hybrid $\mathcal{H}_2$. This hybrid is identical to $\mathcal{H}_1$ except $\mathcal{S}$ aborts at Step 4 in the following conditions: 1) any sk' in Π_{MTA} iterations is not extractable; 2) the extracted sk' s are not the same or any $g^{\text{sk}'} \neq \text{pk}$; 3) the q corresponding to the received V_S is not equal to $(g^{A_1^1}, \dots, g^{A_1^n})$. $\mathcal{S}$ aborts at Step 7 in the following conditions: 1) any ϕ_1^i in Π_{MTA} iterations is not extractable; 2) $\mathbf{h}_i^* \neq g^{\phi_1^i/(\mathbf{m}_i + \mathbf{u}_i^*)}$ or $\mathbf{m}_i \neq \phi_1^i \cdot \text{sk} + A_1^i + B_1^i$.

For Step 4 in hybrid $\mathcal{H}_1$, an honest client aborts if the received $V_S \neq V_R$. The client computes $V_R = H(\text{pk}^{\phi_2^1}/g^{A_2^1}, \dots, \text{pk}^{\phi_2^n}/g^{A_2^n})$. For each $\text{pk}^{\phi_2^i}/g^{A_2^i}$, it equals to $g^{\text{sk} \cdot \phi_2^i - A_2^i}$. 1) Adversary $\mathcal{A}$

might add error e to τ on bits related to sk in one iteration of Π_{MtA} . In this case, sk' is unextractable. If we set a value as sk' , then both parties holds equation $A_1^i + A_2^i = \text{sk}' \cdot \phi_2^i + \mathbf{f}_2$, where $\mathbf{f}_2$ is computed according to Equation. 3.3. As we analyzed above, $\mathbf{f}_2$ is uniformly distributed over $\mathbb{Z}_q$ for any given w and e . Thus, with V_S computed from A_1^i , $V_S \neq V_R$. Furthermore, even $\mathcal{A}$ tries to manipulate V_S , $\mathcal{A}$ is unable to construct $A_1^{i'} = \text{sk} \cdot \phi_2^i - A_2^i$ as ϕ_2^i and A_2^i are uniformly distributed. Thus, the client aborts with all but negligible probability, which is indistinguishable from condition (1) in hybrid $\mathcal{H}_2$. 2) Adversary $\mathcal{A}$ might use inconsistent sk in different Π_{MtA} iterations. If $\mathcal{A}$ use $\text{sk}' \neq \text{sk}$ in Π_{MtA} , both parties holds equation $A_1^i + A_2^i = \text{sk}' \cdot \phi_2^i$. Thus, with V_S computed from A_1^i , $V_S \neq V_R$. $\mathcal{A}$ is not able to construct $A_1^{i'}$ to manipulate V_S either. Thus, the client aborts with all but negligible probability, which is indistinguishable from condition (2) in hybrid $\mathcal{H}_2$. 3) When adversary use correct sk but manipulate V_S from inconsistent q , an honest client aborts which is indistinguishable from condition (3) in hybrid $\mathcal{H}_2$.

For Step 7 in hybrid $\mathcal{H}_1$, an honest client aborts when $F_{\text{sk}}(y_i)$ does not satisfy $e(g^{y_i \cdot pk}, F_{\text{sk}}(y_i)) = e(g, g)$, where $F_{\text{sk}}(y_i) = \mathbf{h}_i \cdot g^{\phi_2^i / (\mathbf{m}_i + \mathbf{u}_i)}$. 1) For the i th iteration of Π_{MtA} , if adversary adds error e to τ such that ϕ_1^i is not extractable, then $\text{diff}_i \neq 0$ with all but negligible probability. Thus, $\mathbf{m}_i + \mathbf{u}_i \neq (\text{sk} + y_i)(\phi_1^i + \phi_2^i)$ with all but negligible probability. Consequently, $\mathbf{h}_i \cdot g^{\phi_2^i / (\mathbf{m}_i + \mathbf{u}_i)} \neq F_{\text{sk}}(y_i)$ with all but negligible probability, identical to condition (1) in hybrid $\mathcal{H}_2$. 2) If Π_{MtA} is executed honestly by $\mathcal{A}$, an honest client aborts if $\mathcal{A}$ provides an incorrect $\mathbf{m}_i$ or $\mathbf{h}_i$, which will result in an incorrect $F_{\text{sk}}(\mathbf{y}_i)$ that does not satisfy the verification procedure, identical to condition (2) in hybrid $\mathcal{H}_2$. Therefore, this hybrid is identically distributed as the previous one.

The above hybrid argument completes this proof. $\square$

3.2.2.4. Complexity Analysis

For each input element $y_i \in Y$, Π_{OVUF} requires $4 \log q + 4\kappa$ COT and one g^R . Thus, this protocol requires $(4 \log q + 4\kappa)n$ COT and n g^R in total. To improve its complexity, we propose an improved OVUF in Appendix A.1 that reduces the number of g^R s to two and achieves better RAM usage.

The key idea is to batch operations with correlated randomness together but refer to Appendix A.1 for complete description of the protocol and the proof.

3.3. Performance Evaluation

We implement our protocols using EMP [75] for COT and RELIC [76] for pairings. We benchmark the performance of our protocol when $\mathcal{F}_{\text{COT}}$ is instantiated using KOS [77] and Ferret [46].

3.3.1. Benchmark Setup

We instantiate everything ensuring a computational security parameter $\lambda = 128$ and a statistical security parameter $\kappa = 40$. To this end, we use BLS12-381 for all type-III pairing operations. We show the performance in two different network settings: a LAN network with 5Gbps bandwidth and a WAN network with 120 Mbps bandwidth. All experiments are performed on AWS EC2 instances of `6a.8xlarge` type with 32vCPU and 128 GB memory.

3.3.2. Efficiency of Server's Encoding

First, we benchmark the performance of the server encoding process. Note that this computation only needs to be executed once given a set of elements. Recall that this step mainly computes the VUF on the input elements. Following conventions from prior works, we hash the output to 64-bit strings, which helps in reducing the encoding size. For example, the encoding file for a set of 10^8 elements is of size 800 MB.

We prepare a list of 256-bit values in a file as the server's set. The benchmark results include the time to: 1) read all elements from the file (w/ disk access), 2) compute the VUF value of each element and then hash it into a 64-bit string, and 3) write the resulting hashes into another file (w/disk access). In Figure 3.5a, we show the performance of our server computation with different set sizes and threads. From the figure, we can see that the performance of the server's local encoding is linear to the set size. We observe a $3.8\times$ improvement when increasing the threads from 1 to 4 and $15\times$ from 1 to 32 threads. We didn't make the file I/O multi-threaded

which we believe could be the bottleneck when we use 32 threads.

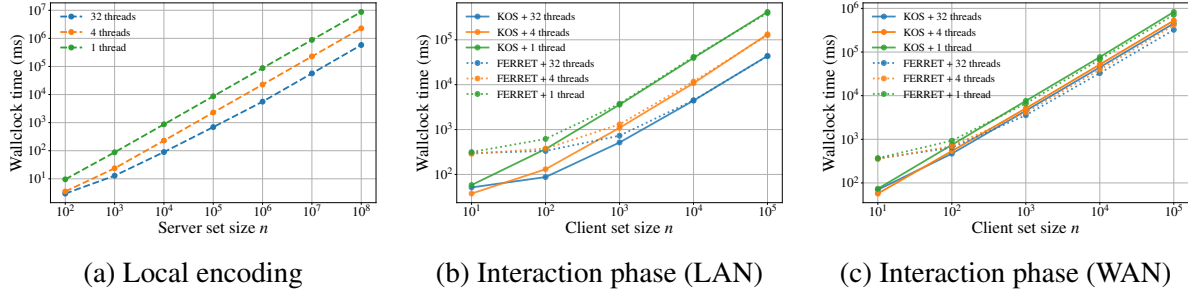


Figure 3.5: Performance of our protocol. We show the performance of both phases. The one-time offline local encoding time for the server set is depicted in (a). The interactive online encoding time for the client set is shown in (b) under LAN network and (c) under WAN network. Both (b) and (c) utilize different OT methods (KOS/FERRET) and number of threads (1/4/32) for comparison.

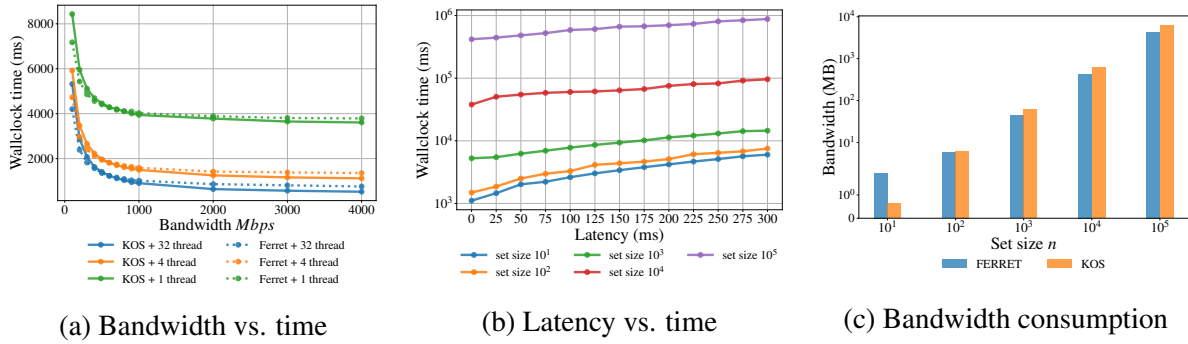


Figure 3.6: Our performance under different network settings. We show our performance of time consumption as bandwidth varies in (a) and as latency varies in (b). (a) uses client set size 10^3 to compare performance under different OT methods (KOS/FERRET) and different thread numbers (1/4/32). (b) takes OT method KOS to compare performance as set sizes vary. Figure (c) shows our protocol’s bandwidth consumption as the set size varies.

3.3.3. Efficiency of Online Computation

Now we show the performance of the interactive process between a server with a VUF secret key, and a client with a private set. As the output, the client will get VUF evaluation on its own set, which can be further used to lookup the server encoding.

Wallclock time. In Figure 3.5b and Figure 3.5c, we show the wallclock of the protocol for different client set sizes. Similarly, the time reported includes the client: 1) reading its own elements from

a file, 2) running OVUF with a server to compute $F_{sk}(x_i)$; 3) computing the hash to derive 64-bit strings that can be used for local matching.

With 1 thread, the average cost for the client to process each element is $8.29ms$ in the WAN setting and $4.17ms$ in the LAN setting. With 32 threads, the average cost is $4.53ms$ in the WAN setting and $0.43ms$ in the LAN setting. Noted that Ferret computes COT in large batches, it is not competitive when the set is small, where the protocol cannot consume all COTs. When the set size is large, our protocol in the LAN setting using KOS or Ferret does not show much difference as they have similar computational costs. In the WAN setting, we can observe a slight improvement with Ferret because it consumes less bandwidth. However, the improvement is not huge because the communication caused by our protocol, not counting the cost of COT, is already significant.

Performance dependence on network. We show the efficiency of our protocol under different network condition in Figure 3.6a and Figure 3.6b. According to Figure 3.6a, the efficiency of a client with a set size of 10^3 in a WAN environment increases as the bandwidth increases. However, once the bandwidth reaches $1Gbps$, the efficiency does not improve significantly with further increases in bandwidth. This indicates that our protocol performs best with bandwidth larger than $1Gbps$. In TCP networks, there is a dependency between latency and bandwidth limitations, wherein an increase in latency leads to a decrease in available bandwidth. Figure 3.6b illustrates that the total protocol wallclock time increases as bandwidth decreases due to added latency. For larger sets that use up more bandwidth, the rise in wallclock time is more significant than for smaller sets experiencing the same increase in latency.

Bandwidth consumption. Regarding bandwidth consumption in Figure 3.6c, we observed that if the set size is less than 10^2 , the protocol using KOS OT performs better in terms of bandwidth usage compared to that using FERRET OT. However, this situation changes once the set size exceeds 10^2 . For a set size of 10^5 , the KOS OT protocol requires $61.7KB$ to process one element, while the FERRET OT protocol needs $43.0KB$ to encode one element. This is the same reason as we stated in **Wallclock time**, that Ferret computes COT in large batches but consumes less bandwidth for each COT compared with KOS. For small set size, Ferret is more bandwidth-intensive as it

generates more COT than necessary. However, for large set size, Ferret is more efficient as the generated COTs can be utilized and each one consumes less bandwidth than KOS.

3.3.4. Comparison with Other Protocols

Our protocol works in a special setting where a server with one set repeatedly runs PSI with many clients with small sets. We noticed that existing prior works do not perform well if used in our setting directly; this is not surprising as they are not designed for this setting. Below, we show some comparisons to state-of-the-art protocols in classical PSI settings.

Comparing with state-of-the-art PSI. The first possible solution is to use the best fully malicious secure PSI protocol [14], and have the server run this protocol with each client. However, there exists a security issue that the server might differentiate its set among different clients. Additionally, the performance is poor: each execution of PSI with a different client requires the server to transmit a different encoding of its set over the internet, which incurs great costs. In Figure 3.7, we compare the cost by the server per client between our protocol and the Blazing-fast [14], which is so far the fastest and improved upon VOLE-based PSI [13]. We assume the server set has 10^8 elements, and the client set ranges from 10^2 to 10^4 elements. We use the real execution time and the instance’s unit price (0.1728USD/Hour for `6a.xlarge`) to compute computational cost. We also estimate the communication cost by multiplying the data size that the server transfers out by the communication unit price (0.05USD/GB). Notice that for our scheme, since the server’s encoding is reusable, we use AWS CloudFront (CDN) to manage it, thereby reducing this part of the communication cost to a lower unit price (0.02USD/GB). The computation of this reusable server set encoding is a one-time and offline process, making the cost per client negligible when amortized. For our scheme, the total cost is 3x lower for a client set 10000 and 5x lower for client sets 100 and 1000. With smaller set sizes, the cost is primarily dominated by the CDN cost, which is a fixed value of 0.016 USD per client. If we switch to managing the server’s encoding through a peer-to-peer network to eliminate the CDN cost, our scheme achieves an 8x reduction in communication cost and a 2x reduction in computation cost compared to Blazing-PSI for a client size

of 10000. In this case, the cost of our scheme scales linearly with the client set size and performs better with smaller client sizes.

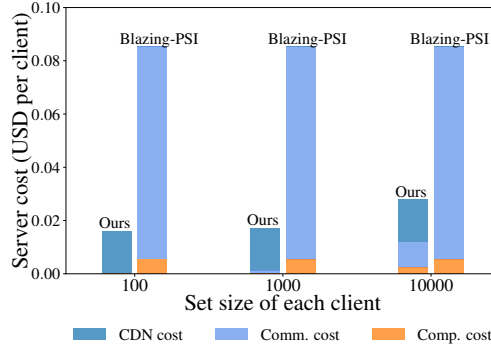


Figure 3.7: Server cost comparison with Blazing-PSI [14]. All experiments are run on AWS instance. Costs are estimated based on AWS instance pricing and network pricing.

Comparing with PSI featuring reusable server encoding. Some unbalanced PSI could be better suited to our setting which allows pushing some work to the offline stage as well. In Table 3.1, we show our protocol performs scalably compared to related protocols across server set sizes $\{2^{20}, 2^{28}\}$ and client set sizes $\{2^7, 2^{10}\}$:

- OPRF-based solutions by [20] allows the server to reuse its computation and encoding that is linear to X across multiple clients. We include two solutions, one based on LowMC PRF and one based on Naor–Reingold PRF. We also update their hash output to achieve a similar level of false positive rate. Our protocol runs at a similar time to OPRF-based protocols with about three times more communication; however, that allows us to achieve full malicious security.
- FHE-based solution [12], [16] does not require sending large encoding but requires more computation. The computation could be made reusable across multiple clients by performing OPRF on top of the value, but existing FHE-PSI implementations or benchmarks do not include these extra steps. We can see that our solution is much faster in terms of online time when the server set is at a large scale of 2^{28} , albeit with higher communication costs. All FHE-based solutions only implement their semi-honest version and could not be made fully malicious secure; however, we do believe that by incorporating our OVUF-based solution, it is possible to achieve full

malicious security as well, which we leave as future work.

- Finally, we also compare with a DH-based solution by Resende and Aranha [78]. The solution is semi-honest, but the original proposal by Jarecki and Liu [79] also includes malicious counterparts, which require further use of zero-knowledge proofs to show correct encoding. This approach essentially follows the VOPRF method, where all efficient solutions do not allow extracting client’s input in the proof. As such, their solution requires much less communication.

Set size		Security	Protocol	Offline		Online	
$ X $	$ Y $			time (s)	comm. (MB)	time (s)	comm. (MB)
2^{28}	2^{10}	Semi-honest	[20] (w/ LowMC)	164.82	2144	1.37	23.6
			[20] (w/ NR)	44681	2144	0.63	6.07
			[12]	4628	0	12.1	18.4
			[16]	4371	0	23.35	12.86
			[78]	3684.1	2415	0.16	0.07
		Malicious	Ours (w/ KOS)	1556.7	2147	0.44	63.23
2^{28}	2^7	Semi-honest	[20] (w/ LowMC)	164.82	2144	0.41	2.96
			[20] (w/ NR)	44681	2144	0.13	0.77
			[12]	4628	0	12.1	18.4
			[16]	4350	0	25.65	12.81
			[78]	3684.1	2415	0.02	0.008
		Malicious	Ours (w/ KOS)	1556.7	2147	0.11	7.92
2^{20}	2^{10}	Semi-honest	[20] (w/ LowMC)	0.51	8.37	1.37	23.6
			[20] (w/ NR)	173.41	8.37	0.63	6.07
			[12]	1.1	0	0.5	5
			[16]	13.4	0	1.1	2.04
			[78]	18.17	9.43	0.06	0.031
		Malicious	Ours (w/ KOS)	5.85	8.38	0.44	63.23
2^{20}	2^7	Semi-honest	[20] (w/ LowMC)	0.51	8.37	0.41	2.96
			[20] (w/ NR)	173.41	8.37	0.13	0.77
			[12]	1.2	0	0.2	3.9
			[16]	13.5	0	1.1	1.99
			[78]	18.17	9.43	0.027	0.004
		Malicious	Ours (w/ KOS)	5.85	8.38	0.11	7.92

Table 3.1: Performance of unbalanced PSI with server set X and client set Y . Our protocol and [16] were tested with 32 threads, with [16] using 256 GB RAM for a 2^{28} server set. [12] is obtained based on numbers from their paper, which is based on faster hardware than our testbed. [20] is tested with 32 threads for the offline phase and 1 thread for the online phase. [78] is tested with 1 thread, except the case of $|X| = 2^{28}$ where the online performance are extrapolated.

CHAPTER 4

Committed-Vector Oblivious Linear Evaluation and its Applications

Vector oblivious linear evaluation (VOLE) is a type of protocol that sets up linear correlations between two parties. It can be viewed as an arithmetic extension of the correlated oblivious transfer [80]. By running a VOLE protocol between P_A , who has an input vector $\mathbf{x} \in \mathbb{F}^m$, and P_B , P_A would obtain a uniform vector $M[\mathbf{x}] \in \mathbb{F}^m$ as output, and P_B would obtain $(\Delta, K[\mathbf{x}]) \in \mathbb{F} \times \mathbb{F}^m$ such that they satisfy the correlation $M[\mathbf{x}] = K[\mathbf{x}] + \mathbf{x} \cdot \Delta$. VOLE correlation can be viewed as an information-theoretic message authentication code (IT-MAC) on P_A 's input, hence serving as a commitment or materials to construct more useful correlations for MPC like Beaver triples. Therefore, extensive practical applications in zero-knowledge proofs (ZKPs) [45], [81], [82], multi-party computation [83], [84], [85], private set intersection (PSI) [13], [14], etc, have been found recently.

In many VOLE-based protocols, it is often desirable to allow P_A to 1) first non-interactively commit to its input, which can be distributed to all parties easily, and later 2) run VOLE with other parties while ensuring that P_A uses the committed input to the VOLE protocol. Such a capability enforces P_A to use the same input across multiple executions of VOLE, potentially with different parties. Moreover, the commitment could be pre-distributed to parties offline using cheaper mechanisms like content distribution networks. In more detail:

1. VOLE-based ZKP [45], [81], [82] is a recent paradigm for constructing lightweight ZKP protocols. However, they are designated verifier ZK and do not provide a cheap mechanism if the statement needs to use a consistent witness across different ZKPs. A recent work [86] explored proving the decommitment phase of a symmetric-key-based commitment; however, its performance is fairly limited due to the large size of SHA-256 and AES. Other works [87], [88], [89], [90] have explored designing ZK-friendly hash functions, but the cost is high as the underlying commitment schemes always require non-black-box use.

2. Among existing dishonest-majority MPC protocols, the most practical way to achieve malicious security is to use IT-MACs to authenticate secret shares [83], [85], [91], [92]. These protocols often require running pair-wise VOLE across all parties so that even if all but one parties are corrupted, the underlying secret sharing is still authenticated. However, a critical issue is ensuring that every party uses the same input vector for all VOLE executions with every other party. To achieve that goal, existing MPC protocols require a joint consistency check among all the parties, which often involves significant computation and communication.
3. For PSI in the client–server setting, the server has a large set and repeatedly runs a PSI protocol with many individual clients, using the same set across different PSI executions. It is useful when in services where many clients need to privately look up some private information on the server side, like password checkup or private contact discovery. Prior work [6] in this setting with malicious security requires a public-key-based pseudorandom function (PRF) and its oblivious evaluation for every element. To adapt existing VOLE-based PSI protocols [13], [14], the most performant PSI protocols right now, to work in this setting, one would need to ensure that the same input vector from the server is used for all VOLE executions, which in turn ensures consistency in the server’s set across all executions.

To summarize, the above examples either require P_A ’s VOLE input vector x to be publicly committed beforehand or require P_A ’s VOLE input vector x to remain the same among multiple VOLE applications with different parties, which can also be ensured if all parties have the same commitment of x . We provide a unified approach to solve all these issues efficiently in a modular way. To this end, we propose *committed-VOLE (C-VOLE)*, which allows the VOLE sender’s input to be pre-committed non-interactively.

In this chapter, we propose concretely efficient C-VOLE protocols based on the assumption of learning parity with noise (LPN). Then, we show how C-VOLE can be used in multiple important protocols and how our new protocols can enhance the efficiency of these applications.

LPN-based C-VOLE protocol. We propose a C-VOLE protocol based on the LPN assumption. Prior work [86] proposed a protocol using a commitment scheme based on random oracle and AES

and proved the correct decommitment, which involves computing SHA256 and AES within ZK. We design an LPN-based commitment scheme specifically for C-VOLE to achieve the best possible efficiency. Then, we exploit the fact that this commitment scheme and the state-of-the-art VOLE protocols share a similar structure, i.e., the LPN-like structure, to design a zero-knowledge proof of decommitment with very small communication. As a result, our C-VOLE protocol achieves significantly better online communication efficiency than the corresponding VOLE protocol, while requiring $2\times$ the online running time. We show how our protocol can be based on either dual-LPN-based VOLE or primal-LPN-based VOLE.

C-VOLE applications and their performance. We provided some concrete case studies of using C-VOLE in prior applications and how it can improve the efficiency of previous works. Additionally, we implement our C-VOLE protocol and benchmark its practical performance.

For our C-VOLE protocol, the offline commitment is nearly the same size as the data being committed. The online communication is almost constant, with a running time of about 900 ns and 420 ns per element in a LAN setting using 4 and 16 threads, respectively, for a vector of length 2^{26} . Note that running VOLE with a chosen input requires essentially the same amount of communication as the input size to “fix” a random correlation to the chosen input. Therefore, ours communication overhead is near optimal while most communication is offline. We also investigate the protocol change and the concrete efficiency when applying our C-VOLE protocol in various applications.

1. C-VOLE directly enables ZK proof on publicly committed values. Compared to Mystique, we observe a $28\times$ improvement in running time and $7\times$ improvement in communication cost.
2. Regarding the multi-party VOLE in generic MPC protocols, C-VOLE avoids the final consistency check, which contains heavy peer-to-peer and broadcast communications. In the context of multi-party VOLE in dishonest-majority MPC and multi-verifier ZKPs [85], [92], [93], [94], C-VOLE similarly avoids the costly final consistency check, significantly reducing the round complexity.

3. By adapting C-VOLE to the VOLE-PSI protocol, we can convert state-of-the-art, actively secure one-shot PSI protocol to the client–server setting without assuming the server’s honest behavior. The C-VOLE PSI protocol requires around 750 ns per server element for computing one-time commitment and around 900 ns per server element for the online interaction with each client, given the server set size of 2^{24} . It reduces online communication overhead by $35\times$ compared to the state-of-the-art PSI protocol in the client–server setting [6].

Other related notions. Note that a different notion of committing VOLE is recently defined in [95] to construct MPC with an input-revealing identifiable abort guarantee. Their focus is a VOLE variant where parties inputs and outputs are committed so that they can later be opened. Therefore, their notion does not provide input consistency between different executions of VOLE. Thus, their notion is more similar to committing OT notation proposed by Jawurek et al. [96].

4.1. Protocol Intuition

In this section, we provide some intuition of our constructions.

Prior VOLE constructions Recall that a *random* VOLE protocol outputs $(\mathbf{u}, \mathbf{M}[\mathbf{u}]) \in \mathbb{F}^m \times \mathbb{F}^m$ to P_A and $(\Delta, \mathbf{K}[\mathbf{u}]) \in \mathbb{F} \times \mathbb{F}^m$ to P_B , such that all values are uniform under the constraint that

$$\mathbf{M}[\mathbf{u}] = \mathbf{K}[\mathbf{u}] + \mathbf{u} \cdot \Delta.$$

When two parties hold VOLE correlations for vector $\mathbf{u}$ like the above, we also write $\langle \mathbf{u} \rangle$ for simplicity. All of the state-of-the-art VOLE protocols [45], [97], [98] compute such random correlation because this kind of correlations can be easily compressed to achieve sublinear communication and they can be easily derandomized to chosen input. More specifically, if P_A wants to correct $\mathbf{u}$ to a designated vector $\mathbf{x}$, it simply sends the difference $\mathbf{d} = \mathbf{u} - \mathbf{x}$ to P_B and defines $\mathbf{M}[\mathbf{x}] = \mathbf{M}[\mathbf{u}]$; P_B can define $\mathbf{K}[\mathbf{x}] = \mathbf{K}[\mathbf{u}] + \mathbf{d} \cdot \Delta$. Now these new values form a correlation

$$\mathbf{M}[\mathbf{x}] = \mathbf{M}[\mathbf{u}] = \mathbf{K}[\mathbf{u}] + \mathbf{u} \cdot \Delta = \mathbf{K}[\mathbf{u}] + (\mathbf{u} - \mathbf{x}) \cdot \Delta + \mathbf{x} \cdot \Delta = \mathbf{K}[\mathbf{x}] + \mathbf{x} \cdot \Delta.$$

To efficiently construct random VOLE, state-of-the-art protocols follow the pseudorandom correlation generator (PCG) framework [44]. For example, the dual version of the learning parity with noise (LPN) assumption states that

$$(\mathbf{H}, \mathbf{u}) \text{ is computationally indistinguishable from } (\mathbf{H}, \mathbf{e}_u \mathbf{H}),$$

where $\mathbf{H} \in \mathbb{F}^{cm \times m}$ is a public random but structured matrix and $\mathbf{e}_u \in \mathbb{F}^{cm}$ is a sparse vector. To allow more efficient constructions, state-of-the-art protocols also assume that $\mathbf{e}_u$ is regular, meaning that it partition $\mathbf{e}_u$ into w number of shorter vectors $(\mathbf{e}_u^1, \dots, \mathbf{e}_u^w)$ each of length 2^L such that each shorter vector is non-zero only at one random location. With this assumption, one can construct an efficient VOLE by first constructing “simple” VOLE on each $\mathbf{e}_u^i$ ’s to obtain $\langle \mathbf{e}_u^i \rangle$, following [44], [99], [100] with cost $O(L\kappa)$, where κ is the computational security parameter. Then, parties can assemble $\langle \mathbf{e}_u \rangle = \langle \mathbf{e}_u^1 \rangle \parallel \dots \parallel \langle \mathbf{e}_u^w \rangle$ locally and further compute $\langle \mathbf{e}_u \mathbf{H} \rangle = \langle \mathbf{e}_u \rangle \cdot \mathbf{H}$ due to linear homomorphism. As a result the total communication cost is sublinear. However, when P_A interacts with different parties, there is no guarantee that they will use the same randomness and hence no guarantee about the consistency of P_A ’s input.

Our main contribution is to design a customized commitment scheme and proof of consistency to force P_A using the same value with every party who holds the commitment.

Blueprint of our protocol. From a high-level view, our protocol functions as follows. First P_A , with an input vector $\mathbf{x} \in \mathbb{F}^m$, can locally compute a commitment of size roughly $m \cdot \log |\mathbb{F}|$ bits. This commitment is distributed to all relevant parties by various means, such as a broadcasting protocol or public bulletin board, and its genuineness can be verified with a signature, assuming PKI exists. Because the same commitment is distributed to all parties, it can enjoy network caching at a reduced cost. Then P_A can interact with each individual party acting as P_B in VOLE and execute the online phase of the C-VOLE protocol such that they obtain VOLE correlation about $\mathbf{x}$ while the protocol ensures that P_A must use $\mathbf{x}$ as its input (otherwise the protocol would abort). The commitment is used for P_A to correct VOLE of random vector to its input $\mathbf{x}$. Different parties

acting as P_B all know that P_A uses the same input under the pre-distributed commitment.

At the core of the protocol is the online phase, where a consistency check is needed to show that the committed x is consistent with the x in VOLE. Assuming a commitment scheme Com and a pseudorandom generator (PRG) PRG , one can convert the problem to ensuring the consistency of a seed as follows. The commitment to x is now $\text{com}_x = (\text{PRG}(r) - x, \text{Com}(r))$. During the online phase, two parties obtain the VOLE correlation for r and x ; then, they can prove using VOLE-ZK that r satisfies two properties. First, when fed to the PRG, it yields a vector consistent with the first value in the commitment com_x , i.e.,

$$\text{PRG}(\langle r \rangle) - \langle x \rangle = \text{com}_x[0].$$

Second, $\langle r \rangle$ is consistent with $\text{Com}(r)$. Parties execute a proof of decommitment to obtain $\langle \text{decom} \rangle = \text{Open}(\text{com}_x[1])$. Then, P_A proves that

$$\langle \text{decom} \rangle = \langle r \rangle.$$

Both are statements relevant to VOLE values; thus, VOLE-based ZK only requires one extra message from P_A but the communication complexity is linear to the circuit size of PRG and $\text{Open}(\cdot)$. The rest of this section discusses how to construct PRG and Com so that proving the above two statements using VOLE-based ZK involves almost no communication.

Commitment based on LPN assumption. Since state-of-the-art VOLE uses LPN assumptions to generate pseudorandom bits, our goal is to design the commitment for it. Rigorously, the underlying PRG can be defined as $\text{PRG} \in \{0, 1\}^{wL} \times \mathbb{F}^w \rightarrow \{0, 1\}^m$:

$$\text{PRG}(k_1, \dots, k_w; a_1, \dots, a_w) = \text{Unary}(k_1, a_1) \parallel \dots \parallel \text{Unary}(k_w, a_w) \cdot \mathbf{H},$$

where $\mathbf{H} \in \mathbb{F}^{w2^L \times m}$ is a public matrix and $\text{Unary}(k_i, a_i)$ is defined as an all zero vector with a_i at position k_i . The seed of this PRG consists of wL bits and w elements in $\mathbb{F}$, while the output is

a vector of m elements. In practice, to maintain 128-bit computational security, we set $w = 224$, $L = 23$, and $m = 2^{28}$, which means the output is much larger than the input, hence a good PRG.

The next step is to design a commitment scheme for $\mathbf{k} = (k_1, \dots, k_w)$ and $\mathbf{a} = (a_1, \dots, a_w)$. We propose to use an LPN-based commitment [101] due to its linearity. For example, to commit the vector $\mathbf{a}$, one can compute

$$\mathbf{a} \cdot \mathbf{H}_1 + \hat{\mathbf{r}} \cdot \mathbf{H}_2 + \hat{\mathbf{e}},$$

where $\mathbf{H}_1$ and $\mathbf{H}_2$ are two public matrices, $\hat{\mathbf{r}}$ is a uniform vector, and $\hat{\mathbf{e}}$ is a sparse vector, all with appropriate length. It can be shown that when picking the dimension of the two matrices and the sparsity of $\hat{\mathbf{e}}$ appropriately, $\hat{\mathbf{r}} \cdot \mathbf{H}_2 + \hat{\mathbf{e}}$ is pseudorandom and thus provides hiding, while it is still difficult to find two pairs of $(\mathbf{a}, \hat{\mathbf{r}}, \hat{\mathbf{e}})$ and $(\mathbf{a}', \hat{\mathbf{r}}', \hat{\mathbf{e}}')$ that lead to the same value, hence binding.

Proving the consistency is fairly cheap: P_A and P_B can first run the VOLE protocol with P_A providing $\mathbf{k}$ and $\mathbf{a}$. As part of the VOLE protocol, two parties would obtain VOLE on values $e_u^i = \text{Unary}(k_i, a_i)$, namely $\langle e_u^i \rangle$ for each $i \in [w]$. One can easily check the correct value of a_i by checking $a_i = \sum_{j \in [2^L]} e_u^i[j]$, which is free. Checking the sparsity of e_u^i is no longer free but can still be done with linear cost by running the binary-to-unary circuit in ZKP from $k_i \in \{0, 1\}^L$ to a Boolean vector of length 2^L , which takes a total of $2m$ non-linear gates. The cost is orders of magnitude better than Mystique, which requires more than 50 non-linear gate per output due to a large AES circuit.

C-VOLE with sublinear online communication. The above protocol is much faster than prior work, but the online communication cost is still linear to the vector size. Note that we already send linear communication as part of the offline commitment, so we ideally want the online phase to be even smaller. This means the consistency check can only prove a circuit of size sublinear to the output size!

In the above protocol, the most expensive operation is proof of sparsity; it is required for the binding property. Our crucial observation is that the VOLE protocol already ensures that each $\langle e_u^i \rangle$ is sparse; we need to prove the binary-to-unary circuit only to ensure that it is consistent with the

committed randomness. To this end, we modify our commitment. Instead of committing to the seed of the LPN-PRG, we commit to some intermediate values in the PRG, i.e., the structured sparse vector, directly. More specifically, our commitment scheme no longer handles any input but only sparse vectors e_u :

$$e_u \cdot \mathbf{H}_1 + e_r \cdot \mathbf{H}_2,$$

where $\mathbf{H}_1$ and $\mathbf{H}_2$ are two public matrices, and e_r is a sparse vector. We can show that it is computationally infeasible to obtain two (e_u, e_r) and (e'_u, e'_r) such that the above commitments are the same and that e'_u is also regular with specified sparsity. Note that because the output is much shorter than $|e_u|$, finding a collision is possible if the sparsity is not guaranteed by other means; but again, in our application, VOLE protocols already ensure it for free.

Now, our protocol is extremely efficient. Two parties obtain $\langle e_u \rangle$ as part of the normal VOLE protocol. They just need to obtain $\langle e_r \rangle$ and check if the $\langle e_u \rangle \cdot \mathbf{H}_1 + \langle e_r \rangle \cdot \mathbf{H}_2$ matches the commitment. We show that $|e_r| \ll |e_u|$, and thus, this extra communication cost is very small compared to the cost of the VOLE protocol. In addition to the VOLE protocol, the main computational cost is the evaluation of $\langle e_u \rangle \cdot \mathbf{H}_1$, which is only a local computation. We provide the detailed description and analysis of LPN-based commitment in Section 4.2.1.1 and Appendix B.1

Applications to VOLE-ZK, MPC, and MVZK. C-VOLE brings immediate performance improvement when used in several recent VOLE-ZK and MPC protocols.

- In VOLE-based ZKP protocols, C-VOLE directly leads to a way to convert public commitments to VOLE-like authentication ready for VOLE-ZK. Compared to the previous work [86], it avoids proving large SHA-256 and AES circuits in ZKP and achieves $28\times$ concrete improvement.
- For generic secure multi-party computation, it can be used to execute multi-party VOLE among all parties [85], [92], [93]. By using C-VOLE, they no longer require heavy peer-to-peer and broadcast communication in the final consistency check. MVZK with dishonest-majority verifiers [94] also relies on similar consistency checks, which can be avoided by C-VOLE.

We defer the details to Section 4.3.

Application to client–server PSI. In the client–server PSI setting, we have one server holding a large set X and a set of clients, each holding a small set Y_i , such that each pair want to obtain the intersection $X \cap Y_i$. State-of-the-art PSI protocols based on VOLE [13], [14] can provide malicious security but cannot ensure consistency of X . A prior work [6] works in this setting but uses public-key based Oblivious PRF (OPRF), which is highly expensive.

We observe that the only place where VOLE-PSI protocols use VOLE is for the server to feed its input set to the VOLE. The server’s input set is encoded into a vector $\mathbf{p}$, such that $\text{Decode}(\mathbf{p}, x) = H_1(x)$ for $x \in X$. The server identifies any client’s set elements in the intersection using an OPRF defined by $\langle \mathbf{p} \rangle$. Specifically, the client encodes its set elements by $F(x) = H_2(\text{Decode}(\mathbf{K}[\mathbf{p}], x) + \Delta \cdot H_1(x), x)$, where H_1 and H_2 are two hash functions, and sends the encodings to the server. The server encodes its set elements by $X' := \{H_2(\text{Decode}(\mathbf{M}[\mathbf{p}], x), x) | x \in X\}$ and checks for matches with the client’s encoding. The two encodings match if the client’s set element is located in the intersection, which satisfies $\text{Decode}(\mathbf{p}, x) = H_1(x)$. In this case, VOLE is utilized as a black-box component in the VOLE-PSI protocol. Compared to VOLE, C-VOLE enforces the consistency of $\mathbf{p}$ across different VOLE executions with other clients, ensuring consistent use of the server’s set. Thus, we can prove that if we replace VOLE with C-VOLE, we can obtain a PSI protocol in the client–server setting.

As a result, we upgrade the state-of-the-art single-client PSI without the set-committing feature into a client–server PSI. The C-VOLE PSI requires around 900 ns per server element for online computation and less than 1.64 B per server element for online communication of set size 2^{20} , which is roughly $2\times$ online time consuming but $13\times$ better online communication overhead compared with VOLE-PSI. Compared with prior work [6] in the client–server setting, C-VOLE PSI reduces initialization time with a 8-fold improvement for set size 2^{20} . It also decreases online communication to 1.73 MB with clients having 2^{10} elements, reduced by a factor of $35\times$.

4.2. Protocol and Security Analysis

This section presents the C-VOLE protocol and its security analysis. We define the C-VOLE functionality below, give an LPN-based construction with a security proof, and then show how C-VOLE securely realizes the server-output PSI functionality (Figure 2.10) in Section 4.2.3.

Committed-vector oblivious linear evaluation (C-VOLE) is a variant of VOLE that allows two parties to generate VOLE relations for a pre-committed vector $\mathbf{x} \in \mathbb{F}^m$. The commitment of $\mathbf{x}$, denoted as $\text{Hcom}_{\mathbf{x}}$, is public. The holder of $\mathbf{x}$ serves as P_A , while any party knows the public commitment $\text{Hcom}_{\mathbf{x}}$ can act as P_B and execute the functionality depicted in Figure 4.1 with P_A . The functionality $\mathcal{F}_{\text{C-VOLE}}$ first checks whether P_A holds the correct $(\mathbf{x}, \text{Hdecom}_{\mathbf{x}})$ that satisfies the public commitment $\text{Hcom}_{\mathbf{x}}$. It then samples a vector $\mathbf{K}[\mathbf{x}] \leftarrow \mathbb{F}^m$ and a field element $\Delta \leftarrow \mathbb{F}$ to P_B and delivers $\mathbf{M}[\mathbf{x}] = \mathbf{K}[\mathbf{x}] + \mathbf{x} \cdot \Delta \in \mathbb{F}^m$ to P_A , as the VOLE functionality typically does.

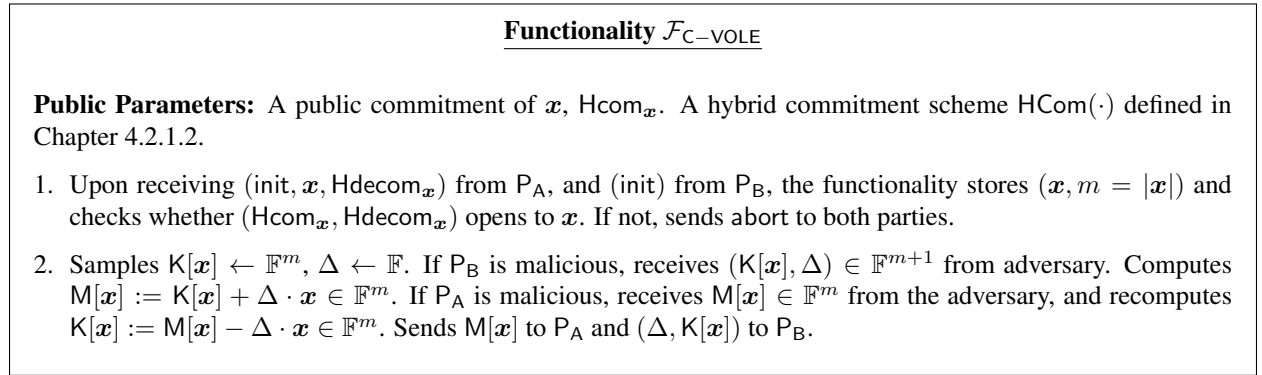


Figure 4.1: The committed-VOLE functionality.

4.2.1. Commitment from LPN

We introduce the commitment scheme we use, denoted as a hybrid commitment, in the committed-VOLE construction. As outlined in Chapter 4.1, the hybrid commitment scheme relies on the LPN assumption and consists of two parts: an LPN commitment to an intermediate regular sparse vector and a correction to the designated committed vector.

We construct the LPN commitment specifically for a regular sparse vector. In Theorem 6, we prove that with sparse message distribution, the new LPN commitment schemes still satisfy the

requirements of binding and hiding. Then, we add a correction vector to the LPN commitment to construct a hybrid commitment scheme for a designated vector, specifically designed for the C-VOLE construction. The proposed hybrid commitment satisfy the binding and hiding property as detailed in Theorem 7.

4.2.1.1. LPN Commitment

Here, we define a commitment scheme $\text{Com}(\cdot)$ based on the primal LPN assumption for committing to a regular- w vector $\mathbf{x}$. This vector $\mathbf{x} \in \mathbb{F}^m$, for parameters w, L and $m = w \cdot 2^L$, consists of w blocks, each of size 2^L , with each block contains exactly one non-zero element.

- **ParamGen.** The public commitment parameter is represented by $K = \{\mathbf{A}_1, \mathbf{A}_2\}$, where $\mathbf{A}_1 \leftarrow \mathcal{C}(N, m, \mathbb{F})$ and $\mathbf{A}_2 \leftarrow \mathbb{F}^{k \times N}$, with $N = t \cdot 2^{L'}$. The algorithm $\mathcal{C}$ is a probabilistic code generation algorithm that outputs a matrix $\mathbf{A}_1 \in \mathbb{F}^{m \times N}$.
- **Commit.** The commitment to a regular- w message $\mathbf{x} \in \mathbb{F}^m$ is expressed as $\text{com}_{\mathbf{x}} = \mathbf{x} \cdot \mathbf{A}_1 + \mathbf{r} \cdot \mathbf{A}_2 + \mathbf{e}$, where $\mathbf{r} \leftarrow \mathbb{F}^k$ and $\mathbf{e}$ is a weight- t regular noise vector of length N . The opening of the commitment is given by $\text{decom}_{\mathbf{x}} = \mathbf{r}$.
- **Open.** Given a commitment $\text{com}_{\mathbf{x}}$, a regular- w message $\mathbf{x}$, and randomness $\text{decom}_{\mathbf{x}}$, a verifier accepts if $\text{com}_{\mathbf{x}} - \mathbf{x} \cdot \mathbf{A}_1 - \text{decom}_{\mathbf{x}} \cdot \mathbf{A}_2$ is weight- t regular.

The corresponding dual LPN commitment scheme $\text{Com}(\cdot)$ for a regular- w message $\mathbf{x} \in \mathbb{F}^m$ is described below:

- **ParamGen.** The public commitment parameter is represented by $K \stackrel{\text{def}}{=} \{\mathbf{H}_1, \mathbf{H}_2\}$, where $\mathbf{H}_1 \leftarrow \mathcal{C}^\perp(n, m, \mathbb{F})$ and $\mathbf{H}_2 \leftarrow \mathbb{F}^{N \times n}$, with $N = t \cdot 2^{L'}$. The algorithm $\mathcal{C}^\perp$ is a probabilistic code generation algorithm that outputs a matrix $\mathbf{H}_1 \in \mathbb{F}^{m \times n}$.
- **Commit.** The commitment to a regular- w message $\mathbf{x} \in \mathbb{F}^m$ is expressed as $\text{com}_{\mathbf{x}} = \mathbf{x} \cdot \mathbf{H}_1 + \mathbf{e} \cdot \mathbf{H}_2$, where $\mathbf{e}$ is a weight- t regular noise vector of length N . The opening of the commitment is given by $\text{decom}_{\mathbf{x}} = \mathbf{e}$.

- **Open.** Given a commitment com_x , a regular- w message x , and randomness decom_x , a verifier accepts if decom_x is weight- t regular and $\text{com}_x = x \cdot \mathbf{H}_1 + \text{decom}_x \cdot \mathbf{H}_2$.

Theorem 6. Assume $L', k, t \in \mathbb{N}$ such that the decisional primal-LPN(N, k, t) problem (with secrets of length k and N samples) and the decisional dual-LPN($N, n = N - k, t$) problem are computationally hard, where $N = t \cdot 2^{L'}$. Then the above two commitment schemes are ε -statistically binding and computationally hiding, where $\lambda = 2^{2wL+2tL'} / |\mathbb{F}|^{t \cdot 2^{L'} - k - 2w - 2t}$.

Proof. The security properties are proved as follows:

1. Statistically binding.

- **For the commitment scheme based on the primal LPN problem.** Suppose there exists a commitment c corresponding to two messages, $x_1 \in \mathbb{F}^m$ and $x_2 \in \mathbb{F}^m$, with randomness pairs (r_1, e_1) and (r_2, e_2) . Define $x \stackrel{\text{def}}{=} x_1 - x_2 \in \mathbb{F}^m$, $r \stackrel{\text{def}}{=} r_1 - r_2 \in \mathbb{F}^k$ and $e \stackrel{\text{def}}{=} e_1 - e_2 \in \mathbb{F}^N$. Given that each x_i is a weight- w regular vector of length $m = w \cdot 2^L$ and each e_i is a weight- t regular vector of length $N = t \cdot 2^{L'}$, the probability that $x \cdot \mathbf{A}_1 + r \cdot \mathbf{A}_2 + e = 0$ is at most $2^{2wL+2tL'} / |\mathbb{F}|^{t \cdot 2^{L'} - 2w - 2t}$. A union bound over all possible $r \in \mathbb{F}^k$ yields the claimed bound $2^{2wL+2tL'} / |\mathbb{F}|^{t \cdot 2^{L'} - k - 2w - 2t}$.
- **For the commitment scheme based on the dual LPN problem.** Considering the commitment c associated with two messages, $x_1 \in \mathbb{F}^m$ and $x_2 \in \mathbb{F}^m$, along with their respective randomness e_1 and e_2 . Define $x \stackrel{\text{def}}{=} x_1 - x_2 \in \mathbb{F}^m$ and $e \stackrel{\text{def}}{=} e_1 - e_2 \in \mathbb{F}^N$. Given that each x_i is a weight- w regular vector of length $m = w \cdot 2^L$ and each e_i is a weight- t regular vector of length $N = t \cdot 2^{L'}$, the probability that $x \cdot \mathbf{H}_1 + e \cdot \mathbf{H}_2 = 0$ is at most $2^{2wL+2tL'} / |\mathbb{F}|^{t \cdot 2^{L'} - k - 2w - 2t}$.

- ### 2. Computationally hiding.
- In the commitment scheme based on the primal LPN problem, we have $c = x \cdot \mathbf{A}_1 + r \cdot \mathbf{A}_2 + e$. In the commitment scheme based on the dual LPN problem, the commitment is $c = x \cdot \mathbf{H}_1 + e \cdot \mathbf{H}_2$. Leveraging the security properties of the primal LPN and dual LPN problems, the terms $r \cdot \mathbf{A}_2 + e$ and $e \cdot \mathbf{H}_2$ are pseudorandom, ensuring the pseudorandomness of the respective commitments.

□

4.2.1.2. Hybrid Commitment

We present the hybrid commitment scheme $\text{HCom}(\cdot)$, designed for C-VOLE construction for messages $\mathbf{x} \in \mathbb{F}^m$, with the following components:

- **ParamGen.** The public commitment parameter consists of the public commitment parameters of Com and a matrix $\mathbf{H}^\top \leftarrow \mathbf{C}^\perp(N, m, \mathbb{F})$, where $N = w \cdot 2^L$. The algorithm $\mathbf{C}^\perp$ is a probabilistic code generation algorithm that outputs a matrix $\mathbf{H} \in \mathbb{F}^{N \times m}$.
- **Commit.** The hybrid commitment to a message $\mathbf{x} \in \mathbb{F}^m$ is expressed as $\text{Hcom}_\mathbf{x} = \{\text{com}_e, \mathbf{c} = e \cdot \mathbf{H} - \mathbf{x}\}$, where e is regular- w vector of length N and $(\text{com}_e, \text{decom}_e) \leftarrow \text{Com}(e)$. The opening of the commitment is given by $\text{Hdecom}_\mathbf{x} = \{e, \text{decom}_e\}$.
- **Open.** Given a commitment $\text{Hcom}_\mathbf{x} = \{\text{com}_e, \mathbf{c}\}$, a message $\mathbf{x}$, and randomness $\text{Hdecom}_\mathbf{x} = \{e, \text{decom}_e\}$, a verifier accepts if $\{e, \text{com}_e, \text{decom}_e\}$ passes verification of Com , ensuring e is a weight- w regular vector; and $\mathbf{c} = e \cdot \mathbf{H} - \mathbf{x}$.

Theorem 7. Assume $L, m, t \in \mathbb{N}$ such that the decisional dual-LPN(N, m, w) problem, with matrix $\mathbf{H}^\top \leftarrow \mathbf{C}^\perp(N, m, \mathbb{F})$, is computationally hard, where $N = w \cdot 2^L$. If the commitment scheme Com is λ -statistically binding and computationally hiding, then the aforementioned hybrid commitment scheme is also λ -statistically binding and computationally hiding.

Proof. The required security properties are as follows:

1. **Statistically binding.** e is λ -statistically binding according to the commitment scheme Com . Given that $\mathbf{x} = e \cdot \mathbf{H} - \mathbf{c}$, it follows that $\mathbf{x}$ is also λ -statistically binding.
2. **Computationally hiding.** The hybrid commitment is based on the dual LPN problem, constructing the commitment $\mathbf{c} = e \cdot \mathbf{H} - \mathbf{x}$. Leveraging the security properties of the dual LPN problem, the term $e \cdot \mathbf{H}$ is pseudorandom, ensuring the pseudorandomness of the resulting commitment.

□

4.2.2. Committed-VOLE

We construct the committed-VOLE protocol using the hybrid commitment we introduced in Chapter 4.2.1.2. The C-VOLE protocol starts from one party P_A , generating a hybrid commitment of its chosen vector $\mathbf{x}$ and publishing the commitment to every other party. Then, P_A generates VOLE correlation with any other party P_B of the $\mathbf{x}$ consistent with the published hybrid commitment. In the construction of the hybrid commitment, there are two methods to commit the intermediate vector $\mathbf{e}_u$: dual-LPN commitment and primal-LPN commitment. This leads to two ways to instantiate the protocol: dual-LPN based C-VOLE and primal-LPN based C-VOLE. Here, we use the dual-LPN based hybrid commitment to P_A 's input vector $\mathbf{x}$ and provide its corresponding C-VOLE instantiation in Figure 4.2. The primal-LPN based C-VOLE is introduced and evaluated in Appendix B.2. The concrete performance is provided in performance evaluation in Chapter 4.4.

The C-VOLE protocol based on dual-LPN hybrid commitment is constructed as follows: First, P_A commits to its chosen input vector $\mathbf{x}$. P_A samples an intermediate regular sparse vector $\mathbf{e}_u$ and computes a dual-LPN commitment $\text{com}_{\mathbf{e}_u}$. P_A computes the hybrid commitment $\text{Hcom}_{\mathbf{x}}$ of $\mathbf{x}$ as $\text{Hcom}_{\mathbf{x}} = \{\text{com}_{\mathbf{e}_u}, \mathbf{x}^* = \mathbf{e}_u \cdot \mathbf{H} - \mathbf{x}\}$, $\text{Hdecom}_{\mathbf{x}} = \{\mathbf{e}_u, \text{decom}_{\mathbf{e}_u}\}$, where $\mathbf{H}$ is a public matrix and $\mathbf{x}^*$ is a correction vector from the intermediate $\mathbf{e}_u$ to the designated $\mathbf{x}$. Then, to generate the VOLE correlation of the designated $\mathbf{x}$ committed by $\text{Hcom}_{\mathbf{x}}$, we let both parties first generate the VOLE correlation of the intermediate $\mathbf{e}_u$ committed by $\text{com}_{\mathbf{e}_u}$ (part of $\text{Hcom}_{\mathbf{x}}$). Given $\langle \mathbf{e}_u \rangle$ is consistent with $\text{com}_{\mathbf{e}_u}$, both parties can locally recover $\langle \mathbf{x} \rangle$, which is consistent with $\text{Hcom}_{\mathbf{x}}$, using the publicly known correction vector $\mathbf{x}^*$ (part of $\text{Hcom}_{\mathbf{x}}$) as follows:

Given $M[\mathbf{e}_u] = K[\mathbf{e}_u] + \mathbf{e}_u \cdot \Delta$, where $(\mathbf{e}_u, M[\mathbf{e}_u])$ is held by P_A and $(K[\mathbf{e}_u], \Delta)$ is held by P_B , and the public correction vector $\mathbf{x}^* = \mathbf{e}_u \cdot \mathbf{H} - \mathbf{x}$, both parties compute $M[\mathbf{x}], K[\mathbf{x}]$, respectively, as:

$$M[\mathbf{x}] = M[\mathbf{e}_u] \cdot \mathbf{H}$$

$$K[\mathbf{x}] = K[\mathbf{e}_u] \cdot \mathbf{H} + \mathbf{x}^* \cdot \Delta,$$

such that $M[\mathbf{x}] = K[\mathbf{x}] + \mathbf{x} \cdot \Delta$.

In this way, generating the VOLE correlation for the committed $\mathbf{x}$ is reduced to generating the

VOLE correlation for the committed intermediate vector e_u . Notice that e_u is a regular- w vector with length N , where $N = w \cdot 2^L$, defined in Chapter 4.2.1.2. Both parties call the multi-point VOLE functionality $\mathcal{F}_{\text{MPVOLE}}$, as depicted in Figure 2.4, to generate $\langle e_u \rangle$. The instantiation of $\mathcal{F}_{\text{MPVOLE}}$ follows the single-point subfield VOLE Π_{spsVOLE} in [45] with a slight variation, which is detailed in Chapter B.4. In the next step, P_A needs to prove to P_B that with e_u input to $\mathcal{F}_{\text{MPVOLE}}$, the resulted $\langle e_u \rangle$ is consistent with commitment com_{e_u} . We construct the proof efficiently leveraging the linear property of the dual-LPN commitment.

Recall that for dual-LPN commitment of e_u , $\text{com}_{e_u} = e_u \cdot \mathbf{H}_1 + e_r \cdot \mathbf{H}_2$ and $\text{decom}_{e_u} = e_r$. We let both parties call $\mathcal{F}_{\text{MPVOLE}}$ again with P_A input $\text{decom}_{e_u}(e_r)$ to generate $\langle e_r \rangle$. Given $M[e_u] = K[e_u] + e_u \cdot \Delta$, where $(e_u, M[e_u])$ is held by P_A and $(K[e_u], \Delta)$ is held by P_B , and $M[e_r] = K[e_r] + e_r \cdot \Delta$, where $(e_r, M[e_r])$ is held by P_A and $(K[e_r])$ is held by P_B , both parties compute $M[\text{com}_{e_u}]$ and $K[\text{com}_{e_u}]$, respectively:

$$\begin{aligned} M[\text{com}_{e_u}] &= M[e_u] \cdot \mathbf{H}_1 + M[e_r] \cdot \mathbf{H}_2 \\ K[\text{com}_{e_u}] &= K[e_u] \cdot \mathbf{H}_1 + K[e_r] \cdot \mathbf{H}_2 \end{aligned}$$

Since the LPN-commitment com_{e_u} is published in the hybrid commitment Hcom_x of x , the following equation ($\langle \text{com}_{e_u} \rangle$) holds,

$$M[\text{com}_{e_u}] = K[\text{com}_{e_u}] + \text{com}_{e_u} \cdot \Delta$$

if the $\langle e_u \rangle$ and $\langle e_r \rangle$ used in computing $M[\text{com}_{e_u}]$ and $K[\text{com}_{e_u}]$ are consistent with the e_u and e_r in computing com_{e_u} . We let P_A send $H_3(M[\text{com}_{e_u}])$ to P_B . P_B verifies whether $\langle \text{com}_{e_u} \rangle$ holds, thereby ensuring that a consistent intermediate VOLE correlation $\langle e_u \rangle$ is generated with the intermediate commitment com_{e_u} .

The detailed protocol for instantiating $\mathcal{F}_{C\text{-VOLE}}$ with a dual-LPN commitment is shown in Figure 4.2. We also proved that the dual-LPN commitment based protocol $\Pi_{C\text{-VOLE-dual}}$ is secure against any malicious adversary, as established in Theorem 8 with a complete proof.

Protocol $\Pi_{C-VOLE-dual}$

Parameters: A public hybrid commitment of $\mathbf{x} \in \mathbb{F}^m$ defined in Chapter 4.2.1.2, that $\text{Hcom}_{\mathbf{x}} = (\text{com}_{e_u} \in \mathbb{F}^n, \mathbf{x}^* = e_u \cdot \mathbf{H} - \mathbf{x} \in \mathbb{F}^m)$. A dual LPN commitment scheme defined in Chapter 4.2.1.1 that $(\text{com}_{e_u} = e_u \cdot \mathbf{H}_1 + e_r \cdot \mathbf{H}_2, \text{decom}_{e_u} = e_r) \leftarrow \text{Com}(e_u)$, where $e_u \in \mathbb{F}^{w \cdot 2^L}$, $\text{com}_{e_u} \in \mathbb{F}^n$, $\text{decom}_{e_u} \in \mathbb{F}^{t \cdot 2^{L'}}$. A Collision-Resistant Hash Function H_3 .

Inputs: P_A holds vector $\mathbf{x} \in \mathbb{F}^m$, $\text{Hdecom}_{\mathbf{x}} = (e_u \in \mathbb{F}^{w \cdot 2^L}, e_r \in \mathbb{F}^{t \cdot 2^{L'}})$.

Compute:

1. Both P_A and P_B call $\mathcal{F}_{\text{MPVOLE}}$ with input (init). P_B receives $\Delta \in \mathbb{F}$ from $\mathcal{F}_{\text{MPVOLE}}$.
2. P_A calls $\mathcal{F}_{\text{MPVOLE}}$ with input (extend, $e_u \in \mathbb{F}^{w \cdot 2^L}, L, w$). P_B calls $\mathcal{F}_{\text{MPVOLE}}$ with input (extend, L, w). The functionality returns $M[e_u] \in \mathbb{F}^{w \cdot 2^L}$ to P_A and $K[e_u] \in \mathbb{F}^{w \cdot 2^L}$ to P_B such that $M[e_u] = K[e_u] + \Delta \cdot e_u \in \mathbb{F}^{w \cdot 2^L}$. If either party receives **abort** from $\mathcal{F}_{\text{MPVOLE}}$ in any of these executions, it aborts.
3. P_A calls $\mathcal{F}_{\text{MPVOLE}}$ with input (extend, $e_r \in \mathbb{F}^{t \cdot 2^{L'}}, L', t$). P_B calls $\mathcal{F}_{\text{MPVOLE}}$ with input (extend, L', t). The functionality returns $M[e_r] \in \mathbb{F}^{t \cdot 2^{L'}}$ to P_A and $K[e_r] \in \mathbb{F}^{t \cdot 2^{L'}}$ to P_B such that $M[e_r] = K[e_r] + \Delta \cdot e_r \in \mathbb{F}^{t \cdot 2^{L'}}$. If either party receives **abort** from $\mathcal{F}_{\text{MPVOLE}}$ in any of these executions, it aborts.
4. P_A computes $M[\text{com}_{e_u}] = M[e_u] \cdot \mathbf{H}_1 + M[e_r] \cdot \mathbf{H}_2 \in \mathbb{F}^n$ and sends $H(M[\text{com}_{e_u}])$ to P_B . P_B computes $K[\text{com}_{e_u}] = K[e_u] \cdot \mathbf{H}_1 + K[e_r] \cdot \mathbf{H}_2 \in \mathbb{F}^n$ and **value** = $K[\text{com}_{e_u}] + \text{com}_{e_u} \cdot \Delta \in \mathbb{F}^n$. P_B checks whether $H_3(M[\text{com}_{e_u}]) = H_3(\text{value})$. If not, P_B aborts.
5. P_A outputs $(\mathbf{x}, M[\mathbf{x}])$, where $M[\mathbf{x}] = M[e_u] \cdot \mathbf{H} \in \mathbb{F}^m$. P_B outputs $(\Delta, K[\mathbf{x}])$, where $K[\mathbf{x}] = K[e_u] \cdot \mathbf{H} + \mathbf{x}^* \Delta \in \mathbb{F}^m$.

Figure 4.2: The committed-VOLE protocol based on dual-LPN-based hybrid commitment

4.2.2.1. Proof of Security

Theorem 8. *Protocol $\Pi_{C-VOLE-dual}$ in Figure 4.2 securely instantiates $\mathcal{F}_{C-VOLE}$ in Figure 4.1 in $\mathcal{F}_{\text{MPVOLE}}$ -hybrid model against any malicious adversaries.*

Proof. Let $\mathcal{A}$ be a PPT adversary that corrupt P_A or P_B . We construct a PPT simulator $\mathcal{S}$ with access to $\mathcal{F}_{C-VOLE}$ and simulates the adversary's view. We will prove that the joint distribution over the output of $\mathcal{A}$ and the honest party in the real world is indistinguishable from the joint distribution over the output of $\mathcal{S}$ and the honest party in the ideal world.

Corrupted P_A . Let $\mathcal{S}$ access $\mathcal{F}_{C-VOLE}$ as an honest P_A and interact with $\mathcal{A}$ as an honest P_B . $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

1. $\mathcal{S}$ holds public commitment of $\mathbf{x}$, that $\text{Hcom}_{\mathbf{x}} = (\text{com}_{e_u}, \mathbf{x}^* = e_u \cdot \mathbf{H} - \mathbf{x})$. $\mathcal{S}$ emulates $\mathcal{F}_{\text{MPVOLE}}$ and receives (init) from $\mathcal{A}$.

2. $\mathcal{S}$ emulates $\mathcal{F}_{\text{MPVole}}$ and receives $(\text{extend}, e'_u \in \mathbb{F}^{w \cdot 2^L}, L, w)$ from $\mathcal{A}$. $\mathcal{S}$ receives $M[e'_u] \in \mathbb{F}^{w \cdot 2^L}$ from $\mathcal{A}$. $\mathcal{S}$ computes $\mathbf{x}' = e'_u \cdot \mathbf{H} - \mathbf{x}^*$.
3. $\mathcal{S}$ emulates $\mathcal{F}_{\text{MPVole}}$ and receives $(\text{extend}, e'_r \in \mathbb{F}^{t \cdot 2^{L'}}, L', t)$ from $\mathcal{A}$. $\mathcal{S}$ receives $M[e'_r] \in \mathbb{F}^{t \cdot 2^{L'}}$ from $\mathcal{A}$. $\mathcal{S}$ sends $(\mathbf{x}', \text{decom}'_x = (e'_u, e'_r))$ to $\mathcal{F}_{\text{C-Vole}}$. If $\mathcal{S}$ receives **abort** from $\mathcal{F}_{\text{C-Vole}}$, $\mathcal{S}$ aborts at step (4).
4. $\mathcal{S}$ receives $H_3(M[\text{com}_{e_u}])'$ from $\mathcal{A}$. $\mathcal{S}$ checks if $H_3(M[\text{com}_{e_u}])' = H_3(M[e'_u] \cdot \mathbf{H}_1 + M[e'_r] \cdot \mathbf{H}_2)$. If not, $\mathcal{S}$ aborts.
5. $\mathcal{S}$ computes $M[\mathbf{x}] = M[e'_u] \cdot \mathbf{H}$ and sends $M[\mathbf{x}]$ to $\mathcal{F}_{\text{C-Vole}}$. $\mathcal{S}$ outputs what $\mathcal{A}$ outputs.

We are going to show the simulated execution is indistinguishable from real-world protocol execution.

Hybrid $\mathcal{H}_0$ Same as real-world execution in $\mathcal{F}_{\text{MPVole}}$ -hybrid model.

Hybrid $\mathcal{H}_1$ Same as Hybrid $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{\text{MPVole}}$. Notice that $\mathcal{S}$ does not simulate any message to $\mathcal{A}$. This hybrid is identical to the previous one.

Hybrid $\mathcal{H}_2$ Same as Hybrid $\mathcal{H}_1$ except $\mathcal{S}$ aborts at step (4) if $\mathcal{S}$ receives **abort** from $\mathcal{F}_{\text{C-Vole}}$ or the received $H_3(M[\text{com}_{e_u}])' \neq H_3(M[e'_u] \cdot \mathbf{H}_1 + M[e'_r] \cdot \mathbf{H}_2)$.

In hybrid $\mathcal{H}_1$, an honest $\mathcal{P}_B$ aborts at step (4) if the received $H_3(M[\text{com}_{e_u}])' \neq H_3(K[e'_u] \cdot \mathbf{H}_1 + K[e'_r] \cdot \mathbf{H}_2 + \text{com}_{e_u} \cdot \Delta)$. There are the following abort conditions with abort probability indistinguishable from hybrid $\mathcal{H}_2$:

1) $\mathcal{A}$ uses $e'_u \neq e_u$ and $e'_r \neq e_r$ to $\mathcal{F}_{\text{MPVole}}$. $\mathcal{A}$ computes $M[\text{com}_{e_u}]'$ honestly from $M[e'_u] \cdot \mathbf{H}_1 + M[e'_r] \cdot \mathbf{H}_2$. Any (e'_u, e'_r) inconsistent with (e_u, e_r) will result the inequality above with the probability of statistical binding failure mentioned in Theorem 6. In this hybrid, any inconsistency will be checked by the $\mathcal{F}_{\text{C-Vole}}$ with the same binding failure and result in an abort at step (4), which is indistinguishable from hybrid $\mathcal{H}_1$. 2) $\mathcal{A}$ uses $e'_u \neq e_u$ and $e'_r \neq e_r$ to $\mathcal{F}_{\text{MPVole}}$. $\mathcal{A}$ samples $M[\text{com}_{e_u}]'$ to $\mathcal{S}$. The consistency check passes if and only if $\mathcal{A}$ samples $M[\text{com}_{e_u}]' = K[e'_u] \cdot \mathbf{H}_1 + K[e'_r] \cdot \mathbf{H}_2 + \text{com}_{e_u} \cdot \Delta$. However, since $K[e'_u]$, $K[e'_r]$, Δ are uniformly distributed over $\mathbb{F}^{w \cdot 2^L}$, $\mathbb{F}^{t \cdot 2^{L'}}$ and $\mathbb{F}$, respectively, the consistency check fails with all but negligible probability.

In this hybrid, given inconsistent (e'_u, e'_r) pair, $\mathcal{S}$ aborts except the failure probability of binding, which is also negligible and thus indistinguishable from hybrid $\mathcal{H}_1$. 3) $\mathcal{A}$ uses $e'_u = e_u$ and $e'_r = e_r$ to $\mathcal{F}_{\text{MPVOLUME}}$. $\mathcal{A}$ samples $M[\text{com}_{e_u}]'$ to $\mathcal{S}$. The consistency check passes if and only if $\mathcal{A}$ samples $M[\text{com}_{e_u}]' = K[e'_u] \cdot \mathbf{H}_1 + K[e'_r] \cdot \mathbf{H}_2 + \text{com}_{e_u} \cdot \Delta$, which is equivalent to checking $M[\text{com}_{e_u}]' = M[e'_u] \cdot \mathbf{H}_1 + M[e'_r] \cdot \mathbf{H}_2$ in this hybrid. Thus, the abort probability is indistinguishable from hybrid $\mathcal{H}_1$.

Therefore, this hybrid is identically distributed as $\mathcal{H}_1$.

Corrupted P_B . Let $\mathcal{S}$ access $\mathcal{F}_{\text{C-VOLUME}}$ as an honest P_B and interact with $\mathcal{A}$ as an honest P_A . $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

1. $\mathcal{S}$ emulates $\mathcal{F}_{\text{MPVOLUME}}$ and receives (init) from $\mathcal{A}$. $\mathcal{S}$ holds public commitment of x , $\text{Hcom}_x = (\text{com}_{e_u}, x^* = e_u \cdot \mathbf{H} - x)$. $\mathcal{S}$ sends (init) to $\mathcal{F}_{\text{C-VOLUME}}$. $\mathcal{S}$ receives $\Delta \in \mathbb{F}$ from $\mathcal{A}$.
2. $\mathcal{S}$ emulates $\mathcal{F}_{\text{MPVOLUME}}$ and receives (extend, L, w) from $\mathcal{A}$. $\mathcal{S}$ receives $K[e_u] \in \mathbb{F}^{w \cdot 2^L}$ from $\mathcal{A}$.
3. $\mathcal{S}$ emulates $\mathcal{F}_{\text{MPVOLUME}}$ and receives (extend, L', t) from $\mathcal{A}$. $\mathcal{S}$ receives $K[e_r] \in \mathbb{F}^{t \cdot 2^{L'}}$ from $\mathcal{A}$.
4. $\mathcal{S}$ computes $H_3(M[\text{com}_{e_u}]) = H_3(K[e_u] \cdot \mathbf{H}_1 + K[e_r] \cdot \mathbf{H}_2 + \text{com}_{e_u} \cdot \Delta)$ and sends it to $\mathcal{A}$. $\mathcal{S}$ aborts if $\mathcal{A}$ aborts.
5. $\mathcal{S}$ computes $K[x] = K[e_u] \cdot \mathbf{H} + x^* \cdot \Delta \in \mathbb{F}^m$ and sends $(\Delta, K[x]) \in \mathbb{F}^{m+1}$ to $\mathcal{F}_{\text{C-VOLUME}}$. $\mathcal{S}$ outputs what $\mathcal{A}$ outputs.

We are going to show the simulated execution is indistinguishable from real-world protocol execution.

Hybrid $\mathcal{H}_0$ Same as real-world execution in $\mathcal{F}_{\text{MPVOLUME}}$ -hybrid model.

Hybrid $\mathcal{H}_1$ Same as Hybrid $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{\text{MPVOLUME}}$ and simulates the messages to $\mathcal{A}$.

$\mathcal{S}$ simulates $H_3(M[\text{com}_{e_u}]) = H_3(K[e_u] \cdot \mathbf{H}_1 + K[e_r] \cdot \mathbf{H}_2 + \text{com}_{e_u} \cdot \Delta)$ and sends it to $\mathcal{A}$. In the previous hybrid, an honest P_A computes $M[\text{com}_{e_u}] = M[e_u] \cdot \mathbf{H}_1 + M[e_r] \cdot \mathbf{H}_2$, which holds the property that $M[e_u] = K[e_u] + e_u \cdot \Delta$ and $M[e_r] = K[e_r] + e_r \cdot \Delta$. Consequently, the $M[\text{com}_{e_u}]$

computed by simulator equals the one computed by honest P_A in real world. Thus, this hybrid is identical to the previous one.

This concludes the proof. $\square$

4.2.2.2. Complexity Analysis

We give an analysis of the communication efficiency of dual-LPN based C-VOLE instantiation. As the protocol call $\mathcal{F}_{\text{MPVOLE}}$ for generating $\langle e_u \rangle$ and $\langle e_r \rangle$, it requires MPVOLE instantiations of $e_u \in \mathbb{F}^{w \cdot 2^L}$ and $e_r \in \mathbb{F}^{t \cdot 2^{L'}}$. The protocol also requires a $H_3(\text{M}[\text{com}_{e_u}])$ for consistency check and publishing a commitment of size $n + m$. According to the instantiation of MPVOLE protocol, we have that for sparse vector of length $w \cdot 2^L$, it requires communication overhead of $wL\kappa$. Thus, the total communication overhead of dual-LPN based C-VOLE protocol is $(wL + tL')\kappa + (n + m) \log |\mathbb{F}|$.

4.2.3. PSI from Committed-VOLE

We consider server-output private set intersection in the client–server setting, as defined in Section 2.3 (Figure 2.10). In this chapter, a server interacts separately with multiple clients and receives the intersection in each execution. Any standalone maliciously secure two-party PSI is insufficient here, as the server may use different private sets when interacting with different clients. Introducing C-VOLE addresses this issue by building upon VOLE-based PSI works [13], [14], the most performant PSI protocols currently, and extending them to a C-VOLE-based client–server PSI protocol. The protocol is constructed as follows to securely realize $\mathcal{F}_{\text{PSI}}$ in Figure 2.10. The protocol is shown in Figure 4.3 and its intuition is detailed as follows:

Initially, P_1 computes a vector $\mathbf{p}$ by solving a linear system of its private set elements, and publishes its commitment. Then, any client P_j executes the C-VOLE functionality with P_1 to get a VOLE relation associated with $\mathbf{p}$. Given the VOLE relation, expressed as $\text{M}[\mathbf{p}]_j = \text{K}[\mathbf{p}]_j + \mathbf{p} \cdot \Delta_j$, party P_j uses $\text{K}[\mathbf{p}]_j$ and Δ_j to encode its private set elements and then sends the encodings to P_1 . P_1 uses the corresponding $\text{M}[\mathbf{p}]_j$ to encode its private set elements $x_i \in X, i \in [n]$. Then, P_1 is

Protocol Π_{PSI}

Parameters: Party P_1 with a set $X = (x_1, \dots, x_n) \subset \mathbb{F}^n$. Party P_j with a set $Y_j \subset \mathbb{F}^{n'}$. A random function $\text{row} : \mathbb{F} \times \{0, 1\}^\kappa \rightarrow \{0, 1\}^m$, where $m = O(n) + O(\lambda)$ as defined by OKVS scheme in [14]. A Decode algorithm $\text{Decode} : \mathbb{F}^m \times \mathbb{F} \times \{0, 1\}^\kappa \rightarrow \mathbb{F}$, defined by [14]. A hybrid commitment scheme defined in Chapter 4.2.1.2, such that $(\text{Hcom}_a, \text{Hdecom}_a) \leftarrow \text{HCom}(a)$, $1 \leftarrow \text{Open}(a, \text{Hdecom}_a, \text{Hcom}_a)$, where $a \in \mathbb{F}^m$, $\text{Hcom}_a \in \mathbb{F}^N$, $\text{Hdecom}_a \in \mathbb{F}^{m'}$. Random oracle $H_1 : \mathbb{F} \rightarrow \mathbb{F}$, $H_2 : \mathbb{F} \times \mathbb{F} \rightarrow \{0, 1\}^{\text{out}}$.

Initialize:

1. P_1 samples $r \leftarrow \{0, 1\}^\kappa$ and solves the following systems to get $\mathbf{p} \in \mathbb{F}^m$.

$$\begin{bmatrix} \text{row}(x_1, r) \\ \vdots \\ \text{row}(x_n, r) \end{bmatrix} \mathbf{p}^T = (H_1(x_1), \dots, H_1(x_n))^T$$

2. P_1 computes $(\text{Hcom}_{\mathbf{p}}, \text{Hdecom}_{\mathbf{p}}) \leftarrow \text{HCom}(\mathbf{p})$ and publishes $\text{Hcom}_{\mathbf{p}}$.

Extend: P_1 execute this phase with each party P_j , who possesses set Y_j consisting elements $(y_1, \dots, y_{n'})$.

3. P_j samples $w^s \leftarrow \mathbb{F}$, sends $c^s := H_1(w^s)$ to P_1 .
4. P_1 samples $w^r \leftarrow \mathbb{F}$, sends w^r, r to P_j .
5. P_1 calls $\mathcal{F}_{\text{C-VOLE}}$ with input $(\text{init}, \mathbf{p}, \text{Hdecom}_{\mathbf{p}})$. P_j call $\mathcal{F}_{\text{C-VOLE}}$ with input (init) . The functionality sends $\mathbf{M}[\mathbf{p}] \in \mathbb{F}^m$ to P_1 and $(\Delta, \mathbf{K}[\mathbf{p}]) \in \mathbb{F}^{m+1}$ to P_j , such that $\mathbf{M}[\mathbf{p}] = \mathbf{K}[\mathbf{p}] + \mathbf{p} \cdot \Delta$.
6. P_j sends w^s to P_1 . P_1 aborts if $c^s \neq H_1(w^s)$. Otherwise, both party computes $w = w^s + w^r$.
7. P_j computes $ey_i = F(y_i) = H_2(\text{Decode}(\mathbf{K}[\mathbf{p}], y_i, r) + H_1(y_i) \cdot \Delta + w, y_i)$ for each $y_i \in Y_j$, inserts ey_i to set EY , and sends EY to P_1 .
8. P_1 computes $ex_i = F(x_i) = H_2(\text{Decode}(\mathbf{M}[\mathbf{p}], x_i, r) + w, x_i)$ for each $x_i \in X$, inserts ex_i to set EX . P_1 computes the intersection $Z : \{x_i | x_i \in X, ex_i \in EX \cap EY\}$

Figure 4.3: Secure PSI protocol from Committed-VOLE.

able to compute the intersection by locally comparing the encodings. By leveraging $\mathcal{F}_{\text{C-VOLE}}$ and pre-published commitment of $\mathbf{p}$, this protocol ensures that P_1 consistently reuses $\mathbf{p}$ in the extend phase with different P_j .

The correctness of encoding and computing the intersection aligns with the VOLE-based PSI protocol [13]. Each set element x_i/y_i is mapped to a specific row vector through random function row . P_j encodes its set elements $y_i, i \in [n']$ by applying $\mathbf{K}[\mathbf{p}]$ to these row vectors to compute the inner product and then add $H(y_i) \cdot \Delta$. P_1 encodes its set elements $x_i, i \in [n]$ by computing the inner product of $\mathbf{M}[\mathbf{p}]$ with the row vectors mapped from $x_i, i \in [n]$. According to the computation of $\mathbf{p}$, for each $y_{i'} = x_i, i \in [n], i' \in [n']$, $H(x_i)/H(y_{i'})$ is equivalent to applying the corresponding row

vectors of x_i/y_i to $\mathbf{p}$. Thus, given the VOLE relation of $\mathbf{p}$, both parties are able to encode identical elements into identical encodings. An enhanced Oblivious Key-Value Store (OKVS) scheme from [14] is used here to sample the random function row, solve the linear system of $\mathbf{p}$, and compute the Decode algorithm. We proved that this client–server PSI protocol is maliciously secure in each distributed two-party PSI computation below. We also want to noted that the coin-flipping procedure in generating w is not mandatory in this client–server PSI protocol as it is originally used for achieving the OPRF protocol in [13], which requires uniformity of OPRF output.

Theorem 9. *Protocol Π_{PSI} in Figure 4.3 securely instantiated $\mathcal{F}_{\text{PSI}}$ in $(\mathcal{F}_{\text{C-VOLE}}, H_1, H_2)$ -hybrid model against malicious adversaries.*

Proof. The proof is deferred to Chapter B.6. □

Additionally, even in semi-honest setting, implementing C-VOLE in client–server PSI could bring the correction vector, which is computed from the difference between randomized VOLE vector and the designated vector, to the initialization phase. This optimization ensures the same correction vector is used for all senders, reduces communication overhead by eliminating the need for individual correction vectors for each sender.

4.3. Other Applications

We present more details of other applications that can benefit from C-VOLE. Our design improves the overall performance of commit-and-prove VOLE-ZK, simplifies the communication pattern of VOLE-based generic MPC protocols. They are discussed in the following.

4.3.1. Commit-and-Prove VOLE-ZK

VOLE-ZK refers to a category of private-coin interactive zero-knowledge proofs [45], [81], [82], [102]. One of its extensions is the commit-and-prove VOLE-ZK that proves a statement over publicly committed data [86]. To achieve this, [86] uses a hybrid commitment scheme that is

based on the Merkle tree construction [103]. Additionally, it proposes a proof of commitment opening by generically re-computing the whole Merkle tree in ZK.

For an arbitrary circuit $C(\cdot)$ and a witness $\mathbf{w}$, define a VOLE-ZK that proves the validity of $C(\mathbf{w}) = 1$ without revealing any information about $\mathbf{w}$. Additionally, define a Merkle tree-based commitment scheme $\text{MerkleCom}(\cdot)$ that commits to arbitrary messages. Assume that a prover commits to the witness $\mathbf{w}$ by $c = \text{MerkleCom}(\mathbf{w})$. Given (c, C) , the commit-and-prove VOLE-ZK can be viewed as proving the possession of witness $\mathbf{w}$ such that $c = \text{MerkleCom}(\mathbf{w}) \wedge C(\mathbf{w}) = 1$.

In [86], the proof of the first part requires a non-black-box access to the commitment scheme. Namely, the prover needs to prove the satisfiability of a composition of hash function circuits, which incurs $O(\ell C_{\text{Hash}})$ communication overhead assuming hash function circuits of size C_{Hash} per block and $\mathbf{w} \in \mathbb{F}^\ell$. Our C-VOLE protocol reduces its cost to $O(\kappa \log \ell)$ by replacing the Merkle tree commitment with the LPN-based commitment $\text{HCom}(\cdot)$, which comes with an efficient proof of opening that facilitates the proof of $c = \text{HCom}(\mathbf{w}) \wedge C(\mathbf{w}) = 1$. We demonstrate the concrete performance improvement in Chapter 4.4.3.1.

4.3.2. Multi-Party VOLE in MPC and MVZK

It is common for multi-party protocols to rely on a multi-party VOLE protocol to generate correlated randomness. Examples include generic dishonest-majority MPC protocols [85], [92], [93] and multi-verifier zero-knowledge proofs [94]. In multi-party VOLE, each pair of parties (P_i, P_j) need to execute VOLE for the correlation $M[\mathbf{x}^i]_j^i = K[\mathbf{x}^i]_i^j + \mathbf{x}^i \cdot \Delta^j$. Achieving malicious security in such protocols requires that P_i uses the same $\mathbf{x}^i$ across all VOLE instances it executes with other parties. To ensure this, a costly consistency check is performed at the end of the multi-party VOLE. In prior works, this check involves n -party joint coin-tossing and multiple rounds of peer-to-peer communication and broadcasting, which are significantly constrained by network latency.

Our C-VOLE protocol eliminate this round-heavy consistency check. Namely, P_i first publishes its commitment to $\mathbf{x}^i$. Then, for each $j \in [n] \setminus \{i\}$, P_i interacts with P_j to execute committed VOLE, which ensures the VOLE correlations are consistent with the committed $\mathbf{x}^i$. As long as

parties reach consensus on the commitment, there is no need for a final multi-round verification.

4.4. Performance Evaluation

We implement our proposed C-VOLE protocol and the C-VOLE based client–server PSI protocol based on the work by [14]. We utilize the EMP library [75] for C-VOLE implementation. For the PSI protocol, we integrate the libOT [104] and volePSI [105] libraries for Paxos operations. We instantiate all the protocols with $\mathbb{F} = \mathbb{F}_{2^{128}}$, the computational security parameter $\kappa = 128$, and the statistical security parameter $\lambda = 40$. All experiments were executed on AWS EC2 instances `6a.8xlarge` with 32vCPU and 128GB memory. We tested our schemes in both the LAN and WAN networks, where we set the LAN network with 0 ms latency and 1Gbps bandwidth, and WAN network with 400 ms latency and 50 Mbps bandwidth.

4.4.1. Parameter Selection of Hybrid Commitment

We explain how to choose parameters for a hybrid commitment Hcom_x of $x \in \mathbb{F}^m$, which will influence the performance of C-VOLE as well. Notice that $\text{Hcom}_x = \{\text{com}_{e_u}, x^* = e_u \cdot \mathbf{H} - x\}$ consists of two parts: the LPN-based commitment com_{e_u} of $e_u \in \mathbb{F}^{w \cdot 2^L}$; and a pseudorandom correction vector x^* corrects x from e_u . We first select the parameters used for computing x^* , which hides x with 128-bit security. Our framework is compatible with any LPN coding in the VOLE context [97], [98]. We use the expand-accumulate (EA) code [97] to build the matrix $\mathbf{H} \in \mathbb{F}^{w \cdot 2^L \times m}$, where the expansion parameter C corresponds to the “expand” step in EA coding. This matrix compresses vector $e_u \in \mathbb{F}^{w \cdot 2^L}$ into a pseudorandom vector of length m . Following the conservative constraints in Chapter 4.4.1 and [97], we set the expansion parameter $C = 10$ and the code rate $R = 1/7$ (i.e., $7 * |x| = |e_u|$). Under these settings, we choose the (w, L) pair for $e_u \in \mathbb{F}^{w \cdot 2^L}$ to achieve 128-bit security. The chosen parameters are shown in Table 4.1. Notice that [98]’s attack only applies to the aggressive parameters of EA code but not the conservative ones we use.¹

¹The attack only applies to $\mathbb{F}_2$ while our implementation is based on $\mathbb{F}_2^{128}$. Therefore, we also provide performance evaluation of C-VOLE under aggressive parameters in Chapter B.3.

Input $ \mathbf{x} $	Intermediate sparse vector			Dual LPN commitment			
	$ \mathbf{e}_u $	w	L	$ \mathbf{e}_r $	t	L'	$ \text{com}_{\mathbf{e}_u} (n)$
2^{20}	$7 * 2^{20}$	224	15	1456	91	4	696
2^{24}	$7 * 2^{24}$	224	19	1472	92	4	710
2^{28}	$7 * 2^{28}$	224	23	1488	93	4	723

Table 4.1: Dual-LPN-based hybrid commitment parameter selection for $\mathbf{x}$ of different lengths. Achieves 128-bit security.

Given the (w, L) pair for m , we fix the structure of $\mathbf{e}_u$ for each $\mathbf{x}$. Then, we choose parameters for constructing a commitment $\text{com}_{\mathbf{e}_u}$ with 128-bit security level, where $\text{com}_{\mathbf{e}_u} = \mathbf{e}_u \cdot \mathbf{H}_1 + \mathbf{e}_r \cdot \mathbf{H}_2$ for the commitment from the dual LPN assumption, to achieve statistically binding and computationally hiding commitment of $\mathbf{e}_u \in \mathbb{F}^{w \cdot 2^L}$. According to Theorem 6, we select the (n, t, L') pair for dual LPN commitment. For each $\mathbf{x} \in \mathbb{F}^m$ of different lengths, we compute groups of parameters (w, L, n, t, L') and select the one that minimizes the communication cost of the C-VOLE protocol from the dual LPN assumption, mentioned in Chapter 4.2.2.2 for implementation. We give the commitment parameter selection that achieving 128 bit security for $\mathbf{x} \in \{\mathbb{F}^{2^{20}}, \mathbb{F}^{2^{24}}, \mathbb{F}^{2^{28}}\}$ in Table 4.1.

4.4.2. Efficiency of Committed-VOLE

Our C-VOLE protocol includes two phases: computing a one-time commitment on the VOLE sender's side (P_A in the protocol) for chosen input vector $\mathbf{x}$ and a compute phase that can be executed between the sender and each individual VOLE receiver (P_B in the protocol). We show the time consumption of the two phases under LAN and WAN with different input vector lengths m and different threads in Figure 4.4a. The communication overhead is shown in Figure 4.4b. The detailed analysis is as follows.

One-time Commitment: In this phase, the sender, who holds the input vector $\mathbf{x}$, generates an LPN-based hybrid commitment of $\mathbf{x}$ and publishes it to all the parties. We tested this process using 4 threads and 16 threads. The results show that it takes 0.84 s for $m = 2^{20}$ and 382.39 s for $m = 2^{28}$ using 4 threads, while it takes 0.33 s for $m = 2^{20}$ and 160.62 s for $m = 2^{28}$ using

16 threads. The communication overhead in this phase is publishing the hybrid commitment of x , which costs 16.78 MB for $m = 2^{20}$ and 4294.9 MB for $m = 2^{28}$, dominating the entire communication of the C-VOLE protocol. As the one-time commitment is published and applicable to each individual receiver interacting with the sender to generate C-VOLE of x , the time consumption for the computation can be amortized to a very small portion of the overall computation. Additionally, the commitment computation is computed locally on the sender's side and is unaffected by network settings.

Compute: We tested the compute phase for the sender generating VOLE of the committed x with one receiver. Note that the performance of the compute phase is influenced by the parameters chosen in Chapter 4.4.1, specifically the Hamming weight and the block size of the sparse vectors in executing Π_{MPVOLE} . For $m = 2^{20}$, the compute phase takes 0.81 s under a LAN network and 9.31 s under a WAN network with a bandwidth of 50 Mbps when using 4 threads. With 16 threads, the compute phase takes 0.62 seconds under LAN network and 8.20 seconds under WAN network. For $m = 2^{28}$, the compute phase with 4 threads takes 407.6 s under a LAN network and 438.8 s under a WAN network. With 16 threads, the corresponding times are 187.22 s under a LAN network and 191.82 s under a WAN network. In terms of communication overhead, which is dominated by the cost of Π_{MPVOLE} , the cost is 1.72 MB for $m = 2^{20}$ and 1.78 MB for $m = 2^{28}$. The communication overhead does not increase significantly with the increase of m , benefiting from the sublinear online communication of our intermediate structured sparse LPN commitment and corresponding consistency check construction.

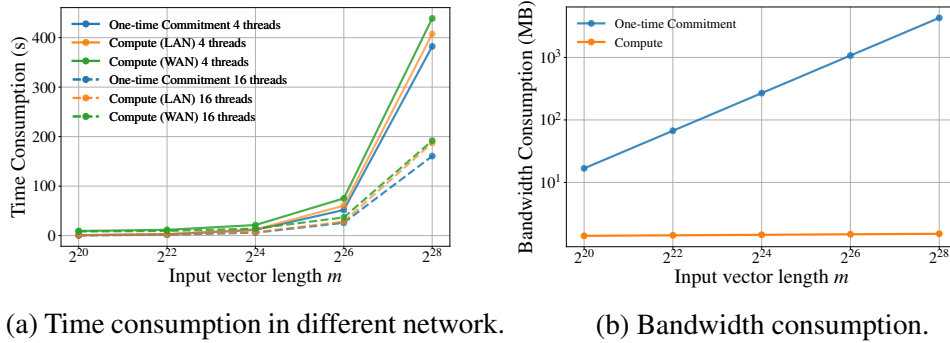


Figure 4.4: Performance of C-VOLE protocol in LAN and WAN setting.

4.4.3. Comparison with Prior Work

4.4.3.1. Comparison with C2A Conversion.

We compare the proposed C-VOLE protocol with the public commitment to private authentication conversion (C2A) in [86], which is used for commit-and-prove zero-knowledge proofs. The C2A conversion in [86] assumes 64-bit data block, while our protocol is evaluated under field $\mathbb{F}_{2^{128}}$, implemented as data block of 128-bit. The time consumption and communication overhead are summarized in Table 4.2 and Table 4.3 and are analyzed below.

Schemes	50 Mbps	200 Mbps	500 Mbps	1 Gbps
C2A [86]	$45\mu s$	$45\mu s$	$44\mu s$	$44\mu s$
C-VOLE	$1.61\mu s$	$1.59\mu s$	$1.57\mu s$	$1.53\mu s$

Table 4.2: Time consumption compared with [86]. With different network settings, we tested C-VOLE of 128-bit data block to compare with C2A of 64-bit data block. Both evaluations are with single thread.

Schemes	2^{20} block	2^{24} block	2^{28} block
C2A [86]	114.52 MB	1829.42 MB	29317.12 MB
C-VOLE	18.49 MB	270.18 MB	4296.74 MB

Table 4.3: Bandwidth consumption compared with [86]. we tested the bandwidth consumption of C-VOLE and C2A under varying input blocks. For each C-VOLE input block, we used a 128-bit data block, while for C2A, a 64-bit data block was employed for comparison.

Wallclock time. C2A conversion is the bottleneck of computing commit-and-prove ZK. As tested from [75], the C2A takes $45\mu s$ per element while the zero-knowledge proof only takes less than $0.1\mu s$ per gate [102]. We tested ours C-VOLE computation time for each block under different network setting, each result is amortized from the performance of vector of length 2^{24} . The results show that our scheme requires $1.61\mu s$ computation for 128-bit data block, which is $28\times$ faster than the $45\mu s$ computation for 64-bit data block in [86]. Note that a follow-up LPZKv2 [106] provides at most a $2\times$ improvement over QuickSilver [102]. The advantage of C-VOLE compared to Mystique remains significant even if its underlying ZKP is replaced from QuickSilver to LPZKv2.

Communication overhead. We tested the communication overhead of the C2A conversion and our proposed C-VOLE for input vector length $m = \{2^{20}, 2^{24}, 2^{28}\}$. The communication overhead includes the published commitment, the interactive generation of VOLE, and the proof of input vector consistency. C2A conversion requires 114.52 MB of communication for 2^{20} blocks while ours C-VOLE only requires 18.49 MB. For 2^{28} blocks, the C2A consumes 29 GB communication overhead and ours C-VOLE consumes 4.2 GB. Note that in the C2A conversion, the commitment size is small, and the proof of consistency dominates the entire communication cost. In contrast, for our C-VOLE, the commitment size constitutes the majority of the communication overhead. This means that when executing C2A operations multiple times on the same publicly committed data, ours C-VOLE protocol is much more communication-efficient.

4.4.3.2. Comparison with CVOLE with Input-revealing Identifiable-abort (IRIA).

We compare the committing VOLE proposed in [95] with our proposed scheme. The committing VOLE with input-revealing identifiable-abort is designed having both parties commit to their input and reveal them to prove consistency in case of an abort. To ensure a fair comparison with our work, we only require the sender to commit to its input vector and reveal it for consistency checks. Since there is no implementation available for [95], we analyze and compare its communication overhead with ours theoretically.

The C-VOLE used in [95] can be divided into two steps according to their paper: 1) Generate oblivious transfer (OT) based on PVW OT [107] and extend it to VOLE using OT-multiplication protocols [63], [72]; 2) Run a sigma protocol to reveal the input to the other party and prove consistency of the input with previous transferred messages for each OT. For step 1), each DDH-based PVW OT requires $8 \log |\mathbb{G}|$ communication. To extend it to VOLE with vector length m , the extension [63] requires $m \cdot (\log |\mathbb{F}| + \kappa + 2\lambda)$ execution of OTs, along with additional consistency check of the sender's input for malicious security, which doubles the the number of OTs. For step 2), each OT requires a communication overhead of $4 \log |\mathbb{F}| + 8 \log |\mathbb{G}|$ for the sigma protocol to prove consistency with the commitment (transferred message). The total communication overhead

for the C-VOLE with sender-committed input and revealing is $2 \cdot m \cdot (\log |\mathbb{F}| + \kappa + 2\lambda) \cdot (8 \log |\mathbb{G}| + 4 \log |\mathbb{F}| + 8 \log |\mathbb{G}|)$. In contrast, ours communication overhead is $(wL + tL')\kappa + (n + m) \log |\mathbb{F}|$ as analyzed in Chapter 4.2.2.2. For $m = 2^{20}$, $\mathbb{G}$ with order 128 bits, the communication cost for [95] is 225 GB, whereas ours is 16.84 MB.

4.4.4. Efficiency of C-VOLE based PSI and Comparison

In this chapter, we evaluate the performance of our proposed C-VOLE based client–server PSI and compare it with the state-of-the-art two party PSI [14] and the state-of-the-art client–server PSI protocol [6].

We integrate C-VOLE into VOLE-based PSI protocol, Blazing PSI [14], which is also the state-of-the-art two-party PSI protocol, by replacing VOLE with C-VOLE to achieve malicious security in the client–server setting. As shown in Figure 4.3, in the C-VOLE based client–server PSI protocol, the C-VOLE sender acts as the receiver (server) of the client–server PSI protocol and computes a publicly committed vector $\mathbf{p}$ for initialization. The server (receiver) then computes the extend phase of client–server PSI with each individual client (sender) to get the intersection. The initialization phase includes a Paxos encoding operation and the computation of a one-time commitment. The extend phase consists of the following steps: 1) Both the server and the client generate the VOLE correlation of the committed $\mathbf{p}$; 2) The client sends the encoding of its set elements to the server; 3) The server performs local encoding of its set and compares the two encodings to determine the intersection with each client.

4.4.4.1. Comparison with VOLE based PSI.

The only difference between ours C-VOLE based PSI and the VOLE-based PSI [14] is that we use committed-VOLE instead of VOLE. Notice that [14] proposed two versions of implementation, one uses VOLE with slightly faster computational overhead, the other one uses subfield-VOLE with reduced communication overhead. We compare with the faster version (VOLE-based) and measure both the computation and the communication overhead by fixing client’s set to 2^{10} el-

ements and scaling the server’s set to $\{2^{20}, 2^{22}, 2^{24}\}$. The detailed comparison are presented in Table 4.4 and Table 4.5.

Set size (m, n)		$(2^{20}, 2^{10})$	$(2^{22}, 2^{10})$	$(2^{24}, 2^{10})$
Blazing PSI [14] (s)		0.41	2.44	9.89
C-VOLE PSI	Init (s)	0.78	3.34	12.37
	Online (s)	1.08	3.51	14.69

Table 4.4: Time consumption compared with [14]. (m, n) represents the server’s set size m , and client’s set size n . PSI Init is a one-time computation and PSI online is executed between the server and each individual client. The experiments are tested with 16 threads.

Set size (m, n)		$(2^{20}, 2^{10})$	$(2^{22}, 2^{10})$	$(2^{24}, 2^{10})$
Blazing PSI [14] (MB)		23.83	92.34	366.55
C-VOLE PSI	Init (MB)	21.94	88.41	353.94
	Online (MB)	1.73	1.75	1.76

Table 4.5: Communication overhead compared with [14]. (m, n) represents the server’s set size m , and client’s set size n . The experiments are executed with 16 threads.

Compared with Blazing PSI [14], the results show that our C-VOLE based PSI consumes approximately $2\times$ more online running time but requires significantly less online communication overhead. This difference can be attributed to the fact that Blazing PSI employs the default VOLE implementation in the library LibOTe using EC code. In contrast, ours uses EA code with parameter $R = \frac{1}{7}$, $C = 10$ and specifically selects the structure of e_u and e_r for both security and performance in Chapter 4.4.1. Moreover, our C-VOLE based PSI has the property of having the initialization phase performed locally in advance to enhance performance. In the client–server PSI setting, when a server executes PSI with multiple clients, the initialization can be amortized to a very small amount of time. The communication can also be managed by other efficient networks such as CDN or P2P, further enhancing performance.

4.4.4.2. Comparison with Prior Work in Client–Server PSI.

There’s only one prior work that considers PSI in the client–server setting [6]. In this work, the server commits to its private input and executes PSI with each individual client using an oblivious verifiable unpredictable function (OVUF). This work allows the client to obtain the intersection, whereas our protocol enables the server to obtain the intersection. We compare our work with this work despite the slight difference in output, as it provides the relevant baseline for performance in client–server setting. We set the server’s set size to $\{2^{20}, 2^{24}, 2^{28}\}$ elements and the client’s set size to $\{2^{10}, 2^{17}, 2^{20}\}$ elements, and present the performance results below.

Set size (m, n)	C-VOLE PSI		OVUF-PSI [6]	
	Init (s)	Online (s)	Init (s)	Online (s)
$(2^{20}, 2^{10})$	0.78	1.03	6.24	0.59
$(2^{20}, 2^{17})$	0.79	1.08	6.24	67.36
$(2^{20}, 2^{20})$	0.79	1.23	6.24	538.88
$(2^{24}, 2^{10})$	12.37	14.35	98.14	0.59
$(2^{24}, 2^{17})$	12.37	14.69	98.14	67.36
$(2^{24}, 2^{20})$	15.65	16.57	98.14	538.88
$(2^{28}, 2^{10})$	458.17	455.55	1596.44	0.59
$(2^{28}, 2^{17})$	458.10	499.04	1596.44	67.36
$(2^{28}, 2^{20})$	465.23	523.17	1596.44	538.88

Table 4.6: Time consumption compared with [6]. (m, n) represents the server’s set size m , and the client’s set size n . PSI Init is a one-time computation and PSI Online is executed between the server and each individual client. All experiments are executed with 16 threads. The online computation time for [6] with a client size of 2^{20} is derived from other tested results based on linear properties.

Wallclock time. We evaluate the time consumption for executing PSI between a server and a client using 16 threads. As shown in Table 4.6, the initialization of both schemes scales linearly with the server’s set size, as it computes commitment for the server’s private set elements. Ours C-VOLE PSI requires 465.23 ms for a server with 2^{28} elements, while OVUF-PSI requires 1596.44 ms. For the online phase, our C-VOLE based PSI includes generating the VOLE correlation of the committed vector (with a length linear to the server set size) and computing the client’s side encoding to the server. The time consumption scales roughly linearly with the server’s set size,

with a small additional increase as the client set size grows. In contrast, the OVUF-PSI scales linearly with the client set size. For a server's set size of 2^{20} , ours C-VOLE PSI requires 1.03 s for client size 2^{10} and 1.23 s for client size 2^{20} . In comparison, OVUF-PSI requires 0.59 s for client size 2^{10} and 538.88 s for client size 2^{20} .

Communication overhead. For communication overhead, the initialization phase of both schemes requires the server to publish the commitment of its set elements. The communication overhead scales linearly with the server set size. However, in OVUF-PSI, each element in $\mathbb{F}_{2^{128}}$ is hashed to 64-bit, while ours C-VOLE PSI can not use such hashing. The communication overhead of our initialization phase scales linearly with the length of p , which is proportional to the server's set size. Consequently, our C-VOLE PSI requires roughly 2.6 times more communication overhead than the OVUF-PSI during initialization. In the online phase, C-VOLE PSI includes generating the C-VOLE correlation of p and having the client send back its set encodings. The online communication overhead is dominated by the client's set size, requires 1.73 MB for a client size of 2^{10} and 18.49 MB for a client size of 2^{20} , with a fixed server size 2^{20} . This significantly outperforms the OVUF-PSI online phase.

Set size (m, n)	C-VOLE PSI		OVUF-PSI [6]	
	Init (MB)	Online (MB)	Init (MB)	Online (MB)
$(2^{20}, 2^{10})$	21.94	1.73	8.38	63.23
$(2^{20}, 2^{17})$	21.94	3.81	8.38	8091
$(2^{20}, 2^{20})$	21.94	18.49	8.38	64747
$(2^{24}, 2^{10})$	353.94	1.76	134.18	63.23
$(2^{24}, 2^{17})$	353.94	3.84	134.18	8091
$(2^{24}, 2^{20})$	353.94	18.52	134.18	64747
$(2^{28}, 2^{10})$	5671.63	1.79	2147	63.23
$(2^{28}, 2^{17})$	5671.63	3.87	2147	8091
$(2^{28}, 2^{20})$	5671.62	18.55	2147	64747

Table 4.7: Communication overhead compared with [6]. (m, n) represents the server's set size m , and client's set size n . The online communication time for [6] with a client size of 2^{20} is derived from other tested results based on linear properties.

CHAPTER 5

Communication-Efficient Private Join and Compute over Distributed Input Sets

In this chapter, we present a practical system for approximate PJC in which the two input sets can be contributed by many clients. Figure 5.1 illustrates the overall workflow of our protocol. We assume a cryptographic service provider (CSP) during the initialization and output phases; otherwise, only the aggregator server (Agg) needs to be online, and each client sends a single message to the aggregator.

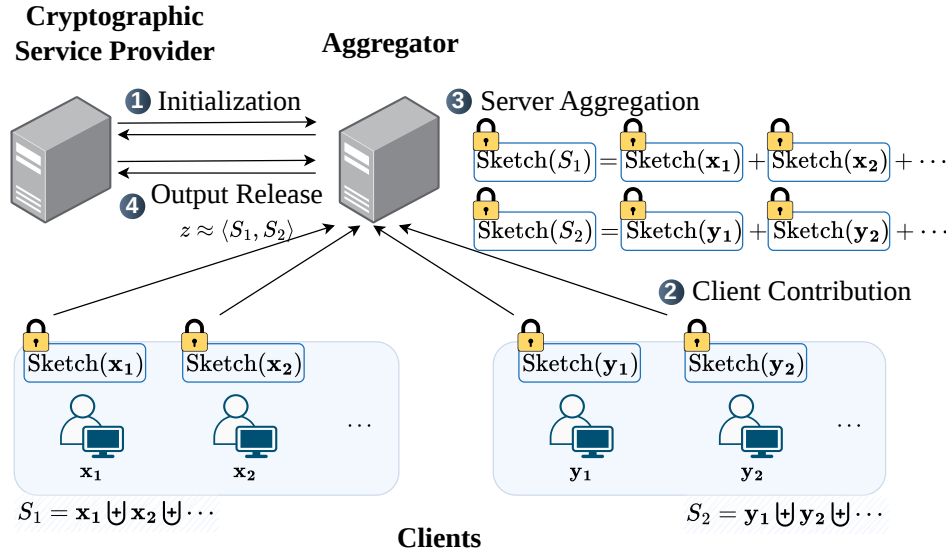


Figure 5.1: System model of Private Join and Compute over Distributed Input Sets. Two non-colluding servers, the Aggregator and the Cryptographic Service Provider (CSP), serve many clients. ① During initialization, the two servers jointly generate encrypted sketching randomness; the CSP can then go offline. ② Each client P_i locally computes an encrypted sketch of its input vector ($\mathbf{x}_i$ for set S_1 , or $\mathbf{y}_i$ for set S_2) and sends it to the Aggregator in a single message. ③ The Aggregator homomorphically aggregates all client sketches into $\llbracket \text{Sketch}(S_1) \rrbracket$ and $\llbracket \text{Sketch}(S_2) \rrbracket$. ④ Finally, the two servers jointly compute the approximate inner product $z \approx \langle S_1, S_2 \rangle$ via threshold decryption and 2PC.

1. **General framework for PJC over distributed sets.** We design a general framework for approximate PJC over distributed input sets. We observe that PJC is most useful for computing

private aggregates and statistics, where the underlying data can naturally be viewed as samples from an underlying distribution and thus approximation is often acceptable. This motivates the use of the AMS sketch [39], which enables approximate PJC in plaintext with very small communication. Building on this observation, we let each client use encrypted common randomness to homomorphically compute a local encrypted sketch and send a single message to Agg, which efficiently aggregates all client contributions. This design ensures that the randomness used by the sketch is never revealed, while keeping both client-server and server-server communication low.

2. **HE-friendly sketch construction.** A naive instantiation of the AMS sketch incurs prohibitive cost when homomorphically expanding the randomness required for sketch evaluation. To obtain a practical protocol, we develop a new AMS-style sketch construction that achieves the same level of accuracy while using only structured randomness that can be evaluated in merely *3 levels of homomorphic multiplication*. Our new sketch requires a significantly more intricate analysis, since the structured randomness we use is weaker than the weakest previously analyzed variant of the AMS sketch.
3. **Protecting the output using differential privacy for free.** Prior work has shown that the output of PJC may already be vulnerable to membership-inference attacks [108]. We therefore augment our framework with an efficient mechanism for releasing a differentially private output, incurring little additional protocol overhead while preserving output utility.
4. **Implementation and benchmark.** We implement and benchmark our protocol to evaluate its accuracy, client scalability, and server performance. Compared with prior solutions, our protocol is well-suited to a wide range of deployment settings. In particular, the server cost is under 10 seconds regardless of the number of clients; the client’s communication is non-interactive and only about 250 KB for any set size; the client’s per-element computation is around 30 ms but is embarrassingly parallelizable.

Our framework is more general than existing PJC protocols and can also realize the standard

two-party PJC functionality. In particular, suppose P_1 and P_2 hold datasets $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. Then P_1 can act as both CSP and a client holding $\mathcal{S}_1$, while P_2 can act as both Agg and a client holding $\mathcal{S}_2$. This yields a two-party approximate PJC protocol with sublinear communication. We present our protocol in the standard simulation-based paradigm by instantiating $\mathcal{F}_{\text{PJC}}$ (Figure 2.11) in the $(\mathcal{F}_{\text{Rand}}, \mathcal{F}_{\text{ThSWHE}}, \mathcal{F}_{\text{2PC}})$ -hybrid model.

5.1. Protocol Intuition

We start with our system setup and security model, also depicted in Figure 5.1. Our framework assumes two non-colluding servers: an Aggregator (Agg) and a Cryptographic Service Provider (CSP), serving w clients $P_1, \dots, P_w$. We adopt the CSP+Aggregator architecture introduced by Nikolaenko et al. [109] for privacy-preserving matrix factorization: CSP only needs to be online to interact with Agg during the initialization phase and the finalization phase to obtain output; once setup completes, CSP can go offline and clients send a single message directly to Agg. We allow a semi-honest adversary to corrupt one of the servers (CSP or Agg) and a subset of the clients; the corrupted server learns nothing beyond what the ideal functionality reveals to it (in particular, when Agg is corrupted, it learns the final approximate output). Our proposed protocol securely instantiates the ideal functionality in Figure 2.11.

5.1.1. A New Framework for Distributed PJC

Motivating Example. We start from a simple observation that the desired functionality can be achieved in plaintext with small communication using sketches. We take the AMS sketch [39] as an example. Given a multi-set $\mathcal{X}$ over $[N]$, one can always encode it into a frequency vector $\mathbf{x} \in \mathbb{Z}^N$. The basic AMS sketch is defined as $X = \langle \mathbf{x}, \mathbf{r} \rangle$ with a uniformly sampled $\mathbf{r} \leftarrow \{-1, +1\}^N$. It is shown that X^2 serves as an unbiased estimator of $\|\mathbf{x}\|_2^2$, with variance at most $O(\|\mathbf{x}\|_2^4)$. To improve accuracy to $\mathcal{E}$, one can take the average of $k = O(1/\mathcal{E}^2)$ independent copies. Similarly, let $X = \langle \mathbf{x}, \mathbf{r} \rangle$ and $Y = \langle \mathbf{y}, \mathbf{r} \rangle$ be AMS sketches computed using the same randomness $\mathbf{r}$. Then the product XY is an unbiased estimator of the inner product $\langle \mathbf{x}, \mathbf{y} \rangle$ and has variance bounded by

$$O(\|\mathbf{x}\|_2^2 \|\mathbf{y}\|_2^2).$$

AMS is a linear sketch in that computing the sketch X only involves linear operations over $\mathbf{x}$, given a fixed $\mathbf{r}$. Therefore, distributed PJC can be achieved with small communication in plaintext as follows. A server publishes a random vector $\mathbf{r}$ as described above. Each client with input vector $\mathbf{x}_i$ computes and sends $X_i = \langle \mathbf{x}_i, \mathbf{r} \rangle$ to the server. The server aggregates these values as $X = \sum_{i \in \mathcal{S}_1} X_i$, which corresponds to the sketch of the multiset union of all inputs, as $\sum_{i \in \mathcal{S}_1} X_i = \langle \sum_{i \in \mathcal{S}_1} \mathbf{x}_i, \mathbf{r} \rangle$. Similarly, for vector $\mathbf{y}_j$, the server obtains $Y = \sum_{j \in \mathcal{S}_2} Y_j$ from sketches $Y_j = \langle \mathbf{y}_j, \mathbf{r} \rangle$. Finally, the server can compute an approximation of the inner product $XY \approx \langle \sum_{i \in \mathcal{S}_1} \mathbf{x}_i, \sum_{j \in \mathcal{S}_2} \mathbf{y}_j \rangle$, optionally averaging over k independent repetitions to improve accuracy.

The use of the same randomness $\mathbf{r}$ is crucial for correctness, but is also a curse when we want to turn it into a privacy-preserving protocol. First of all, since $\mathbf{r}$ is public and all parties submit their sketch to the server, the server can easily perform a dictionary attack on each client's set and even recover it when it is small. One could possibly use two-party computation to securely perform the work of the server so that only the final output is revealed but even given just $\mathbf{r}$ and the final output, a dictionary attack can still be performed on the union of clients' sets.

Our Framework. In this chapter, we decide to keep $\mathbf{r}$ private all the time so that none of the parties know the randomness and thus prevent the above attack. It roughly works as follows, ignoring some technical details.

1. **Initialization.** Two servers perform a threshold key generation protocol for a homomorphic encryption (HE) scheme so that at the end of the protocol, they have the public key and the evaluation key. Agg initializes two collections of encrypted sketches $\{\llbracket c_1^\ell \rrbracket\}^{\ell \in [k]}$ and $\{\llbracket c_2^\ell \rrbracket\}^{\ell \in [k]}$, where each entry is a fresh encryption of 0, representing $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. Two servers jointly generate encryptions of k random seeds $\{\llbracket s^\ell \rrbracket\}^{\ell \in [k]}$, where the seeds are jointly generated and hidden from both servers, and publish them to all clients.
2. **Client Contribution.** For each client with input vector $\mathbf{x}$, the client homomorphically computes

the ciphertext $\llbracket X^\ell \rrbracket = \langle G(\llbracket s^\ell \rrbracket), \mathbf{x} \rangle$, where $G : \{0, 1\}^n \rightarrow \{-1, 1\}^N$ is an expanding function, e.g., a pseudorandom generator (PRG). The client sends $(p, \{\llbracket X^\ell \rrbracket\}_{\ell \in [k]})$ to Agg to indicate that it will contribute to set $\mathcal{S}_p$, where $p \in \{1, 2\}$.

3. **Server Aggregation.** Upon receiving $(p, \{\llbracket X^\ell \rrbracket\}_{\ell \in [k]})$ from a client, Agg homomorphically update $\llbracket c_p^\ell \rrbracket \leftarrow \llbracket c_p^\ell \rrbracket + \llbracket X^\ell \rrbracket$ for all $\ell \in [k]$, which effectively performs a multiset union.
4. **Output Release.** Agg and CSP perform threshold decryption on encrypted sketches $\{\llbracket c_1^\ell \rrbracket, \llbracket c_2^\ell \rrbracket\}_{\ell \in [k]}$ to obtain additive secret shares of the corresponding plaintexts. They then invoke a generic $\mathcal{F}_{2PC}$ functionality to securely compute $Z = \sum_{\ell \in [k]} c_1^\ell c_2^\ell$ and the final result is Z/k .

This protocol is highly efficient in communication: the communication between the two servers, the size of the published seeds, and the message from each client are all $O(k)$. The advantage in communication crucially relies on the use of homomorphic encryption to expand the seed. Next, we discuss the exact computational cost of this step under various instantiations.

5.1.2. Why Classic Constructions Fail

The above framework yields secure, sublinear-communication protocols, but the computational cost of expanding randomness can be high. We discuss some plausible choices. As a summary, we have the flexibility in choosing G , but the output should be $\{-1, +1\}^N$ for use in the next step. In practice, it suffices to generate outputs in $\{0, 1\}^N$ and map them to $\{-1, +1\}^N$ via $\mathbf{b} \mapsto 1 - 2\mathbf{b}$. As a first optimization, if each output bit of $G(s)$ can be independently computed, one can compute

$$\llbracket X \rrbracket = \langle G(\llbracket s \rrbracket), \mathbf{x} \rangle = \sum_{i \in \mathcal{X}} G(\llbracket s \rrbracket)[i],$$

where we avoid fully expanding to an N -length vector and instead compute only the required randomness. This property is satisfied by, e.g., local PRGs and the counter-mode instantiation from PRF. We reviewed classic ideas from both theory and cryptography, but we found that existing possible instantiations of G all suffer from large seed sizes and/or high computational cost for homomorphically expanding the seed.

Goldreich PRG. Homomorphic evaluation of PRGs has recently been explored in a different context [110], using Goldreich-style random local PRGs [111] instantiated with XOR-MAJ_{4,7}. Such PRGs require a seed of length $\sqrt{N}$ to produce $O(N)$ bits; thus, this instantiation requires a seed size of $\Omega(\sqrt{N})$. It is also conjectured that reducing the seed length would require much deeper circuits.

PRG from primal LPN. Another plausible solution discussed in [112] was to use LPN-based PRGs. The output is defined as

$$G(\mathbf{s}, i_1, \dots, i_\ell) = \mathbf{A} \cdot \mathbf{s} + \mathbf{e} ,$$

where $\mathbf{s}$ is a short random seed, $\mathbf{A}$ is a public sparse matrix over $\mathbb{F}_2$, and the regular noise vector $\mathbf{e} = (\text{OneHot}(i_1), \dots, \text{OneHot}(i_\ell))$ is the concatenation of small vectors of length N/ℓ , each as the one-hot encoding of index i . The seed size of this LPN-based PRG is $|\mathbf{s}| + |i_1| + \dots + |i_\ell|$, which is lower bounded by $\Omega(\sqrt{N})$ due to the security requirement [48] that $|\mathbf{s}|\ell > N$.

PRG from dual LPN. A related approach is to use a dual-LPN-based PRG, whose output is defined as

$$G(i_1, \dots, i_\ell) = \mathbf{H} \cdot \mathbf{e} ,$$

where $\mathbf{H}$ is a uniformly random parity-check matrix over $\mathbb{F}_2$, and $\mathbf{e} = (\text{OneHot}(i_1), \dots, \text{OneHot}(i_\ell))$. This construction has a small seed, but the main drawback is the prohibitively large circuit size/depth required to compute $\mathbf{H} \cdot \mathbf{e}$ given a dense public matrix $\mathbf{H}$, especially in the context of HE, where Free-XOR is not available. In this setting, Boolean XOR gates must be implemented using arithmetic circuits, causing each XOR operation to incur the same cost as an AND gate.

4-wise independent randomness. It is well-known that the AMS sketch can keep the asymptotic accuracy even when $\mathbf{r}$ is 4-wise independent uniform, which means that any subset of 4 values is indistinguishable from uniform. The classical instantiation is to pick uniform values $a, b, c, d \in \mathbb{Z}_p$

for some prime $p \geq N$ and define $G : [N]^4 \rightarrow \{0, 1\}^N$ such that

$$G(a, b, c, d)[i] = (a \cdot i^3 + b \cdot i^2 + c \cdot i + d) \bmod p \bmod 2.$$

However, computing $G(a, b, c, d)[i]$ can be slow in practice under HE, since the computation is over $\mathbb{Z}_p$ and requires a modular operation from $\mathbb{Z}_p$ to $\mathbb{Z}_2$ for each output bit, which is very expensive.

5.1.3. Warm-Up Solution: HE-Friendly AMS Sketch

As discussed above, we want a G that is friendly for HE evaluation and has sublinear seed size. Here, we first present a warm-up solution that generates 4-wise independent randomness but only requires **2** homomorphic multiplications for each element. We use this to show that an HE-aware design can significantly improve the state-of-the-art solution.

This warm-up solution has a seed size of $4\sqrt{N}$ bits consisting of three independent uniform Boolean vectors $\mathbf{c} \in \{0, 1\}^{\sqrt{N}}$, $\mathbf{d} \in \{0, 1\}^{\sqrt{N}}$, $\mathbf{e} \in \{0, 1\}^{2\sqrt{N}}$, and we define the expansion as

$$G(\mathbf{c}, \mathbf{d}, \mathbf{e})[i \cdot \sqrt{N} + j] = \mathbf{c}[i] \oplus \mathbf{d}[j] \oplus \mathbf{e}[i + j],$$

for any $i, j \in [\sqrt{N}]$.

We now show that $G(\mathbf{c}, \mathbf{d}, \mathbf{e})$ is four-wise independent. Since every bit of the output can be viewed as XOR of some number of uniform and independent bits, we just need to show that for any four distinct indices $(i_1, j_1), (i_2, j_2), (i_3, j_3), (i_4, j_4)$, no subset of outputs deterministically XORs to zero. That is,

$$\bigoplus_{k=1}^{\ell} G(\mathbf{c}, \mathbf{d}, \mathbf{e})[i_k \cdot \sqrt{N} + j_k] \equiv 0 \text{ for } \ell = 2, 3, 4.$$

The two-output case is trivial. In the three-output case, three outputs of G involve 9 bits (possibly with repetitions), which cannot XOR to zero deterministically. For the XOR of four outputs to be identically zero, the first components (i_1, i_2, i_3, i_4) would need exactly two distinct values, each appearing twice, and similarly for the second components (j_1, j_2, j_3, j_4) . This implies that the four

relevant positions of e , namely $\{i_1 + j_1\}_{k \in [4]}$, must contain at least three distinct values and therefore cannot XOR to zero deterministically. Therefore, G is four-wise independent. We provide a complete proof in Appendix C.3.

This solution is faster than all instantiations in Section 5.1.2 as it only requires 2 homomorphic multiplications per element in the set. However, the seed size is $O(\sqrt{N})$, which is still undesirable. It is possible to generalize this approach to further reduce the seed size, but its concrete computational cost increases quickly; therefore, we take a different route in our final solution.

5.1.4. AMS Sketch with Imperfect Randomness

If the solution above is about designing a better randomness generation process that is also HE-friendly at the same time, our final solution takes a step further in co-designing between sketching algorithm and cryptography.

A Cheap and Unbiased Estimator. Our starting point is the LPN-based PRG solution:

$$G(\mathbf{s}, i_1, \dots, i_\ell) = \mathbf{A} \cdot \mathbf{s} + e,$$

where $e = (\text{Unary}(i_1), \dots, \text{Unary}(i_\ell))$. This solution is overkill, since the AMS sketch does not require full randomness. Thus, we try to explore ways to reduce its structure while maintaining the accuracy of the sketch. We do not like e , which is expensive to expand in HE, and we do not like the fact that the seed size cannot be small.

To this end, we construct an expansion function as follows. We first sample a short private seed $\mathbf{s} \in \{0, 1\}^n$ uniformly at random, and then independently sample a public matrix $\mathbf{A} \in \{0, 1\}^{N \times n}$ such that each row has Hamming weight $\text{HW}(\mathbf{A}_i) = d$ for some constant d , and $\mathbf{A}_i \neq \mathbf{A}_j$ for all $i \neq j$. The output random vector $\mathbf{r} \in \{-1, +1\}^N$, as required for the AMS sketch, is then obtained via

$$\mathbf{r} = (-1)^{\mathbf{A} \cdot \mathbf{s} \bmod 2}.$$

Note that this construction is computationally cheap as the number of homomorphic multipli-

k	100	200	400	1000	2500	10000
AMS + uniform r	0.055	0.041	0.028	0.017	0.011	0.006
AMS + our r	0.056	0.039	0.028	0.017	0.011	0.005

Table 5.1: Approximation accuracy comparison under different k . Accuracy is measured as median over 1000 executions.

cations to compute any bit of $\mathbf{A} \cdot \mathbf{s}$ is linear in d . More precisely, since $\mathbf{A}$ is public, we can compute in HE

$$r[i] = 1 - 2 \cdot \left(\bigoplus_{j \in [n] \text{ s.t. } \mathbf{A}_i[j]=1} s[j] \right).$$

Moreover, the solution is unbiased since we require that all rows in matrix $\mathbf{A}$ are different. Indeed, we have

$$\mathbb{E}[X^2] = \mathbb{E} \left[\left(\sum_i \mathbf{x}[i] \cdot r[i] \right)^2 \right] = \sum_i \mathbf{x}[i]^2 + \sum_{i \neq j} \mathbf{x}[i] \mathbf{x}[j] \mathbb{E}[r[i] r[j]].$$

In our case, $\mathbf{r} = (-1)^{\mathbf{A} \cdot \mathbf{s}}$ is not fully independent: although they are pair-wise independent, higher-order correlations remain.

$$\mathbb{E}[r[i] r[j]] = \mathbb{E} \left[(-1)^{\mathbf{A}_i \cdot \mathbf{s}} \cdot (-1)^{\mathbf{A}_j \cdot \mathbf{s}} \right] = \mathbb{E} \left[(-1)^{(\mathbf{A}_i \oplus \mathbf{A}_j) \cdot \mathbf{s}} \right].$$

Since $\mathbf{A}_i \neq \mathbf{A}_j$ for all $i \neq j$, then by the pairwise independence of $\mathbf{s}$, we have $\mathbb{E}[r[i] r[j]] = 0$. This means that X^2 is an unbiased estimator for $\|\mathbf{x}\|_2^2$.

Bounding Variance. Experimentally, we observed that the above construction works and achieves an accuracy comparable to that of the original AMS sketch! As shown in Table 5.1, the accuracy of this simplified structure is empirically identical to that achieved with uniform r . To complete the analysis, it remains to establish a provable accuracy bound for this cryptography-inspired design.

The accuracy of the AMS sketch fundamentally follows from its tightly controlled variance.

Concretely, since the first term of $X^2 = \sum_i \mathbf{x}[i]^2 + \sum_{i \neq j} \mathbf{x}[i]\mathbf{x}[j]\mathbf{r}[i]\mathbf{r}[j]$ is a constant, we have

$$\begin{aligned} \text{Var}[X^2] &= \text{Var}\left[\sum_{i \neq j} \mathbf{r}[i]\mathbf{r}[j]\mathbf{x}[i]\mathbf{x}[j]\right] = \mathbb{E}\left[\left(\sum_{i \neq j} \mathbf{r}[i]\mathbf{r}[j]\mathbf{x}[i]\mathbf{x}[j]\right)^2\right] \\ &= \sum_{i \neq j, k \neq \ell} \mathbb{E}\left[\mathbf{r}[i]\mathbf{r}[j]\mathbf{r}[k]\mathbf{r}[\ell]\right] \mathbf{x}[i]\mathbf{x}[j]\mathbf{x}[k]\mathbf{x}[\ell]. \end{aligned}$$

Note that $\mathbb{E}[\mathbf{r}[i]\mathbf{r}[j]\mathbf{r}[k]\mathbf{r}[\ell]] = 0$ if $\mathbf{A}_i \oplus \mathbf{A}_j \oplus \mathbf{A}_k \oplus \mathbf{A}_\ell \neq \mathbf{0}$ and equals 1 otherwise; hence, all terms that vanish in expectation can be discarded. Consequently, we obtain

$$\text{Var}[X^2] = \sum_{\mathbf{v} \neq \mathbf{0}} \sum_{\substack{\mathbf{A}_i \oplus \mathbf{A}_j = \mathbf{v} \\ \mathbf{A}_k \oplus \mathbf{A}_\ell = \mathbf{v}}} \mathbf{x}[i]\mathbf{x}[j]\mathbf{x}[k]\mathbf{x}[\ell] = \sum_{\mathbf{v} \neq \mathbf{0}} \left(\sum_{\mathbf{A}_i \oplus \mathbf{A}_j = \mathbf{v}} \mathbf{x}[i]\mathbf{x}[j] \right)^2.$$

Applying the Cauchy–Schwarz inequality yields

$$\text{Var}[X^2] \leq \sum_{\mathbf{v} \neq \mathbf{0}} N(\mathbf{v}) \sum_{\mathbf{A}_i \oplus \mathbf{A}_j = \mathbf{v}} \mathbf{x}[i]^2 \mathbf{x}[j]^2,$$

where $N(\mathbf{v}) \stackrel{\text{def}}{=} \#\{(i, j) \in [N]^2 \mid \mathbf{A}_i \oplus \mathbf{A}_j = \mathbf{v}\}$ denotes the count of pairs corresponding to a fixed $\mathbf{v}$. This count is a rare-event variable with a small expected value $o(1)$, and exhibits sub-exponential tail behavior. By taking a union bound over all possible $\mathbf{v}$, we obtain that $\max_{\mathbf{v} \neq \mathbf{0}} N(\mathbf{v}) \leq \log \log(n)$ with overwhelming probability (detailed in Theorem 10). Hence,

$$\text{Var}[X^2] \leq \log \log(n) \cdot \sum_{\mathbf{v} \neq \mathbf{0}} \sum_{\mathbf{A}_i \oplus \mathbf{A}_j = \mathbf{v}} \mathbf{x}[i]^2 \mathbf{x}[j]^2 \leq \log \log(n) \|\mathbf{x}\|_2^4.$$

This establishes that the variance remains tightly controlled, which underlies the accuracy of the cryptography-inspired AMS sketch.

Moreover, the sparsity of $\mathbf{A}$ keeps the matrix-vector product highly efficient, as each output entry depends on only a few seed bits, requiring only a small number of modulo-2 additions per entry. This property is particularly advantageous for evaluation under HE. Consequently, this approach

enables scalable generation of $\mathbf{r}$ without incurring the prohibitive overhead of fully independent random bits or of directly evaluating a PRG under FHE. For a realistic choice of $d = 8$, the per-element evaluation cost is 25.34 ms and is highly parallelizable, demonstrating high efficiency gains due to the introduced sparsity.

Connection to Subsampled-Hadamard Sketching. In hindsight, our construction turns out to have a clean interpretation in the language of randomness-efficient sketching. Recall a $2^n \times 2^n$ Hadamard matrix $\mathbf{H}$ has entries $\mathbf{H}[\mathbf{i}, \mathbf{j}] = (-1)^{\langle \mathbf{i}, \mathbf{j} \rangle}$, where $\mathbf{i}, \mathbf{j} \in \{0, 1\}^n$. A line of work in compressed sensing [113], [114] showed that when sketching a vector of dimension 2^n , one can build an AMS sketch by picking a random row index $\mathbf{s} \in \{0, 1\}^n$ and using the entire row $\mathbf{H}_{\mathbf{s}} \in \{-1, +1\}^{2^n}$ as the sketch vector—known as *subsampled-Hadamard sketching*.

Our construction uses exactly the same Hadamard matrix and the same row-sampling step, but reads off only N entries of the chosen row at positions selected by a fixed sparse pattern, with $N \ll 2^n$. Concretely, fix N distinct points $\mathbf{A}_1, \dots, \mathbf{A}_N \in \{0, 1\}^n$, each of Hamming weight d , and set $\mathbf{r}[j] = \mathbf{H}[\mathbf{s}, \mathbf{A}_j]$ for $j = 1, \dots, N$. The sketch is computed without materializing $\mathbf{H}$: each entry $\mathbf{r}[j] = (-1)^{\langle \mathbf{s}, \mathbf{A}_j \rangle}$ is the XOR of just d seed bits selected by the support of $\mathbf{A}_j$, which is what makes the sketch cheap under HE. We show that, with overwhelming probability, this structured-sparse pattern incurs at most a $\log \log n$ multiplicative overhead in sketch variance relative to standard AMS. Empirically, as shown in Figures 5.3b and 5.3c, the observed error closely matches that of standard AMS, suggesting that this overhead is negligible in practice. To our knowledge, this regime—where the row is sampled uniformly but the entries are restricted to a sparse subset of positions—has not been studied in the classical sketching literature, as it arises naturally only in cost models like HE evaluation where sparse linear maps are dramatically cheaper than dense ones.

5.1.5. Supporting Differentially Private Output

Finally, we show that our protocol can achieve differentially private output almost for free. The high-level idea is to add a small amount of Laplace noise to the aggregated encrypted sketches before computing the final estimate in 2PC. We show that, under modest assumptions on the input

norms, this noise is small enough that it does not meaningfully degrade the output quality. Moreover, the noise can be added to the encrypted sketches outside of MPC, so the servers do not pay the cost of sampling DP noise inside a 2PC protocol. See Section 5.4 for details.

5.2. AMS Sketch with Inadmissible Distribution

5.2.1. Δ -Inadmissible Distribution

Definition 4 (Inadmissible Set). *Let $\mathcal{D}$ be a distribution over $\{-1, 1\}^N$. A set of distinct index pairs $\mathcal{I} \subseteq [N]^2$ is called inadmissible if every $(i, j) \in \mathcal{I}$ satisfies $i \neq j$, and for all $(i, j), (k, \ell) \in \mathcal{I}$,*

$$\mathbb{E}_{\mathbf{r} \leftarrow \mathcal{D}}[\mathbf{r}[i]\mathbf{r}[j]\mathbf{r}[k]\mathbf{r}[\ell]] = 1 .$$

Definition 5 (Δ -Inadmissible Distribution). *Let $\mathcal{D}$ be a distribution over $\{-1, 1\}^N$. We say that $\mathcal{D}$ is Δ -inadmissible if it satisfies the following properties:*

1. **No pair-wise collision:** *For all $i \neq j$, $\mathbb{E}_{\mathbf{r} \leftarrow \mathcal{D}}[\mathbf{r}[i]\mathbf{r}[j]] = 0$.*
2. **Bounded inadmissible set size:** *Every inadmissible set has size at most Δ , i.e., for any inadmissible set $\mathcal{I}$ over distribution $\mathcal{D}$, we have $|\mathcal{I}| \leq \Delta$.*
3. **Partition with zero cross-correlation:** *There exists a partition $\{\mathcal{I}_1, \dots, \mathcal{I}_t\}$ of $\{(i, j) \in [N]^2 : i \neq j\}$ such that each $\mathcal{I}_m$ is an inadmissible set, and distinct sets have zero cross-correlation, i.e., for any $a \neq b$, $(i, j) \in \mathcal{I}_a$, and $(k, \ell) \in \mathcal{I}_b$,*

$$\mathbb{E}_{\mathbf{r} \leftarrow \mathcal{D}}[\mathbf{r}[i]\mathbf{r}[j]\mathbf{r}[k]\mathbf{r}[\ell]] = 0 .$$

Fact 1 (Uniform Distribution is 2-Inadmissible). *Let $\mathcal{D}$ be the uniform distribution over $\{-1, +1\}^N$. Then $\mathcal{D}$ is 2-inadmissible.*

Sketch of proof. For the uniform distribution:

1. Each inadmissible set contains at most two ordered pairs, namely (i, j) and (j, i) for some $i > j$.

We denote such a set by $\mathcal{I}_{(i,j)}$. Hence, every inadmissible set has size at most two.

2. For any $i \neq j$, we have $\mathbb{E}_{\mathbf{r} \sim \mathcal{D}}[\mathbf{r}[i]\mathbf{r}[j]] = 0$, since $\mathbf{r}[i]$ and $\mathbf{r}[j]$ are independent and uniformly distributed over $\{-1, +1\}$.
3. For any distinct inadmissible set $\mathcal{I}_{(i,j)} \neq \mathcal{I}_{(k,\ell)}$ with $i > j$ and $k > \ell$, we have

$$\mathbb{E}_{\mathbf{r} \leftarrow \mathcal{D}}[\mathbf{r}[i]\mathbf{r}[j]\mathbf{r}[k]\mathbf{r}[\ell]] = 0$$

by independence. Thus, $\mathcal{D}$ satisfies the partition property with zero cross-correlation.

Hence, the uniform distribution satisfies all the conditions of a 2-inadmissible distribution. $\square$

Theorem 10. *Given a binary matrix $\mathbf{A} \in \{0, 1\}^{N \times n}$, we define distribution $\mathcal{D}_{\mathbf{A}}$ over $\{-1, +1\}^N$ as follows: 1) sample a binary vector $\mathbf{s} \in \{0, 1\}^n$ uniformly at random and 2) output $\mathbf{r} = 2(\mathbf{A}\mathbf{s} \bmod 2) - \mathbf{1}$. Let $d > 1$ is a constant, $n \geq 4d$ and $N \leq n^{d/2}$. For an overwhelming fraction of possible choices of $\mathbf{A}$ with each row having Hamming weight d and $\mathbf{A}_i \neq \mathbf{A}_j$ for all $i \neq j$, we have distribution $\mathcal{D}_{\mathbf{A}}$ is $\log \log(n)$ -inadmissible.*

We defer the full proof of Theorem 10 to Appendix C.1 and provide a sketch below.

Proof sketch. We analyze the correlation structure induced by the distribution $\mathcal{D}_{\mathbf{A}}$. For any nonzero $\mathbf{v} \in \{0, 1\}^n$, we define

$$\mathcal{S}_{\mathbf{v}} \stackrel{\text{def}}{=} \{(i, j) \in [N]^2 \mid \mathbf{A}_i \oplus \mathbf{A}_j = \mathbf{v}\}.$$

Each $\mathcal{S}_{\mathbf{v}}$ forms an inadmissible set, since for any $(i, j) \in \mathcal{S}_{\mathbf{v}}$, the random variables $\mathbf{r}[i]$ and $\mathbf{r}[j]$ are correlated through the relation $\mathbf{A}_i \oplus \mathbf{A}_j = \mathbf{v}$. Thus, bounding $\Delta(\mathcal{D}_{\mathbf{A}})$ reduces to bounding $|\mathcal{S}_{\mathbf{v}}|$.

We first characterize the probability that two random weight- d vectors $\mathbf{A}_i, \mathbf{A}_j$ satisfy $\mathbf{A}_i \oplus \mathbf{A}_j = \mathbf{v}$. Let $t = \text{HW}(\mathbf{v})$. A combinatorial counting argument shows that $p_{\mathbf{v}} = \Pr[\mathbf{A}_i \oplus \mathbf{A}_j = \mathbf{v}] = \frac{\binom{t}{t/2} \binom{n-t}{d-t/2}}{\binom{n}{d}^2 - \binom{n}{d}}$. We show that for constant d and $n \geq 4d$, this probability is maximized at $t = 2$, yielding $p_{\max} = O(n^{-(d+1)})$.

To bound $|\mathcal{S}_v|$, define indicator variables $X_{(i,j)} = \mathbf{1}[\mathbf{A}_i \oplus \mathbf{A}_j = \mathbf{v}]$, so that $|\mathcal{S}_v| = \sum_{i \neq j} X_{(i,j)}$. We analyze the tail probability $\Pr[|\mathcal{S}_v| \geq k]$ via the factorial moment method, by bounding $\mathbb{E}[(|\mathcal{S}_v|)_k]$.

We distinguish two cases depending on whether the indices appearing in the k pairs are all distinct. Note that for any pair (i, j) satisfying $\mathbf{A}_i \oplus \mathbf{A}_j = \mathbf{v}$, the reversed pair (j, i) also satisfies the same condition. Hence, each unordered pair $\{i, j\}$ corresponds to at most two ordered pairs. Without loss of generality, we may restrict attention to a set of $k/2$ unordered pairs, denoted by $\{i_1, j_1\}, \dots, \{i_{k/2}, j_{k/2}\}$ with $i_t > j_t$ for all t . The *good case* corresponds to the situation where these $k/2$ pairs involve k distinct indices, i.e., no two pairs share a common index. The *bad case* corresponds to collisions among the indices. A key observation is that the bad case cannot occur: if two pairs share a vertex, the constraints imply that two distinct rows of $\mathbf{A}$ must be equal, contradicting the assumption that all rows are distinct. Hence, only the good case contributes.

In the good case, although rows are sampled without replacement, we can upper bound the joint probability via a sequential exposure argument, yielding a bound of $p_{\max}^{k/2}$. Since the number of such tuples is at most N^k , we obtain $\mathbb{E}[(|\mathcal{S}_v|)_k] \leq N^k \cdot p_{\max}^{k/2}$. Using $N \leq n^{d/2}$, this implies $\mathbb{E}[(|\mathcal{S}_v|)_k] \leq n^{-\Omega(k)}$.

Setting $k = \log \log n$ and applying Markov's inequality, we get

$$\Pr[|\mathcal{S}_v| \geq \log \log n] \leq \Pr[(|\mathcal{S}_v|)_k \geq k!] \leq \frac{\mathbb{E}[(|\mathcal{S}_v|)_k]}{k!} \leq \text{negl}(n).$$

Finally, a union bound over all $\mathbf{v} \neq 0$ (using $\text{HW}(\mathbf{v}) \leq 2d$) implies that $\Delta(\mathcal{D}_{\mathbf{A}}) \leq \log \log n$ with overwhelming probability. $\square$

5.2.2. AMS Sketch

Given a Δ -inadmissible distribution $\mathcal{D}$ (see Theorem 10), we now show how such a distribution can be directly used to instantiate an AMS-type sketch for inner product estimation.

At a high level, the classical AMS analysis relies on 4-wise independence to ensure unbiased-

ness and small variance. In our setting, although 4-wise independence no longer holds, the Δ -inadmissibility property guarantees that correlations are confined within small inadmissible sets. This structure is sufficient to recover unbiasedness and bound the variance with a multiplicative loss that depends only on $\Delta(\mathcal{D})$.

Theorem 11 (AMS sketch). *Let $\mathbf{x}, \mathbf{y} \in \mathbb{Z}^N$, and let $\mathcal{D}$ be a Δ -inadmissible distribution over $\{-1, +1\}^N$. Define AMS estimator*

$$Z^2 \stackrel{\text{def}}{=} \left(\sum_{i \in [N]} \mathbf{r}[i] \mathbf{x}[i] \right) \cdot \left(\sum_{j \in [N]} \mathbf{r}[j] \mathbf{y}[j] \right),$$

where $\mathbf{r} \leftarrow \mathcal{D}$. Then, $\mathbb{E}[Z^2] = \langle \mathbf{x}, \mathbf{y} \rangle$, $\text{Var}[Z^2] \leq \Delta(\mathcal{D}) \|\mathbf{x}\|_2^2 \|\mathbf{y}\|_2^2$.

Proof. We first rewrite

$$\begin{aligned} Z^2 &\stackrel{\text{def}}{=} \left(\sum_{i \in [N]} \mathbf{r}[i] \mathbf{x}[i] \right) \cdot \left(\sum_{j \in [N]} \mathbf{r}[j] \mathbf{y}[j] \right) \\ &= \sum_{i \in [N]} \mathbf{x}[i] \mathbf{y}[i] + \sum_{i \neq j} \mathbf{r}[i] \mathbf{r}[j] \mathbf{x}[i] \mathbf{y}[j], \end{aligned}$$

where the second equality follows from the fact that $\mathbf{r}[i]^2 = 1$ for all $i \in [N]$.

By definition of Δ -inadmissible distribution (see Definition 4 and Definition 5), we have $\mathbb{E}[\mathbf{r}[i] \mathbf{r}[j]] = 0$ for all $i \neq j$. Thus, we obtain

$$\mathbb{E}[Z^2] = \sum_{i \in [N]} \mathbf{x}[i] \mathbf{y}[i] + \sum_{i \neq j} \mathbf{x}[i] \mathbf{y}[j] \mathbb{E}[\mathbf{r}[i] \mathbf{r}[j]] = \langle \mathbf{x}, \mathbf{y} \rangle.$$

Thus, Z^2 is an unbiased estimation of $\langle \mathbf{x}, \mathbf{y} \rangle$.

Then, we bound the variance of the estimator. Since the first term of Z^2 , namely $\sum_{i \in [N]} \mathbf{x}[i] \mathbf{y}[i]$,

is a constant, we have

$$\begin{aligned}\text{Var}[Z^2] &= \text{Var}\left[\sum_{i \neq j} \mathbf{r}[i]\mathbf{r}[j]\mathbf{x}[i]\mathbf{y}[j]\right] = \mathbb{E}\left[\left(\sum_{i \neq j} \mathbf{r}[i]\mathbf{r}[j]\mathbf{x}[i]\mathbf{y}[j]\right)^2\right] \\ &= \sum_{i \neq j, k \neq l} \mathbf{x}[i]\mathbf{y}[j]\mathbf{x}[k]\mathbf{y}[l]\mathbb{E}\left[\mathbf{r}[i]\mathbf{r}[j]\mathbf{r}[k]\mathbf{r}[l]\right],\end{aligned}$$

where $\mathbb{E}\left[\sum_{i \neq j} \mathbf{r}[i]\mathbf{r}[j]\mathbf{x}[i]\mathbf{y}[j]\right] = 0$. There exists a partition of $\{(i, j) \in [N]^2 : i \neq j\}$ into inadmissible sets $\mathcal{I}_1 \dot{\cup} \dots \dot{\cup} \mathcal{I}_t$. Moreover, they have zero cross-correlation, i.e.,

$$\mathbb{E}\left[\mathbf{r}[i]\mathbf{r}[j]\mathbf{r}[k]\mathbf{r}[l]\right] = \begin{cases} 1, & (i, j), (k, l) \in \mathcal{I}_m \\ 0, & (i, j) \in \mathcal{I}_m, (k, l) \in \mathcal{I}_n, m \neq n \end{cases}.$$

Thus, in $\text{Var}[Z^2]$ all cross-terms between different $\mathcal{I}_m$'s are vanished and we have

$$\begin{aligned}\text{Var}[Z^2] &\leq \sum_m \sum_{(i,j), (k,l) \in \mathcal{I}_m} \mathbf{x}[i]\mathbf{y}[j]\mathbf{x}[k]\mathbf{y}[l] = \sum_m \left(\sum_{(i,j) \in \mathcal{I}_m} \mathbf{x}[i]\mathbf{y}[j]\right)^2.\end{aligned}$$

Applying Cauchy-Schwarz and Theorem 10 gives us

$$\begin{aligned}\text{Var}[Z^2] &\leq \sum_m |\mathcal{I}_m| \sum_{(i,j) \in \mathcal{I}_m} \mathbf{x}[i]^2 \mathbf{y}[j]^2 \leq \Delta(\mathcal{D}) \sum_m \sum_{(i,j) \in \mathcal{I}_m} \mathbf{x}[i]^2 \mathbf{y}[j]^2 \\ &\leq \Delta(\mathcal{D}) \sum_{i,j \in [N]} \mathbf{x}[i]^2 \mathbf{y}[j]^2 \leq \Delta(\mathcal{D}) \|\mathbf{x}\|_2^2 \|\mathbf{y}\|_2^2.\end{aligned}$$

□

Intuitively, the AMS estimator only depends on the coordinates where $\mathbf{a}$ and $\mathbf{b}$ are nonzero. Let $\text{supp} \stackrel{\text{def}}{=} \text{supp}(\mathbf{a}) \cup \text{supp}(\mathbf{b})$ denote this set of effective indices. Therefore, it suffices for the randomness vector $\mathbf{r}$ to satisfy the inadmissibility condition restricted to these effective indices.

Motivated by this observation, we relax the condition $N \leq n^{d/2}$ to $|\text{supp}| \leq n^{d/2}$. Indeed, the AMS estimator $Z^2 \stackrel{\text{def}}{=} \left(\sum_{i \in [N]} \mathbf{r}[i] \mathbf{x}[i] \right) \cdot \left(\sum_{j \in [N]} \mathbf{r}[j] \mathbf{y}[j] \right)$ only depends on coordinates in supp , and all terms $\mathbf{r}[i] \mathbf{r}[j] \mathbf{x}[i] \mathbf{y}[j]$, which are nonzero only when $i, j \in \text{supp}$. Thus, the analysis only involves pairs $(i, j) \in \text{supp} \times \text{supp}$. It follows that it suffices for $\mathcal{D}$ to be Δ -inadmissible over supp , i.e., restricted to pairs of indices in supp . Consequently, as long as $|\text{supp}| \leq n^{d/2}$, the same inadmissibility guarantee applies, and the variance bound follows.

Having bounded the variance, we can design a final sketch from the above basic sketch by a median-of-means improvement. By Theorem 2, this will blow up the space usage by a factor of $\frac{\text{Var}[Z^2]}{\mathbb{E}[Z^2]^2} O(\frac{1}{\varepsilon^2} \log \frac{1}{\delta}) = O(\Delta(\mathcal{D}) \frac{1}{\varepsilon^2} \log \frac{1}{\delta})$ in order to give an (ε, δ) -estimate.

5.2.3. Inner Product Approximation

The above analysis shows that the AMS estimator, when instantiated with a Δ -inadmissible distribution $\mathcal{D}$, provides an unbiased estimate of the inner product with controlled variance. We encapsulate this estimator as a primitive $\text{InnProdApprox}_{(k,t)}$, which takes two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}^N$ and a distribution $\mathcal{D}$ over $\{-1, +1\}^N$, and outputs a value z approximating $\langle \mathbf{a}, \mathbf{b} \rangle$. This instantiation is given in Algorithm 1. The sketch randomness $\mathbf{r}$ is sampled from $\mathcal{D}$. The number of trials required to achieve an (ε, δ) -estimation is $k = O\left(\Delta(\mathcal{D}) \cdot \frac{1}{\varepsilon^2}\right)$ and $t = O\left(\log \frac{1}{\delta}\right)$. Then, with probability at least $1 - \delta$, the output satisfies $|z - \langle \mathbf{a}, \mathbf{b} \rangle| \leq \varepsilon \cdot (\|\mathbf{a}\|_2 \|\mathbf{b}\|_2)$.

Algorithm 1 Inner Product Approximation via AMS with Inadmissible Distribution:
 $\text{InnProdApprox}_{(k,t)}(\mathcal{D}, \mathbf{a}, \mathbf{b})$

Input: $\mathbf{a} \in \mathbb{Z}^N, \mathbf{b} \in \mathbb{Z}^N$; an inadmissible distribution $\mathcal{D}$.

Output: Inner product approximation z .

- 1: **for** $\ell \in [k]$ and $h \in [t]$ **do**
 - 2: $\mathbf{r} \leftarrow \mathcal{D}$
 - 3: $u_a := \langle \mathbf{a}, \mathbf{r} \rangle$
 - 4: $u_b := \langle \mathbf{b}, \mathbf{r} \rangle$
 - 5: $z_{\ell,h} := u_a \cdot u_b$
 - 6: $z := \text{MoM}(\{z_{\ell,h}\}_{\ell \in [k], h \in [t]})$
 - 7: **return** z
-

5.3. PJC Protocol for Distributed Input Sets

We propose Algorithm 1 for private join and compute. The Δ -inadmissible distribution $\mathcal{D}$ is instantiated via FHE, enabling the encrypted generation of the sketching vector. The sketching procedure is then evaluated homomorphically, and the resulting sketches are combined using a 2PC protocol with a median-of-means aggregator. This yields an accurate approximation of the inner product while preserving input privacy.

Our framework naturally extends to PJC over distributed input sets. We encode the inputs as vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}^N$, where each coordinate aggregates the payloads associated with the corresponding key. We then estimate the inner product via $\langle \mathbf{a}, \mathbf{b} \rangle \approx \langle \mathbf{a}, \mathbf{r} \rangle \cdot \langle \mathbf{b}, \mathbf{r} \rangle$, where $\mathbf{r} \leftarrow \mathcal{D}$. This abstraction further generalizes seamlessly to the distributed setting. Given collections of parties $\mathcal{S}_1$ and $\mathcal{S}_2$, we aggregate their encoded inputs and compute

$$\left\langle \sum_{i \in \mathcal{S}_1} \mathbf{a}^{(i)}, \sum_{j \in \mathcal{S}_2} \mathbf{b}^{(j)} \right\rangle \approx \left(\sum_{i \in \mathcal{S}_1} \langle \mathbf{a}^{(i)}, \mathbf{r} \rangle \right) \cdot \left(\sum_{j \in \mathcal{S}_2} \langle \mathbf{b}^{(j)}, \mathbf{r} \rangle \right).$$

Therefore, PJC over distributed inputs is obtained directly, without any modification to the underlying protocol.

We present Π_{PJC} (Figure 5.2) to realize $\mathcal{F}_{\text{PJC}}$ for private join and compute over distributed input sets. The protocol involves two servers, Agg and CSP, and proceeds in the FHE + 2PC hybrid model described above. The sketching vector $\mathbf{r}$ remains hidden throughout, and all computations are performed over encrypted sketches, with only the final approximate result revealed. The detailed protocol is given below.

Protocol Π_{PJC}

Participants: Two servers Agg, CSP and w Parties $P_1, \dots, P_w$.

Parameters: Let $k, t, n, N, d \in \mathbb{N}$. Let $H : \{0, 1\}^* \rightarrow \{0, 1\}^{N \times n}$ be a random oracle that, on any input, outputs a Boolean matrix such that each row has Hamming weight exactly d and all rows are pairwise distinct.

Private sets: Each party P_i holds a private input set $\mathcal{X}_i = \{(x_j^i, v_j^i)\}_{j \in [m]}$, where $x_j^i \in [N]$ and $v_j^i \in \mathbb{Z}$.

Initialization:

1. Two servers invoke $\mathcal{F}_{\text{Rand}}$ and $\mathcal{F}_{\text{ThSWHE}}$ to obtain $seed \in \{0, 1\}^\lambda$ and keys (ek, pk), and then publish them.
2. For each $\ell \in [tk]$, Agg samples $s_1^\ell \in \{0, 1\}^n$, encrypts it as $\llbracket s_1^\ell \rrbracket$, and sends it to CSP. Agg also initializes two encrypted sketches $\llbracket c_1^\ell \rrbracket$ and $\llbracket c_2^\ell \rrbracket$ as fresh encryptions of zero.
3. CSP samples $s_2^\ell \in \{0, 1\}^n$, homomorphically computes $\llbracket s^\ell \rrbracket = \llbracket s_1^\ell \rrbracket \oplus s_2^\ell$, then rerandomizes $\llbracket s^\ell \rrbracket$ by homomorphically adding an encryption of zero, and finally publishes the rerandomized ciphertext $\llbracket s^\ell \rrbracket$.

Union operation over the set of P on (input, p, S):

1. P computes $\mathbf{A} \leftarrow H(seed)$.
2. For $\ell \in [tk]$, given P 's private set $\mathcal{X} = \{(a_i, b_i)\}_{i \in [m]}$, P homomorphically computes its encrypted sketching result:

$$\llbracket z^\ell \rrbracket = \sum_{i \in [m]} b_i \llbracket \mathbf{r}^\ell[a_i] \rrbracket,$$

where $\llbracket \mathbf{r}^\ell[i] \rrbracket$ is obtained via homomorphic evaluation as $\llbracket \mathbf{r}^\ell[i] \rrbracket = 2(\mathbf{A}_i \llbracket s^\ell \rrbracket \bmod 2) - 1$. Then, P rerandomizes each ciphertext $\llbracket z^\ell \rrbracket$ by homomorphically using a fresh encryption of zero $\llbracket 0 \rrbracket$, and sends the resulting ciphertexts to Agg.

3. For $\ell \in [tk]$, Agg updates $\llbracket c_p^\ell \rrbracket \leftarrow \llbracket c_p^\ell \rrbracket + \llbracket z^\ell + 0 \rrbracket$, where $p \in \{1, 2\}$.

Join Computation via 2PC on Servers:

1. For each $\ell \in [tk]$, Agg and CSP interact with $\mathcal{F}_{\text{ThSWHE}}$ to jointly decrypt $\llbracket c_1^\ell \rrbracket$ and $\llbracket c_2^\ell \rrbracket$, such that Agg obtains (d_1^ℓ, d_2^ℓ) and CSP obtains $(c_1^\ell - d_1^\ell, c_2^\ell - d_2^\ell)$.
2. Two servers call $\mathcal{F}_{\text{2PC}}$ on inputs (d_1^ℓ, d_2^ℓ) and $(c_1^\ell - d_1^\ell, c_2^\ell - d_2^\ell)$ to compute $z = \text{MoM}(\{c_1^\ell \cdot c_2^\ell\}_{\ell \in [tk]})$ and send z to Agg.

Figure 5.2: Private join and compute over distributed input sets, secure against semi-honest adversaries in the $(\mathcal{F}_{\text{Rand}}, \mathcal{F}_{\text{ThSWHE}}, \mathcal{F}_{\text{2PC}})$ -hybrid model.

Initialization: Servers Agg and CSP initialize $\mathcal{F}_{\text{ThSWHE}}$ to obtain $(\text{sk}, \text{pk}, \text{ek})$, and invoke $\mathcal{F}_{\text{Rand}}$ to sample a public seed $\text{seed} \in \{0, 1\}^\lambda$. The public parameters $(\text{pk}, \text{ek}, \text{seed})$ are published.

Both servers jointly generate an encrypted seed $\{\llbracket \mathbf{s}^\ell \rrbracket\}_{\ell \in [tk]}$ as follows. Agg samples a random vector $\mathbf{s}_1^\ell$ and sends its encryption $\llbracket \mathbf{s}_1^\ell \rrbracket$ to CSP. CSP samples $\mathbf{s}_2^\ell$, homomorphically computes $\llbracket \mathbf{s}^\ell \rrbracket = \llbracket \mathbf{s}_1^\ell \rrbracket \oplus \llbracket \mathbf{s}_2^\ell \rrbracket$, and rerandomizes the result by adding a fresh encryption of zero. The resulting $\llbracket \mathbf{s}^\ell \rrbracket$ is then published.

Server Agg initializes $\llbracket c_1^\ell \rrbracket$ and $\llbracket c_2^\ell \rrbracket$ as fresh encryptions of zero for all $\ell \in [tk]$.

Union operation over the set of P on $(\text{input}, p, \mathcal{S})$: Let P hold a private set $\mathcal{X} = \{(a_j, b_j)\}_{j \in [m]}$, and derive $\mathbf{A} \leftarrow H(\text{seed})$. For each $\ell \in [tk]$, P homomorphically computes an encrypted sketch $\llbracket z^\ell \rrbracket = \sum_{j \in [m]} b_j \cdot \llbracket \mathbf{r}^\ell[a_j] \rrbracket$, where $\llbracket \mathbf{r}^\ell[i] \rrbracket = 2(\mathbf{A}_i \llbracket \mathbf{s}^\ell \rrbracket \bmod 2) - 1$. Each $\llbracket z^\ell \rrbracket$ is rerandomized and sent to Agg, who updates $\llbracket c_p^\ell \rrbracket \leftarrow \llbracket c_p^\ell \rrbracket + \llbracket z^\ell \rrbracket$.

Join Computation via 2PC on Servers: The servers jointly decrypt $\{\llbracket c_1^\ell \rrbracket, \llbracket c_2^\ell \rrbracket\}_{\ell \in [tk]}$ via $\mathcal{F}_{\text{ThSWHE}}$ to obtain secret shares, and invoke $\mathcal{F}_{2\text{PC}}$ to compute $\text{MoM}(\{c_1^\ell \cdot c_2^\ell\}_{\ell \in [tk]})$.

In Theorem 12, we show that Π_{PJC} (Figure 5.2) securely realizes $\mathcal{F}_{\text{PJC}}$ (Figure 2.11) against semi-honest adversaries.

Theorem 12. *Let H be a random oracle, and assume that the SWHE scheme satisfies ciphertext pseudorandomness. Then, Protocol Π_{PJC} (Figure 5.2) securely realizes the functionality for private join and compute over distributed input sets in the presence of semi-honest adversaries in the $(\mathcal{F}_{\text{Rand}}, \mathcal{F}_{\text{ThSWHE}}, \mathcal{F}_{2\text{PC}})$ -hybrid model.*

The proof of Theorem 12 is deferred to Appendix C.2. Since fresh $\mathbf{s}^\ell$ are sampled per invocation while the public matrix $\mathbf{A}$ is derived from a fixed seed via a random oracle, the security analysis composes naturally across multiple protocol invocations on disjoint inputs.

5.4. Obtaining Differential Privacy for Free

So far, we have presented our PJC protocol based on homomorphic encryption, which achieves an error of $\mathcal{E} \cdot (\|\mathbf{a}\|_2^2 \|\mathbf{b}\|_2^2)$. In this chapter, we show that the protocol can be seamlessly combined with

differential privacy to further protect the released output. Notably, this adaptation is essentially “free of charge” when $\mathcal{E}$ is not too small, in the sense that achieving ε -DP incurs only an additional error term

$$O\left(\frac{\log \frac{1}{\delta} \sqrt{\Delta(\mathcal{D})}}{\varepsilon \mathcal{E}} (\|\mathbf{a}\|_2 + \|\mathbf{b}\|_2) + \frac{\log^2 \frac{1}{\delta} \Delta(\mathcal{D})^{1.5}}{\varepsilon^2 \mathcal{E}^3}\right).$$

Moreover, this additional term is asymptotically no larger than the original error of Algorithm 1 whenever $\mathcal{E} = \Omega(1/\min(\|\mathbf{a}\|_2, \|\mathbf{b}\|_2))$ considering that ε is typically treated as a constant.

We now describe how to modify Algorithm 1 so that the output z satisfies ε -DP. The main idea is to privatize each $z_{\ell,h}$. Rather than adding noise directly to $z_{\ell,h}$, we apply the Laplace mechanism (see Chapter 2.2.6) separately to u_a and u_b . We work in the record-level DP model, where neighboring datasets differ in the addition or removal of a single (key, value) record. Since each such record affects at most one entry of $\mathbf{a}$ or $\mathbf{b}$ by one, the global sensitivity of both u_a and u_b is 1. Therefore, adding Laplace noise with scale $1/\varepsilon$ to each of u_a and u_b ensures that each $z_{\ell,h}$ satisfies ε -DP. Since there are $k \times t$ number of $z_{\ell,h}$, we then apply the basic composition theorem and allocate a privacy budget of $\varepsilon/(kt)$ to each one, so that the entire collection $\{z_{\ell,h}\}_{\ell \in [k], h \in [t]}$ together satisfies ε -DP. Finally, because z is obtained as a post-processing of $\{z_{\ell,h}\}_{\ell \in [k], h \in [t]}$, it also satisfies ε -DP. In summary, the modified algorithm adds Laplace noise with scale kt/ε to each of u_a and u_b . Importantly, these additional operations can be implemented without increasing the size of MPC, by each server adding a Laplace noise to their shares before feeding them to MPC. Below, we further show that this modification will only introduce a minor extra additive error.

Theorem 13. *For any $\varepsilon > 0$ and any $0 < \mathcal{E} < 1$ let $t = O(\log \frac{1}{\delta})$ and $k = O(\Delta(\mathcal{D}) \cdot \frac{1}{\varepsilon^2})$, then, with probability at least $1 - \delta$, DP version Algorithm 1 returns an $\tilde{z}$ such that*

$$\begin{aligned} & |\tilde{z} - \langle \mathbf{a}, \mathbf{b} \rangle| \\ & \leq \mathcal{E} \|\mathbf{a}\|_2 \|\mathbf{b}\|_2 + O\left(\frac{\log \frac{1}{\delta} \sqrt{\Delta(\mathcal{D})}}{\varepsilon \mathcal{E}} (\|\mathbf{a}\|_2 + \|\mathbf{b}\|_2) + \frac{\log^2 \frac{1}{\delta} \Delta(\mathcal{D})^{1.5}}{\varepsilon^2 \mathcal{E}^3}\right). \end{aligned}$$

Proof. By following the same analysis as in Theorem 11, we have $\mathbb{E}(\tilde{z}_{\ell,h}) = \langle \mathbf{a}, \mathbf{b} \rangle$ and

$$\text{Var}(\tilde{z}_{\ell,h}) \leq \Delta(\mathcal{D}) \|\mathbf{a}\|_2^2 \|\mathbf{b}\|_2^2 + 2\left(\frac{tk}{\varepsilon}\right)^2 (\|\mathbf{a}\|_2^2 + \|\mathbf{b}\|_2^2) + 4\left(\frac{tk}{\varepsilon}\right)^4.$$

Therefore, with probability at least $1 - \delta$, we have

$$\mathcal{E} \|\mathbf{a}\|_2 \|\mathbf{b}\|_2 + \frac{2t\sqrt{k}}{\varepsilon} (\|\mathbf{a}\|_2 + \|\mathbf{b}\|_2) + \frac{4t^2 k^{1.5}}{\varepsilon^2}.$$

Combining with $k = O(\Delta(\mathcal{D}) \cdot \frac{1}{\varepsilon^2})$ and $t = O(\log \frac{1}{\delta})$, we get our target claim. $\square$

When $\mathcal{E}$ is at least on the order of $\Omega(1/\min(\|\mathbf{a}\|_2, \|\mathbf{b}\|_2))$, the first term, also corresponding to the error of Algorithm 1, dominates the remaining two terms. Thus, differential privacy comes essentially for free. For simplicity we use basic composition; tighter accountants such as Rényi DP [115] would shrink the per-mechanism noise scale by roughly a $\sqrt{kt}$ factor and improve the ε -regime over which DP comes essentially for free, but do not change the asymptotic form of Theorem 13.

5.5. Performance Evaluation

We implement our protocol end-to-end to provide an accurate benchmark of its efficiency. Homomorphic encryption operations are implemented using the Microsoft SEAL library [116], while two-party computation (2PC) operations are realized with the EMP toolkit [75]. In SEAL, we set the plaintext modulus to 24 bits and the polynomial modulus degree to 8192, supporting final output up to 2^{13} . Larger ranges require higher SEAL parameters. For efficiency, we use packing to batch up to 8192 instances into a single ciphertext. All experiments are conducted on AWS `c6i.xlarge` instances using a single thread. We consider two network configurations: a LAN setting with 5 Gbps bandwidth, and a WAN setting with 250 Mbps bandwidth and 80 ms latency.

5.5.1. Benchmark Accuracy

We evaluate the accuracy of $\text{InnProdApprox}_{(k,t)}(\mathcal{D}, \mathbf{a}, \mathbf{b})$ in Algorithm 1 under different parameter configurations and data distributions. We use binary vectors $\mathbf{a}, \mathbf{b} \in \{0, 1\}^N$ with $N = 2^{24}$, $\text{HW}(\mathbf{a}) = \text{HW}(\mathbf{b}) \leq 2^{18}$, and $\langle \mathbf{a}, \mathbf{b} \rangle \leq 2^{17}$. The inadmissible distribution $\mathcal{D}$ is instantiated following Theorem 10 via a binary matrix $\mathbf{A} \in \{0, 1\}^{N \times n}$ with row weight d . We examine how the empirical accuracy tracks our theoretical bound as k, t, n , and d vary. Each reported value is the median over 1000 independent trials.

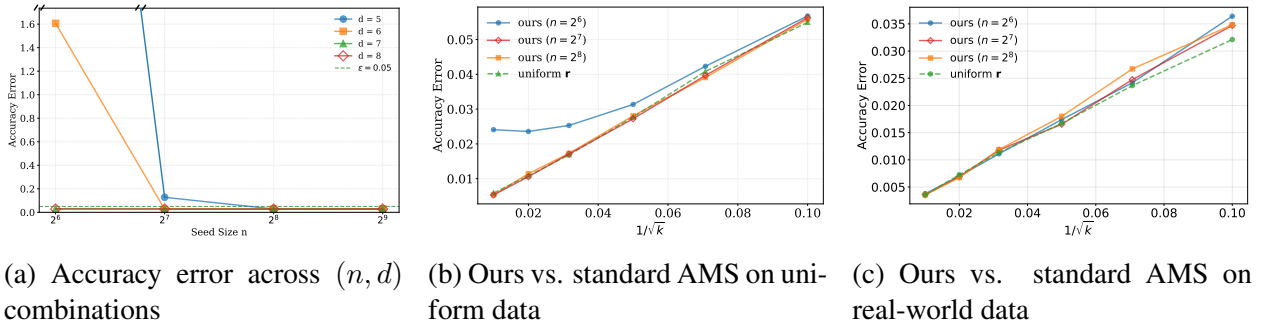


Figure 5.3: Accuracy Benchmark of Algorithm 1. (a) Empirical error across (n, d) combinations with $k = 400, t = 11, N = 2^{24}$, and $\text{supp}(\mathbf{a}) \cup \text{supp}(\mathbf{b}) = 1.5 \cdot 2^{18}$. (b,c) Empirical errors of Algorithm 1 against the standard AMS sketch on uniform and real-world data, with $d = 7, n \in \{2^6, 2^7, 2^8\}$; $\text{supp}(\mathbf{a}) \cup \text{supp}(\mathbf{b}) = 1.5 \cdot 2^{18}$ for uniform data and $\text{supp}(\mathbf{a}) \cup \text{supp}(\mathbf{b}) = 102,660$ for real-world data.

Accuracy with respect to (n, d) . We study how the empirical accuracy of Algorithm 1 depends on the parameters (n, d) that define $\mathcal{D}$. Recall that n controls the inadmissibility level $\Delta(\mathcal{D}) = \log \log n$, while d governs the feasibility condition $\text{supp}(\mathbf{a}) \cup \text{supp}(\mathbf{b}) \leq n^{d/2}$. We fix $k = 400, t = 11$, which yields a theoretical bound of $\mathcal{E} \approx 0.05$, and sweep $d \in \{5, 6, 7, 8\}$ and $n \in \{2^6, 2^7, 2^8, 2^9\}$, reporting the median accuracy error over 1000 trials. As shown in Figure 5.3a, the empirical median error falls below the theoretical bound $\mathcal{E} \approx 0.05$ precisely when $n^{d/2} \geq \text{supp}(\mathbf{a}) \cup \text{supp}(\mathbf{b})$, and degrades sharply once this bound is violated. This empirical result confirms that (n, d) need only be large enough to cover the effective inputs $\text{supp}(\mathbf{a}) \cup \text{supp}(\mathbf{b})$ rather than the whole dimension N , which is a great saving for sparse vectors.

Accuracy on uniform data. Recall that k controls the approximation error $\mathcal{E}$, while t controls

the confidence δ . We fix $t = 11$ and vary $k \in \{100, 200, 400, 1000, 2500, 10000\}$ to evaluate the accuracy of Algorithm 1, and compare with the standard AMS sketch as a baseline. We generate $\mathbf{a}$ and $\mathbf{b}$ uniformly at random with $\text{HW}(\mathbf{a}) = \text{HW}(\mathbf{b}) = 2^{18}$ and $\langle \mathbf{a}, \mathbf{b} \rangle = 2^{17}$, so $\text{supp}(\mathbf{a}) \cup \text{supp}(\mathbf{b}) = 1.5 \cdot 2^{18}$. We instantiate three distributions $\mathcal{D}$ with $d = 7$ and $n \in \{2^6, 2^7, 2^8\}$. The standard AMS sketch samples $\mathbf{r} \in \{-1, +1\}^N$ uniformly and admits the theoretical error bound $1/\sqrt{k}$.

As shown in Figure 5.3b, the empirical accuracy of Algorithm 1 matches the standard AMS sketch for $d = 7$ and $n \in \{2^7, 2^8\}$ across all k . This indicates that although our theoretical error bound is looser, we achieve the same practical accuracy on uniform data. One caveat is that $\mathcal{D}$ is generated from a short seed of length n , so n must provide enough entropy to support inputs with large $\text{supp}(\mathbf{a}) \cup \text{supp}(\mathbf{b})$. This is reflected in the $(2^6, 7)$ configuration, where the error fails to decrease as k increases.

Accuracy on real-world data. To evaluate Algorithm 1 on realistic workloads, we repeat the above experiment using vectors derived from the UCI Bag-of-Words dataset [117]. Unlike the uniform case where $\mathbf{a}, \mathbf{b}$ are binary, here $\mathbf{a}, \mathbf{b} \in \mathbb{Z}^N$ are count vectors. Specifically, we use a document with 10^8 total word occurrences over 102,660 unique words, and on each side we randomly sample 2^{18} words to form $\mathbf{a}$ and $\mathbf{b}$. We use the parameter sweep $t = 11$, $d = 7$, $k \in \{100, 200, 400, 1000, 2500, 10000\}$, and $n \in \{2^6, 2^7, 2^8\}$, comparing against the standard AMS sketch.

As shown in Figure 5.3c, the empirical accuracy of Algorithm 1 closely tracks that of the standard AMS sketch across all k and n , and all empirical errors lie below the theoretical bound $1/\sqrt{k}$ of the standard AMS sketch. The entropy limitation observed on uniform data at $n = 2^6$ does not appear here, because real-world document vectors have much smaller $\text{supp}(\mathbf{a}) \cup \text{supp}(\mathbf{b})$ (at most 102,660). Thus, Algorithm 1 remains accurate on realistic inputs.

5.5.2. Protocol Efficiency

Building on the accuracy results above, we now fix $t = 11$ and $k = 400$, which yields an error less than 0.05, and evaluate end-to-end protocol efficiency. The private join computation among

		$n = 128$	$n = 256$	$n = 512$	$n = 1024$
Comp (s)	LAN	1.83	3.28	6.22	12.12
	WAN	6.96	12.00	22.95	43.68
Comm (MB)		52.38	104.76	209.53	419.06
Publish (MB)		54.28	106.66	211.43	420.96

Table 5.2: Initialization costs for different n .

distributed clients proceeds in three phases:

1. **Server initialization:** Before clients provide their private inputs, CSP and Agg jointly generate and publish the encrypted random vectors, a public *seed*, and the key material (ek, pk) . This phase involves only server-to-server computation and a one-time broadcast to clients; no client input is required.
2. **Client local sketching:** Each client locally derives its encrypted sketch by expanding the encrypted randomness into the inadmissible distribution on its private elements and evaluating the sketch. The client sends a single sketch ciphertext to Agg.
3. **Server aggregation and output:** Agg homomorphically aggregates all client encrypted sketches into two ciphertexts, runs distributed decryption with CSP, executes 2PC to reconstruct the sketches, and computes the median-of-means estimate.

We evaluate per-phase cost under LAN and WAN settings below.

5.5.2.1. Server Initialization

In this phase, the communication overhead comes from CSP sending its encrypted random vectors to Agg, and the computational cost comes from Agg homomorphically combining them with its own random vectors. We assume the homomorphic-encryption keys are already generated, so this CSP-Agg communication consists solely of the encrypted random vectors. We vary the length $n \in \{128, 256, 512, 1024\}$ of the random vectors and evaluate both costs.

As shown in Table 5.2, both grow linearly with n . Concretely, for $n = 1024$, initialization takes 12.12 s in LAN and 43.68 s in WAN, with 419.06 MB of communication between the two

servers. Agg then publishes the encrypted randomness, a public seed, and key material, totaling 420.96 MB.

5.5.2.2. Client Local Sketching

In this phase, all computations are performed locally by each client. The dominant cost is expanding the encrypted randomness at each private element. For each element, this expansion multiplies d ciphertexts together, where $d = \text{HW}(\mathbf{A}_i)$. We implement this as a balanced binary tree of homomorphic multiplications, reducing the multiplication depth to $\lceil \log d \rceil$. Recall the kt rounds are packed into a single ciphertext; total cost grows linearly with number of private elements.

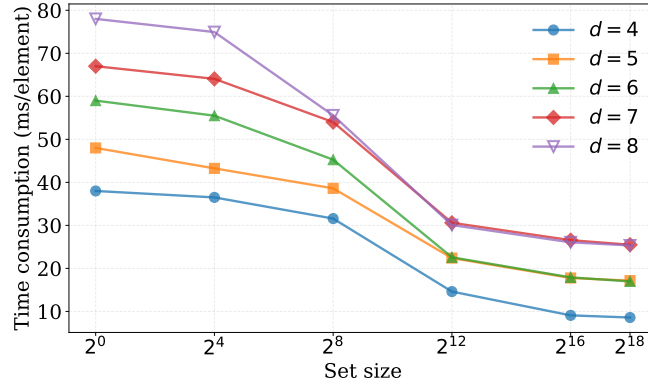


Figure 5.4: Amortized per-element cost for client local sketching. Tested under varying $d = \text{HW}(\mathbf{A}_i)$ and set size m .

We vary $d \in \{4, 5, 6, 7, 8\}$ and set size $m \in \{2^0, 2^4, 2^8, 2^{12}, 2^{16}, 2^{18}\}$; reporting the amortized per element cost in Figure 5.4. Two trends emerge. First, the amortized per element cost decreases as m grows. This is due to a caching optimization at the first tree level: pairwise products of encrypted-randomness indices are stored in a hash table and reused across elements, so larger m enables more cache hits. Second, the curves collapse into three tiers at large m . Once the cache absorbs nearly all first-level multiplications, the amortized cost is determined by the remaining uncached upper-tree multiplications: one for $d = 4$, two for $d = \{5, 6\}$, and three for $d = \{7, 8\}$. Values of d sharing the same uncached count therefore incur essentially the same cost. Concretely, for $d = 8$, the amortized cost is 30.08 ms at $m = 2^{12}$ and 25.34 ms at 2^{18} .

		Aggregate Client Encrypted Sketch	Threshold Decryption	2PC	Total
Comp. (s)	LAN	0.17	0.001	0.04	0.211
	WAN	0.17	0.006	1.50	1.676
Comm. (MB)		0	1.01	3.23	4.24

Table 5.3: Server aggregation and output costs. Broken down by step. The encrypted sketch aggregation includes 2000 clients.

On the communication side, each client sends only its sketch ciphertext to Agg, incurring 0.25 MB per client regardless of m .

5.5.2.3. Server Aggregation and Output

In this phase, three steps dominate the cost: (1) aggregating encrypted sketches from multiple clients; (2) threshold decryption to obtain secret shares of the two sketches; (3) a 2PC protocol to reconstruct the sketches and compute the median-of-means estimate. Aggregation cost scales linearly with the number of clients (one HE addition per client); we report the time for 2000 clients. We estimate the distributed decryption cost following [110], which takes 0.001 s in LAN and 0.006 s in WAN, with 1.01 MB communication. The 2PC cost depends on kt because, unlike earlier phases that pack all kt instances into a single ciphertext via SIMD, the 2PC operates on plaintext and handles each instance separately. For $kt = 4400$ in our setting, the 2PC takes 0.04 s in LAN and 1.50 s in WAN, with 3.23 MB communication.

5.5.2.4. Efficiency under Large Polynomial Modulus Degree.

Our default SEAL parameters support PJC with unit payloads, and random payloads when outputs remain below 2^{13} due to the limited noise budget. To support larger payloads, we test under polynomial modulus degree 16384, supporting output up to 2^{58} and batches up to $k \cdot t = 16384$ rounds for better accuracy. In our experiments, we use plaintext modulus 2^{24} . The test results are shown in Appendix C.4. Overall, the efficiency impact is limited: server initialization increases by $3\times$, client computation increases by $3\times$ but remains fully parallelizable, and client-server communi-

cation grows from 0.25 MB to 0.71 MB. Server aggregation and output increases by $4\times$ due to overflow protection for large payloads.

5.5.3. Comparison with Distributed-PJC Baselines

As discussed in Chapter 5, we compare our scheme against two natural baselines for extending PJC to the distributed setting: PJC decomposition from existing two-party PJC protocols [31], [32], [118]; and server-based generic 2PC. We evaluate both under two settings: a large- w setting (small m) with $w = 2,000$ clients (e.g., small businesses with limited per-party data); a small- w setting (large m) with $w = 20$ clients (e.g., enterprises each holding sizable datasets). In both, clients are evenly split between the two sides, and each holds m elements. Results are reported in Table 5.4.

- **PJC-decomposition baselines.** We first measure the per-execution cost of each two-party PJC scheme [31], [32], [118]. As no public implementation is available for [31] or for the inner-product construction of [118], we adopt the measurements reported in [31] and extrapolate communication and LAN running time to other input sizes using their asymptotic complexities. For [32] we run its implementation on a machine with 128 GB of RAM. Every client sequentially executes pairwise PJC against every client on the opposite side, so per-client communication and computation scale by $1,000\times$ in the large- w setting and $10\times$ in the small- w setting with respect to a single 2-party PJC. The server’s role is to sum the received pairwise inner products, which is negligible.
- **Server-based generic 2PC baseline.** Each client secret-shares its input to two non-colluding servers running a generic 2PC protocol. The protocol jointly sorts all entries by key (with source tags), and for each matching key aggregates payloads within each source, multiplies them across sources, and sums the results. The oblivious sort dominates, scaling as $O(N \log^2 N)$ in the aggregate input size N . Due to memory limitations, we partition each side’s aggregated input into chunks of 2^{13} and perform pairwise PJC. A single such PJC takes 21 s in LAN and 60 s

Scheme	Metric	$w = 2,000$ (1000/side)			$w = 20$ (10/side)		
		$m=2^8$	$m=2^{12}$	$m=2^{16}$	$m=2^{16}$	$m=2^{20}$	$m=2^{24}$
Ours	Client comm	0.25 MB	0.25 MB	0.25 MB	0.25 MB	0.25 MB	0.25 MB
	Client time	14.2 s	2.05 m	28.3 m	28.3 m	7.36 h	4.92 d*
	Server comm	58.52 MB	58.52 MB	110.90 MB	58.52 MB	110.90 MB	215.67 MB
	Server time LAN	2.04 s	2.04 s	3.49 s	1.87 s	3.32 s	6.26 s
	Server time WAN	8.48 s	8.48 s	13.5 s	8.48 s	13.5 s	24.5 s
KLS24 [32]	Client comm	910 MB	2.15 GB	20.2 GB	202 MB	3.07 GB	51 GB
	Client time LAN	3.55 m	5.20 m	1.25 h	44.9 s	13.5 m	4.81 h
	Client time WAN	45.7 m	52.8 m	2.52 h	1.51 m	18.7 m	5.39 h
	Server comm	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0
	Server time	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0
CHIKT23 [31]	Client comm	67 MB*	1.10 GB	17.2 GB	172 MB	2.75 GB	44 GB*
	Client time	1.67 m*	27.0 m	7.19 h	4.32 m	1.18 h	18.8 h*
	Server comm	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0
	Server time	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0
PSTY19 [118]	Client comm	1.84 GB*	29.5 GB*	472 GB	4.72 GB	75 GB*	1.33 TB*
	Client time	8.8 s*	5.08 m*	2.37 h	1.42 m	8.92 m*	1.93 h*
	Server comm	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0
	Server time	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0
Server-based 2PC	Client comm	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0
	Client time	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0
	Server comm	2.19 TB	561 TB	140 PB	14.37 TB	3.6 PB	920 PB
	Server time LAN	5.70 h	60.8 d	42.6 y	1.56 d	1.1 y	279 y
	Server time WAN	16.28 h	173 d	122 y	4.4 d	3.12 y	798 y

Table 5.4: PJC over distributed inputs. We report per-client and total server cost under a large- w setting and a small- w setting; each client holds m elements. For the PJC-decomposition baselines [31], [32], [118], each client executes one pairwise PJC against every client on the opposite side; the server’s role is summing pairwise inner products and is negligible. For the server-based generic 2PC baseline, client-side cost is negligible (preparing and sending secret shares). Time values are in seconds (s), minutes (m), hours (h), days (d), or years (y) as appropriate. Values marked with * are extrapolated.

in WAN with 2.3 GB communication; scaling to a per-side aggregate of size M increases cost by roughly $(M/2^{13})^2$ due to cross-chunk computation. Entries in Table 5.4 are extrapolated accordingly.

As shown in Table 5.4, our scheme achieves significant reductions in both settings. Client communication is a constant 0.25 MB, regardless of m . In small- w setting ($m = 2^{24}$), total communication is 221 MB, roughly $4,000\times$ less than [31] and over $10^9\times$ less than server-based 2PC.

The advantage widens in the large- w setting: at $m = 2^{16}$ with $w = 2,000$, total communication is 611 MB, four orders of magnitude below the PJC-decomposition and eight below server-based 2PC. Computation gains follow the same trend: in $w = 2,000(m = 2^{16})$, each client finishes in 28.3 minutes, whereas each client in KLS24 takes 1.25 hours and the server-based 2PC server requires 42.6 years.

CHAPTER 6

Conclusion

Real-world deployments of private set operators rarely match the classical one-shot two-party model: servers hold large persistent sets, the same server interacts with many clients over time, and the dataset on each side of a join is often distributed across many parties. This thesis develops private set intersection (PSI) and private join and compute (PJC) protocols for such client–server and distributed settings, with simulatable security guarantees and online communication efficiency.

1. **Maliciously secure client-output PSI with reusable server encoding (Chapter 3).** We design a PSI protocol for the password-checkup setting in which the server publishes a one-time, linear-size encoding of its set and then runs many lightweight sessions, each with online cost linear in the client’s set size. The construction is built on a new *oblivious verifiable unpredictable function* (OVUF), which binds every session to a single committed server key. We give efficient OVUF instantiations and prove simulation-based malicious security; the OVUF abstraction itself is of independent interest.
2. **Committed VOLE and maliciously secure server-output PSI (Chapter 4).** For the complementary server-output setting, we introduce *committed VOLE* (C-VOLE), a primitive that lets a party generate VOLE correlations on a pre-committed vector with many partners. Together with an LPN-based commitment whose structure minimizes the cost of consistency checks, C-VOLE yields a maliciously secure server-output PSI protocol with online communication linear in the client set size and a single committed dataset across all clients. C-VOLE also serves as a building block for VOLE-based zero-knowledge proofs and malicious MPC.
3. **Approximate PJC over distributed input sets with sublinear communication (Chapter 5).** We give the first approximate PJC protocol whose inputs are contributed by many clients, in the

semi-honest two-server model. The protocol provides a general framework for approximating set-intersection cardinality, summation, and inner products, with communication sublinear in the domain size. Its core technical contribution is an HE-friendly variant of the AMS sketch driven by an *inadmissible distribution* expanded homomorphically from a short seed, retaining the accuracy of standard AMS while keeping homomorphic cost low. Each client sends a single encrypted sketch; the two servers perform a one-time setup and a lightweight output phase. The released output can additionally be made differentially private at modest extra error.

Outlook. Several directions remain open. Lifting the distributed PJC protocol to malicious security would broaden its applicability to settings with stronger adversarial models. Tightening the theoretical accuracy bound for HE-compatible sketches would put approximate private analytics on firmer theoretical footing. More broadly, reducing the computational overhead of both PSI and PJC in the client–server setting remains essential for large-scale deployment. We hope that the primitives developed here serve as building blocks for scalable, deployable private set operators in modern data-sharing applications.

Bibliography

- [1] P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, and G. Tsudik, “Countering GATTACA: Efficient and secure testing of fully-sequenced human genomes,” in *ACM CCS 2011*, Y. Chen, G. Danezis, and V. Shmatikov, Eds., ACM Press, Oct. 2011, pp. 691–702.
- [2] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov, “BotGrep: Finding P2P bots with structured graph analysis,” in *USENIX Security 2010*, USENIX Association, Aug. 2010, pp. 95–110.
- [3] M. Ion et al., *On deploying secure computing commercially: Private intersection-sum protocols and their business applications*, Cryptology ePrint Archive, Report 2019/723, 2019.
- [4] B. Pal et al., “Might I get pwned: A second generation compromised credential checking service,” in *USENIX Security 2022*, K. R. B. Butler and K. Thomas, Eds., USENIX Association, Aug. 2022, pp. 1831–1848.
- [5] D. Kales, C. Rechberger, T. Schneider, M. Senker, and C. Weinert, “Mobile private contact discovery at scale,” in *USENIX Security 2019*, N. Heninger and P. Traynor, Eds., USENIX Association, Aug. 2019, pp. 1447–1464.
- [6] Y. Sun, J. Katz, M. Raykova, P. Schoppmann, and X. Wang, “Actively secure private set intersection in the client-server setting,” in *ACM CCS 2024*, B. Luo, X. Liao, J. Xu, E. Kirda, and D. Lie, Eds., ACM Press, Oct. 2024, pp. 1478–1492.
- [7] Y. Sun, H. Liu, K. Yang, Y. Yu, X. Wang, and C. Weng, “Committed vector oblivious linear evaluation and its applications,” in *ACM CCS 2025*, C.-Y. Huang, J.-C. Chen, S.-P. Shieh, D. Lie, and V. Cortier, Eds., ACM Press, Oct. 2025, pp. 3635–3648.
- [8] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold, “Keyword search and oblivious pseudorandom functions,” in *TCC 2005*, J. Kilian, Ed., ser. LNCS, vol. 3378, Springer, Berlin, Heidelberg, Feb. 2005, pp. 303–324.
- [9] C. Hazay and Y. Lindell, “Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries,” in *TCC 2008*, R. Canetti, Ed., ser. LNCS, vol. 4948, Springer, Berlin, Heidelberg, Mar. 2008, pp. 155–175.
- [10] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams, “Secure two-party computation is practical,” in *ASIACRYPT 2009*, M. Matsui, Ed., ser. LNCS, vol. 5912, Springer, Berlin, Heidelberg, Dec. 2009, pp. 250–267.

- [11] S. Jarecki and X. Liu, “Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection,” in *TCC 2009*, O. Reingold, Ed., ser. LNCS, vol. 5444, Springer, Berlin, Heidelberg, Mar. 2009, pp. 577–594.
- [12] H. Chen, Z. Huang, K. Laine, and P. Rindal, “Labeled PSI from fully homomorphic encryption with malicious security,” in *ACM CCS 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds., ACM Press, Oct. 2018, pp. 1223–1237.
- [13] P. Rindal and P. Schoppmann, “VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE,” in *EUROCRYPT 2021, Part II*, A. Canteaut and F.-X. Standaert, Eds., ser. LNCS, vol. 12697, Springer, Cham, Oct. 2021, pp. 901–930.
- [14] S. Raghuraman and P. Rindal, “Blazing fast PSI from improved OKVS and subfield VOLE,” in *ACM CCS 2022*, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds., ACM Press, Nov. 2022, pp. 2505–2517.
- [15] H. Chen, K. Laine, and P. Rindal, “Fast private set intersection from homomorphic encryption,” in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds., ACM Press, Oct. 2017, pp. 1243–1255.
- [16] K. Cong et al., “Labeled PSI from homomorphic encryption with reduced computation and communication,” in *ACM CCS 2021*, G. Vigna and E. Shi, Eds., ACM Press, Nov. 2021, pp. 1135–1150.
- [17] G. Ateniese, E. De Cristofaro, and G. Tsudik, “(If) size matters: Size-hiding private set intersection,” in *PKC 2011*, D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, Eds., ser. LNCS, vol. 6571, Springer, Berlin, Heidelberg, Mar. 2011, pp. 156–173.
- [18] N. Alapati, P. Branco, N. Döttling, S. Garg, M. Hajiabadi, and S. Pu, “Laconic private set intersection and applications,” in *TCC 2021, Part III*, K. Nissim and B. Waters, Eds., ser. LNCS, vol. 13044, Springer, Cham, Nov. 2021, pp. 94–125.
- [19] D. F. Aranha, C. Lin, C. Orlandi, and M. Simkin, “Laconic private set-intersection from pairings,” in *ACM CCS 2022*, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds., ACM Press, Nov. 2022, pp. 111–124.
- [20] Á. Kiss, J. Liu, T. Schneider, N. Asokan, and B. Pinkas, “Private set intersection for unequal set sizes with mobile applications,” *PoPETs*, vol. 2017, no. 4, pp. 177–197, Oct. 2017.
- [21] O. Nevo, N. Trieu, and A. Yanai, “Simple, fast malicious multiparty private set intersection,” in *ACM CCS 2021*, G. Vigna and E. Shi, Eds., ACM Press, Nov. 2021, pp. 1151–1165.

- [22] M. Rosulek and N. Trieu, “Compact and malicious private set intersection for small sets,” in *ACM CCS 2021*, G. Vigna and E. Shi, Eds., ACM Press, Nov. 2021, pp. 1166–1181.
- [23] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, “PSI from PaXoS: Fast, malicious private set intersection,” in *EUROCRYPT 2020, Part II*, A. Canteaut and Y. Ishai, Eds., ser. LNCS, vol. 12106, Springer, Cham, May 2020, pp. 739–767.
- [24] E. De Cristofaro, J. Kim, and G. Tsudik, “Linear-complexity private set intersection protocols secure in malicious model,” in *ASIACRYPT 2010*, M. Abe, Ed., ser. LNCS, vol. 6477, Springer, Berlin, Heidelberg, Dec. 2010, pp. 213–231.
- [25] T. Lepoint, S. Patel, M. Raykova, K. Seth, and N. Trieu, “Private join and compute from PIR with default,” in *ASIACRYPT 2021, Part II*, M. Tibouchi and H. Wang, Eds., ser. LNCS, vol. 13091, Springer, Cham, Dec. 2021, pp. 605–634.
- [26] Y. Wang and K. Yi, “Secure yannakakis: Join-aggregate queries over private data,” in *Proceedings of the 2021 International Conference on Management of Data*, ser. SIGMOD ’21, New York, NY, USA: Association for Computing Machinery, 2021, 1969–1981, ISBN: 9781450383431.
- [27] Google. “Helping organizations do more without collecting more data,” Google Online Security Blog, Accessed: Apr. 14, 2026.
- [28] P. Mohassel, P. Rindal, and M. Rosulek, “Fast database joins and PSI for secret shared data,” in *ACM CCS 2020*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds., ACM Press, Nov. 2020, pp. 1271–1287.
- [29] M. Ion et al., “On deploying secure computing: Private intersection-sum-with-cardinality,” in *2020 IEEE European Symposium on Security and Privacy*, IEEE Computer Society Press, Sep. 2020, pp. 370–389.
- [30] W. Chongchitmate, Y. Ishai, S. Lu, and R. Ostrovsky, “PSI from ring-OLE,” in *ACM CCS 2022*, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds., ACM Press, Nov. 2022, pp. 531–545.
- [31] K. Chida, K. Hamada, A. Ichikawa, M. Kii, and J. Tomida, “Communication-efficient inner product private join and compute with cardinality,” in *ASIACCS 23*, J. K. Liu, Y. Xiang, S. Nepal, and G. Tsudik, Eds., ACM Press, Jul. 2023, pp. 678–688.
- [32] J. Kim, H. T. Lee, and Y. Son, *Private set operations from circuit-based PSI*, Cryptology ePrint Archive, Paper 2024/1560, 2024.
- [33] Y. Son and J. Jeong, “PSI with computation or circuit-PSI for unbalanced sets from homomorphic encryption,” in *ASIACCS 23*, J. K. Liu, Y. Xiang, S. Nepal, and G. Tsudik, Eds., ACM Press, Jul. 2023, pp. 342–356.

- [34] Y. Dodis and A. Yampolskiy, “A verifiable random function with short proofs and keys,” in *PKC 2005*, S. Vaudenay, Ed., ser. LNCS, vol. 3386, Springer, Berlin, Heidelberg, Jan. 2005, pp. 416–431.
- [35] D. Boneh and X. Boyen, “Short signatures without random oracles,” in *EUROCRYPT 2004*, C. Cachin and J. Camenisch, Eds., ser. LNCS, vol. 3027, Springer, Berlin, Heidelberg, May 2004, pp. 56–73.
- [36] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya, and M. Meyerovich, “How to win the clonewars: Efficient periodic n-times anonymous authentication,” in *ACM CCS 2006*, A. Juels, R. N. Wright, and S. De Capitani di Vimercati, Eds., ACM Press, Oct. 2006, pp. 201–210.
- [37] A. Blum, M. L. Furst, M. J. Kearns, and R. J. Lipton, “Cryptographic primitives based on hard learning problems,” in *CRYPTO ’93*, D. R. Stinson, Ed., ser. LNCS, vol. 773, Springer, Berlin, Heidelberg, Aug. 1994, pp. 278–291.
- [38] C. Gentry, “A fully homomorphic encryption scheme,” in *Ph.D. thesis, Stanford University*, 2009.
- [39] N. Alon, Y. Matias, and M. Szegedy, “The space complexity of approximating the frequency moments,” in *28th ACM STOC*, ACM Press, May 1996, pp. 20–29.
- [40] A. S. Nemirovsky and D. B. Yudin, *Problem Complexity and Method Efficiency in Optimization* (Wiley-Interscience Series in Discrete Mathematics), trans. by E. R. Dawson. Chichester, New York, Brisbane, Toronto, Singapore: John Wiley & Sons, 1983, ISBN: 0-471-10345-4.
- [41] C. Dwork, F. McSherry, K. Nissim, and A. Smith, “Calibrating noise to sensitivity in private data analysis,” in *TCC 2006*, S. Halevi and T. Rabin, Eds., ser. LNCS, vol. 3876, Springer, Berlin, Heidelberg, Mar. 2006, pp. 265–284.
- [42] R. Canetti, “Universally composable security: A new paradigm for cryptographic protocols,” in *42nd FOCS*, IEEE Computer Society Press, Oct. 2001, pp. 136–145.
- [43] J. Camenisch, M. Drijvers, T. Gagliardoni, A. Lehmann, and G. Neven, “The wonderful world of global random oracles,” in *EUROCRYPT 2018, Part I*, J. B. Nielsen and V. Rijmen, Eds., ser. LNCS, vol. 10820, Springer, Cham, Apr. 2018, pp. 280–312.
- [44] E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai, “Compressing vector OLE,” in *ACM CCS 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds., ACM Press, Oct. 2018, pp. 896–912.

- [45] C. Weng, K. Yang, J. Katz, and X. Wang, “Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits,” in *2021 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, May 2021, pp. 1074–1091.
- [46] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, “Ferret: Fast extension for correlated OT with small communication,” in *ACM CCS 2020*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds., ACM Press, Nov. 2020, pp. 1607–1626.
- [47] P. Schoppmann, A. Gascón, L. Reichert, and M. Raykova, “Distributed vector-OLE: Improved constructions and implementation,” in *ACM CCS 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds., ACM Press, Nov. 2019, pp. 1055–1072.
- [48] H. Liu, X. Wang, K. Yang, and Y. Yu, “The hardness of LPN over any integer ring and field for PCG applications,” in *EUROCRYPT 2024, Part VI*, M. Joye and G. Leander, Eds., ser. LNCS, vol. 14656, Springer, Cham, May 2024, pp. 149–179.
- [49] C. Hazay, P. Scholl, and E. Soria-Vazquez, “Low cost constant round MPC combining BMR and oblivious transfer,” in *ASIACRYPT 2017, Part I*, T. Takagi and T. Peyrin, Eds., ser. LNCS, vol. 10624, Springer, Cham, Dec. 2017, pp. 598–628.
- [50] K. Yang, X. Wang, and J. Zhang, “More efficient MPC from improved triple generation and authenticated garbling,” in *ACM CCS 2020*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds., ACM Press, Nov. 2020, pp. 1627–1646.
- [51] S. Ben-David, Y. T. Kalai, and O. Paneth, “Verifiable private information retrieval,” in *TCC 2022, Part III*, E. Kiltz and V. Vaikuntanathan, Eds., ser. LNCS, vol. 13749, Springer, Cham, Nov. 2022, pp. 3–32.
- [52] L. de Castro and K. Lee, “Verisimplepir: Verifiability in simplepir at no online cost for honest servers,” in *USENIX Security 2024*, USENIX Association, Aug. 2024.
- [53] A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda, “Privacy pass: Bypassing internet challenges anonymously,” *PoPETs*, vol. 2018, no. 3, pp. 164–180, Jul. 2018.
- [54] A. Everspaugh, R. Chatterjee, S. Scott, A. Juels, and T. Ristenpart, “The pythia PRF service,” in *USENIX Security 2015*, J. Jung and T. Holz, Eds., USENIX Association, Aug. 2015, pp. 547–562.
- [55] S. Jarecki, H. Krawczyk, and J. K. Resch, “Updatable oblivious key management for storage systems,” in *ACM CCS 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds., ACM Press, Nov. 2019, pp. 379–393.

- [56] B. Kreuter, T. Lepoint, M. Orrù, and M. Raykova, “Anonymous tokens with private meta-data bit,” in *CRYPTO 2020, Part I*, D. Micciancio and T. Ristenpart, Eds., ser. LNCS, vol. 12170, Springer, Cham, Aug. 2020, pp. 308–336.
- [57] T. Silde and M. Strand, “Anonymous tokens with public metadata and applications to private contact tracing,” in *FC 2022*, I. Eyal and J. A. Garay, Eds., ser. LNCS, vol. 13411, Springer, Cham, May 2022, pp. 179–199.
- [58] N. Tyagi, S. Celi, T. Ristenpart, N. Sullivan, S. Tessaro, and C. A. Wood, “A fast and simple partially oblivious PRF, with applications,” in *EUROCRYPT 2022, Part II*, O. Dunkelman and S. Dziembowski, Eds., ser. LNCS, vol. 13276, Springer, Cham, May 2022, pp. 674–705.
- [59] A. Basso, “A post-quantum round-optimal oblivious PRF from isogenies,” in *SAC 2023*, C. Carlet, K. Mandal, and V. Rijmen, Eds., ser. LNCS, vol. 14201, Springer, Cham, Aug. 2024, pp. 147–168.
- [60] D. Boneh, D. Kogan, and K. Woo, “Oblivious pseudorandom functions from isogenies,” in *ASIACRYPT 2020, Part II*, S. Moriai and H. Wang, Eds., ser. LNCS, vol. 12492, Springer, Cham, Dec. 2020, pp. 520–550.
- [61] M. R. Albrecht, A. Davidson, A. Deo, and N. P. Smart, “Round-optimal verifiable oblivious pseudorandom functions from ideal lattices,” in *PKC 2021, Part II*, J. Garay, Ed., ser. LNCS, vol. 12711, Springer, Cham, May 2021, pp. 261–289.
- [62] I. A. Seres, M. Horváth, and P. Burcsi, *The Legendre pseudorandom function as a multivariate quadratic cryptosystem: Security and applications*, Cryptology ePrint Archive, Report 2021/182, 2021.
- [63] J. Doerner, Y. Kondi, E. Lee, and a. shelat, “Secure two-party threshold ECDSA from ECDSA assumptions,” in *2018 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, May 2018, pp. 980–997.
- [64] H. Xue, M. H. Au, X. Xie, T. H. Yuen, and H. Cui, “Efficient online-friendly two-party ECDSA signature,” in *ACM CCS 2021*, G. Vigna and E. Shi, Eds., ACM Press, Nov. 2021, pp. 558–573.
- [65] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker, “Two-party ECDSA from hash proof systems and efficient instantiations,” in *CRYPTO 2019, Part III*, A. Boldyreva and D. Micciancio, Eds., ser. LNCS, vol. 11694, Springer, Cham, Aug. 2019, pp. 191–221.
- [66] S. Micali, M. O. Rabin, and S. P. Vadhan, “Verifiable random functions,” in *40th FOCS*, IEEE Computer Society Press, Oct. 1999, pp. 120–130.

- [67] J. Doerner, Y. Kondi, E. Lee, and a. shelat, “Threshold ECDSA from ECDSA assumptions: The multiparty case,” in *2019 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, May 2019, pp. 1051–1066.
- [68] I. Haitner, N. Makriyannis, S. Ranellucci, and E. Tsfadia, “Highly efficient OT-based multiplication protocols,” in *EUROCRYPT 2022, Part I*, O. Dunkelman and S. Dziembowski, Eds., ser. LNCS, vol. 13275, Springer, Cham, 2022, pp. 180–209.
- [69] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled, “UC non-interactive, proactive, threshold ECDSA with identifiable aborts,” in *ACM CCS 2020*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds., ACM Press, Nov. 2020, pp. 1769–1787.
- [70] R. Gennaro and S. Goldfeder, “Fast multiparty threshold ECDSA with fast trustless setup,” in *ACM CCS 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds., ACM Press, Oct. 2018, pp. 1179–1194.
- [71] Y. Lindell and A. Nof, “Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody,” in *ACM CCS 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds., ACM Press, Oct. 2018, pp. 1837–1854.
- [72] N. Gilboa, “Two party RSA key generation,” in *CRYPTO ’99*, M. J. Wiener, Ed., ser. LNCS, vol. 1666, Springer, Berlin, Heidelberg, Aug. 1999, pp. 116–129.
- [73] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, and V. Vaikuntanathan, *One server for the price of two: Simple and fast single-server private information retrieval*, Cryptology ePrint Archive, Report 2022/949, 2022.
- [74] L. Li, B. Pal, J. Ali, N. Sullivan, R. Chatterjee, and T. Ristenpart, “Protocols for checking compromised credentials,” in *ACM CCS 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds., ACM Press, Nov. 2019, pp. 1387–1403.
- [75] X. Wang, A. J. Malozemoff, and J. Katz, *EMP-toolkit: Efficient MultiParty computation toolkit*, <https://github.com/emp-toolkit>, 2016.
- [76] D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao, *RELIC is an Efficient Library for Cryptography*, <https://github.com/relic-toolkit/relic>.
- [77] M. Keller, E. Orsini, and P. Scholl, “Actively secure OT extension with optimal overhead,” in *CRYPTO 2015, Part I*, R. Gennaro and M. J. B. Robshaw, Eds., ser. LNCS, vol. 9215, Springer, Berlin, Heidelberg, Aug. 2015, pp. 724–741.
- [78] A. C. D. Resende and D. F. Aranha, “Faster unbalanced private set intersection,” in *FC 2018*, S. Meiklejohn and K. Sako, Eds., ser. LNCS, vol. 10957, Springer, Berlin, Heidelberg, Feb. 2018, pp. 203–221.

- [79] S. Jarecki and X. Liu, “Fast secure computation of set intersection,” in *SCN 10*, J. A. Garay and R. De Prisco, Eds., ser. LNCS, vol. 6280, Springer, Berlin, Heidelberg, Sep. 2010, pp. 418–435.
- [80] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra, “A new approach to practical active-secure two-party computation,” in *CRYPTO 2012*, R. Safavi-Naini and R. Canetti, Eds., ser. LNCS, vol. 7417, Springer, Berlin, Heidelberg, Aug. 2012, pp. 681–700.
- [81] C. Baum, A. J. Malozemoff, M. B. Rosen, and P. Scholl, “Mac’n’cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions,” in *CRYPTO 2021, Part IV*, T. Malkin and C. Peikert, Eds., ser. LNCS, vol. 12828, Virtual Event: Springer, Cham, Aug. 2021, pp. 92–122.
- [82] S. Dittmer, Y. Ishai, and R. Ostrovsky, “Line-point zero knowledge and its applications,” in *ITC 2021*, S. Tessaro, Ed., ser. LIPIcs, vol. 199, Schloss Dagstuhl, Jul. 2021, 5:1–5:24.
- [83] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias, “Semi-homomorphic encryption and multiparty computation,” in *EUROCRYPT 2011*, K. G. Paterson, Ed., ser. LNCS, vol. 6632, Springer, Berlin, Heidelberg, May 2011, pp. 169–188.
- [84] S. Dittmer, Y. Ishai, S. Lu, and R. Ostrovsky, “Authenticated garbling from simple correlations,” in *CRYPTO 2022, Part IV*, Y. Dodis and T. Shrimpton, Eds., ser. LNCS, vol. 13510, Springer, Cham, Aug. 2022, pp. 57–87.
- [85] R. Rachuri and P. Scholl, “Le mans: Dynamic and fluid MPC for dishonest majority,” in *CRYPTO 2022, Part I*, Y. Dodis and T. Shrimpton, Eds., ser. LNCS, vol. 13507, Springer, Cham, Aug. 2022, pp. 719–749.
- [86] C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang, “Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning,” in *USENIX Security 2021*, M. Bailey and R. Greenstadt, Eds., USENIX Association, Aug. 2021, pp. 501–518.
- [87] M. R. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen, “MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity,” in *ASIACRYPT 2016, Part I*, J. H. Cheon and T. Takagi, Eds., ser. LNCS, vol. 10031, Springer, Berlin, Heidelberg, Dec. 2016, pp. 191–219.
- [88] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, “Poseidon: A new hash function for zero-knowledge proof systems,” in *USENIX Security 2021*, M. Bailey and R. Greenstadt, Eds., USENIX Association, Aug. 2021, pp. 519–535.
- [89] A. Aly, T. Ashur, E. Ben-Sasson, S. Dhooghe, and A. Szepieniec, “Design of symmetric-key primitives for advanced cryptographic protocols,” *IACR Trans. Symm. Cryptol.*, vol. 2020, no. 3, pp. 1–45, 2020.

- [90] L. Grassi, D. Khovratovich, and M. Schofnegger, “Poseidon2: A faster version of the poseidon hash function,” in *AFRICACRYPT 23*, N. El Mrabet, L. De Feo, and S. Duquesne, Eds., ser. LNCS, vol. 14064, Springer, Cham, Jul. 2023, pp. 177–203.
- [91] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, “Multiparty computation from somewhat homomorphic encryption,” in *CRYPTO 2012*, R. Safavi-Naini and R. Canetti, Eds., ser. LNCS, vol. 7417, Springer, Berlin, Heidelberg, Aug. 2012, pp. 643–662.
- [92] X. Wang, S. Ranellucci, and J. Katz, “Global-scale secure multiparty computation,” in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds., ACM Press, Oct. 2017, pp. 39–56.
- [93] C. Hazay, P. Scholl, and E. Soria-Vazquez, “Low cost constant round MPC combining BMR and oblivious transfer,” *Journal of Cryptology*, vol. 33, no. 4, pp. 1732–1786, Oct. 2020.
- [94] D. Escudero, A. Polychroniadou, Y. Song, and C. Weng, “Multi-verifier zero-knowledge proofs for any constant fraction of corrupted verifiers,” in *ACM CCS 2024*, B. Luo, X. Liao, J. Xu, E. Kirda, and D. Lie, Eds., ACM Press, Oct. 2024, pp. 4092–4106.
- [95] R. Cohen, J. Doerner, Y. Kondi, and a. shelat, “Secure multiparty computation with identifiable abort via vindicating release,” in *CRYPTO 2024, Part VIII*, L. Reyzin and D. Stebila, Eds., ser. LNCS, vol. 14927, Springer, Cham, Aug. 2024, pp. 36–73.
- [96] M. Jawurek, F. Kerschbaum, and C. Orlandi, “Zero-knowledge using garbled circuits: How to prove non-algebraic statements efficiently,” in *ACM CCS 2013*, A.-R. Sadeghi, V. D. Gligor, and M. Yung, Eds., ACM Press, Nov. 2013, pp. 955–966.
- [97] E. Boyle et al., “Correlated pseudorandomness from expand-accumulate codes,” in *CRYPTO 2022, Part II*, Y. Dodis and T. Shrimpton, Eds., ser. LNCS, vol. 13508, Springer, Cham, Aug. 2022, pp. 603–633.
- [98] S. Raghuraman, P. Rindal, and T. Tanguy, “Expand-convolute codes for pseudorandom correlation generators from LPN,” in *CRYPTO 2023, Part IV*, H. Handschuh and A. Lysyanskaya, Eds., ser. LNCS, vol. 14084, Springer, Cham, Aug. 2023, pp. 602–632.
- [99] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl, “Efficient pseudorandom correlation generators: Silent OT extension and more,” in *CRYPTO 2019, Part III*, A. Boldyreva and D. Micciancio, Eds., ser. LNCS, vol. 11694, Springer, Cham, Aug. 2019, pp. 489–518.
- [100] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, and M. Orrù, “Homomorphic secret sharing: Optimizations and applications,” in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds., ACM Press, 2017, pp. 2105–2122.

- [101] A. Jain, S. Krenn, K. Pietrzak, and A. Tentes, “Commitments and efficient zero-knowledge proofs from learning parity with noise,” in *ASIACRYPT 2012*, X. Wang and K. Sako, Eds., ser. LNCS, vol. 7658, Springer, Berlin, Heidelberg, Dec. 2012, pp. 663–680.
- [102] K. Yang, P. Sarkar, C. Weng, and X. Wang, “QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field,” in *ACM CCS 2021*, G. Vigna and E. Shi, Eds., ACM Press, Nov. 2021, pp. 2986–3001.
- [103] R. C. Merkle, “A digital signature based on a conventional encryption function,” in *CRYPTO’87*, C. Pomerance, Ed., ser. LNCS, vol. 293, Springer, Berlin, Heidelberg, Aug. 1988, pp. 369–378.
- [104] P. Rindal and L. Roy, *libOTe: An efficient, portable, and easy to use oblivious transfer library*, <https://github.com/osu-crypto/libOTe>.
- [105] Visa Research, *Vole-PSI: Efficient private set intersection based on VOLE*, <https://github.com/Visa-Research/volepsi>.
- [106] S. Dittmer, Y. Ishai, S. Lu, and R. Ostrovsky, “Improving line-point zero knowledge: Two multiplications for the price of one,” in *ACM CCS 2022*, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds., ACM Press, Nov. 2022, pp. 829–841.
- [107] C. Peikert, V. Vaikuntanathan, and B. Waters, “A framework for efficient and composable oblivious transfer,” in *CRYPTO 2008*, D. Wagner, Ed., ser. LNCS, vol. 5157, Springer, Berlin, Heidelberg, Aug. 2008, pp. 554–571.
- [108] F. Falzon and T. Tang, “Learning from functionality outputs: Private join and compute in the real world,” in *USENIX Security 2025*, L. Bauer and G. Pellegrino, Eds., USENIX Association, Aug. 2025, pp. 7603–7622.
- [109] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh, “Privacy-preserving matrix factorization,” in *ACM CCS 2013*, A.-R. Sadeghi, V. D. Gligor, and M. Yung, Eds., ACM Press, Nov. 2013, pp. 801–812.
- [110] W. Zhang et al., “BitGC made (more) efficient,” in *Advances in Cryptology - EUROCRYPT 2026*, ser. Lecture Notes in Computer Science, Springer, 2026.
- [111] O. Goldreich, “Candidate one-way functions based on expander graphs,” in *Elect. Colloq. on Computational Complexity (ECCC)*, vol. 7, 2000.
- [112] A. Agarwal, C. Baum, L. Braun, and P. Scholl, “Low-bandwidth mixed arithmetic in VOLE-based ZK from low-degree PRGs,” in *EUROCRYPT 2025, Part IV*, S. Fehr and P.-A. Fouque, Eds., ser. LNCS, vol. 15604, Springer, Cham, May 2025, pp. 396–426.

- [113] F. Krahmer and R. A. Ward, “New and improved johnson-lindenstrauss embeddings via the restricted isometry property,” *SIAM J. Math. Anal.*, vol. 43, no. 3, pp. 1269–1281, 2011.
- [114] I. Haviv and O. Regev, “The restricted isometry property of subsampled fourier matrices,” in *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, R. Krauthgamer, Ed., SIAM, 2016, pp. 288–297.
- [115] I. Mironov, “Rényi differential privacy,” in *CSF 2017 Computer Security Foundations Symposium*, B. Köpf and S. Chong, Eds., IEEE Computer Society Press, 2017, pp. 263–275.
- [116] *Microsoft SEAL (release 4.1)*, <https://github.com/Microsoft/SEAL>, Microsoft Research, Redmond, WA., Jan. 2023.
- [117] D. Newman, *Bag of Words*, UCI Machine Learning Repository, DOI: <https://doi.org/10.24432/C5ZG6P>, 2008.
- [118] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai, “Efficient circuit-based PSI with linear communication,” in *EUROCRYPT 2019, Part III*, Y. Ishai and V. Rijmen, Eds., ser. LNCS, vol. 11478, Springer, Cham, May 2019, pp. 122–153.
- [119] E. Boyle, N. Gilboa, and Y. Ishai, “Function secret sharing,” in *EUROCRYPT 2015, Part II*, E. Oswald and M. Fischlin, Eds., ser. LNCS, vol. 9057, Springer, Berlin, Heidelberg, Apr. 2015, pp. 337–367.
- [120] E. Boyle, N. Gilboa, and Y. Ishai, “Function secret sharing: Improvements and extensions,” in *ACM CCS 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds., ACM Press, Oct. 2016, pp. 1292–1303.
- [121] O. Goldreich, S. Goldwasser, and S. Micali, “How to construct random functions (extended abstract),” in *25th FOCS*, IEEE Computer Society Press, Oct. 1984, pp. 464–479.

APPENDIX A

Appendix for Chapter 3

A.1. Batched OVUF with Improved Efficiency

In this chapter, we give an optimized maliciously secure oblivious verifiable unpredictable protocol Π_{OVUF_2} in terms of efficiency. This protocol combines a new sub-protocol $\Pi_{\text{U-MtA}}$ called unbalanced imperfect multiplicative to additive shares transformation, introduced in Appendix A.1.1. The details and proof of Π_{OVUF_2} are depicted in Appendix A.1.2.

A.1.1. Unbalanced Imperfect MtA

This transformation computes the additive secret shares of scalar-vector multiplication, where the scalar and vector held by different parties are regarded as unbalanced input. Its imperfection follows the same idea as Chapter 3.2.2.2 that a malicious sender can execute attacks and result in incorrect additive secret shares depending on the receiver's input.

The most straightforward way to achieve imperfect scalar-vector multiplicative to additive shares is as follows: Given the input vector $\mathbf{a} \in \mathbb{Z}_q^n$ on party P_0 and scalar $b \in \mathbb{Z}_q$ on party P_1 , let P_1 create a new vector $\mathbf{b}$ with each element $b_i = b$. Then, both parties execute Π_{MtA} , using $\mathbf{a}$ and $\mathbf{b}$ as inputs. However, in our construction in Figure A.1, we designate P_1 as receiver of $\mathcal{F}_{\text{COT}}$ and have P_1 employ $\text{Encode}(\mathbf{g}^R, b)$. Sender P_0 inputs vector element $\mathbf{a}_i$ and the receiver P_1 inputs encoded bit element of b to run $\mathcal{F}_{\text{COT}}$ to compute the additive secret share of $\mathbf{a}_i \cdot b$. This approach consumes $n(2 \log q + 2s)$ iterations of $\mathcal{F}_{\text{COT}}$, the same as the straightforward approach stated above. However, it eliminates pseudorandom vector γ of length $(n - 1) \log q$ and repetitive encoding of b of length $(n - 1) \log q$ when implementing the encoding algorithm.

For the incorrectness caused by the sender P_0 's malicious behavior as stated in Chapter 3.2.2.2, it follows the same error representation as the single encoded version in Chapter 3.2.2.2, that

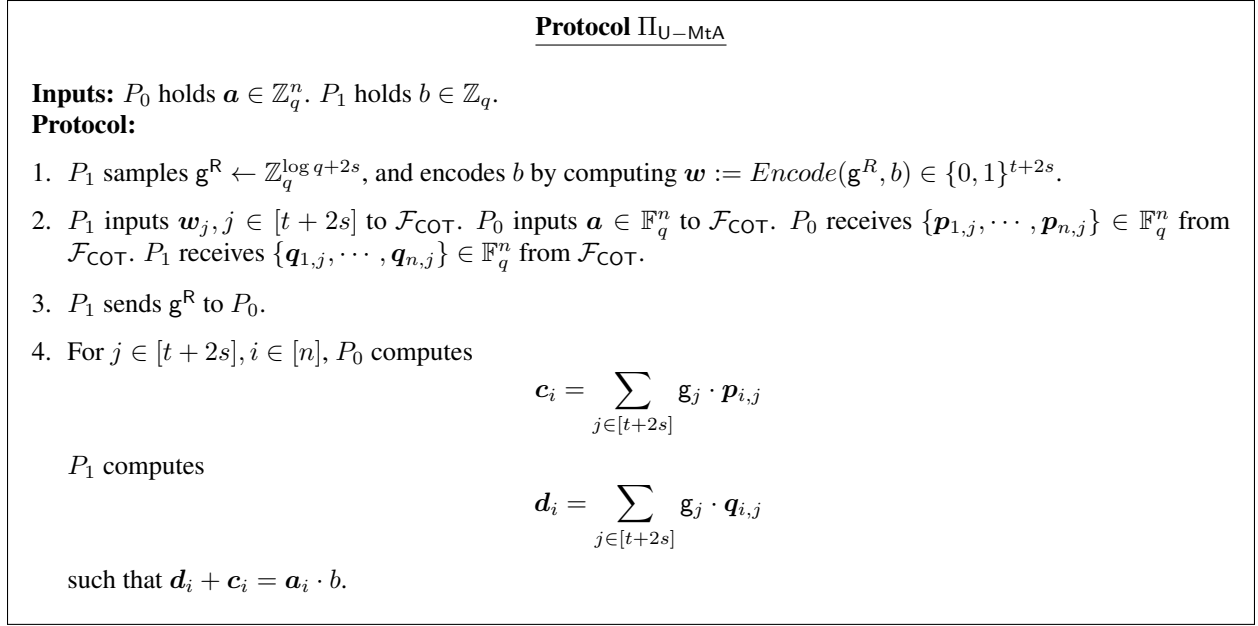


Figure A.1: The U-MtA protocol in $\mathcal{F}_{\text{COT}}$ -hybrid.

$\mathbf{d}_i + \mathbf{c}_i = \mathbf{a}_i \cdot b + \mathbf{f}_i$. $\mathbf{f}_i$ is denote as follows with respect to $\mathbf{w} = \text{Encode}(\mathbf{g}^R, b) \in \mathbb{Z}_q^{t+2s}$.

$$\mathbf{f}_i = \sum_{i \in [t]} \mathbf{g}_i \mathbf{e}_i \mathbf{w}_i + \sum_{i \in [2s]} \mathbf{g}_{t+i} \mathbf{e}_{t+i} \mathbf{w}_{t+i} \quad (\text{A.1})$$

Given $\mathbf{e}$, the correctness of MtA transformation depends on $\mathbf{w}$ and $\mathbf{g}^R$. Specifically, the transformation is correct when $\mathbf{f}_i = 0$. Still, this incorrectness will be caught by $\Pi_{\text{U-MtA}}$ in Appendix A.1.2 with a detailed proof.

A.1.2. OVUF with Improved Efficiency

In Chapter 3.2.2, we introduced the basic version of oblivious verifiable unpredictable protocol. In the context of Π_{OVUF} , when each client holds a set of n elements $(y_1, \dots, y_n)$ and collaborates with a server to compute OVUF, the Π_{OVUF} processes each input element $y_i, i \in [n]$ one by one. This sequential processing involves n iterations of Π_{MtA} . For each iteration of Π_{MtA} , it runs with input vector $(sk, \phi_1^i) \in \mathbb{Z}_q^2$ and $(\phi_2^i, y_i) \in \mathbb{Z}_q^2$ to compute additive share of $sk \cdot \phi_2^i$ and $\phi_1^i \cdot y_i$. In this chapter, we maximize the batch feature of MTA protocols and execute $sk \cdot \phi_2^i$ and $\phi_1^i \cdot y_i$ for

Protocol Π_{OVUF2}

Inputs and parameters: Hash function H modeled as RO. Client C_j holds vector $(y_1, \dots, y_n) \in \mathbb{Z}_q^n$.

Initialization: S chooses $\text{sk} \in \mathbb{Z}_q$, sets $\text{pk} = g^{\text{sk}}$, and sends pk to $\mathcal{F}_{\text{BB}}$.

Key query: Client C_j sends fetch to $\mathcal{F}_{\text{BB}}$ and receives pk .

Evaluation:

1. C_j checks if $g^{-y_i} = \text{pk}$ for each $i \in [n]$. If it is, C_j inserts i to set I and sets $y_i \leftarrow \mathbb{Z}_q, i \in I$.
2. Server S chooses $\phi_1 \leftarrow \mathbb{Z}_q^n$; client C_j chooses $\phi_2 \leftarrow \mathbb{Z}_q^n$.
3. S and C_j inputs sk and $\phi_2 \in \mathbb{Z}_q^n$ to $\Pi_{\text{U-MtA}}$, receives $A_1 \in \mathbb{Z}_q^n$ and $A_2 \in \mathbb{Z}_q^n$ respectively, such that $A_1 + A_2 = \text{sk} \cdot \phi_2$.
4. S and C_j inputs $\phi_1 \in \mathbb{Z}_q^n$ and $(y_1, \dots, y_n) \in \mathbb{Z}_q^n$ to Π_{MtA} , receives $B_1 \in \mathbb{Z}_q^n$ and $B_2 \in \mathbb{Z}_q^n$ respectively, such that $B_1^i + B_2^i = \phi_1^i \cdot y_i$ for each $i \in [n]$.
5. S samples $t \in [n]$, computes $V_S = H(g^{A_1^t})$, and sends (t, V_S) to C_j . C_j computes $V_R = H(\text{pk}^{\phi_2^t}/g^{A_2^t})$. C_j checks whether $V_R = V_S$ and aborts if they are not equal.
6. S sends $\mathbf{m}$ that $\mathbf{m}_i = \text{sk} \cdot \phi_1^i + A_1^i + B_1^i$ to C_j . C_j sends $\mathbf{u}$ that $\mathbf{u}_i = y_i \cdot \phi_2^i + A_2^i + B_2^i$ to S . Both S and C_j computes $\mathbf{v} = \mathbf{u} + \mathbf{m}$.
7. For each $i \in [n]$, S sends $\mathbf{h}_i = g^{\phi_1^i/v_i}$ to C_j . C_j sets $F_{\text{sk}}(y_i) = 1, i \in I$. For each $i \in [n] \setminus I$, C_j computes $F_{\text{sk}}(y_i) = \mathbf{h}_i \cdot g^{\phi_2^i/v_i}$.
8. C_j outputs $(F_{\text{sk}}(y_1), \dots, F_{\text{sk}}(y_n))$ if $e(g^{y_i} \cdot \text{pk}, F_{\text{sk}}(y_i)) = e(g, g)$ for each $i \in [n] \setminus I$. Otherwise it aborts.

Figure A.2: The OVUF2 protocol in $(\mathcal{F}_{\text{COT}}, \mathcal{F}_{\text{BB}})$ -hybrid model with sub-protocol Π_{MtA} and $\Pi_{\text{U-MtA}}$.

each $i \in [n]$ as follows:

1. Execute $\Pi_{\text{U-MtA}}$ to efficiently compute additive shares of $\text{sk} \cdot \phi_2^i, i \in [n]$.
2. Execute Π_{MtA} to compute additive shares of $\phi_1^i \cdot y_i, i \in [n]$.

The other parts of this optimized-oblivious verifiable unpredictable protocol Π_{OVUF2} follow the same idea as Π_{OVUF} . The detailed scheme is shown in Figure A.2. Its correctness can be verified directly. Security-wise, this protocol involves two different MtA transformations. For $\Pi_{\text{U-MtA}}$, the client C_j is regarded as the sender of $\mathcal{F}_{\text{COT}}$ and the one who executes a selective failure attack to Π_{OVUF2} . For Π_{MtA} , we assume server S as the sender of $\mathcal{F}_{\text{COT}}$ and the one who executes selective failure attacks to Π_{OVUF2} without loss of generality. We prove that Π_{OVUF2} (with sub-protocols Π_{MtA} and $\Pi_{\text{U-MtA}}$) is secure in the $(\mathcal{F}_{\text{COT}}, \mathcal{F}_{\text{BB}})$ -hybrid model.

Theorem 14. *If H is modeled as a random oracle, then protocol Π_{OVUF2} UC-realizes $\mathcal{F}_{\text{OVUF}}$ in the*

$(\mathcal{F}_{\text{COT}}, \mathcal{F}_{\text{BB}})$ -hybrid model with sub-protocol Π_{MtA} and $\Pi_{\text{U-MtA}}$.

Proof. Let $\mathcal{A}$ be a PPT adversary that allows to corrupt the server or the client. We construct a PPT simulator $\mathcal{S}$ with access to functionality $\mathcal{F}_{\text{OVUF}}$, which simulates the adversary's view. We consider the following two cases: malicious client and malicious server. We will prove that the joint distribution over the output of $\mathcal{A}$ and the honest party in the real world is indistinguishable from the joint distribution over the outputs of $\mathcal{S}$ and the honest party in the ideal world execution.

Corrupted client. Let $\mathcal{S}$ access to the $\mathcal{F}_{\text{OVUF}}$ as an honest client and interact with $\mathcal{A}$ as an honest server. $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

0. $\mathcal{S}$ emulates $\mathcal{F}_{\text{BB}}$, once it receives fetch from $\mathcal{A}$. $\mathcal{S}$ sends fetch to $\mathcal{F}_{\text{OVUF}}$ and receives pk. $\mathcal{S}$ sends pk to $\mathcal{A}$.

2-3. $\mathcal{S}$ simulates the sub-protocol $\Pi_{\text{U-MtA}}$ and acts as an honest receiver of $\mathcal{F}_{\text{COT}}$ below.

(1)-(2) $\mathcal{S}$ emulates $\mathcal{F}_{\text{COT}}$. $\mathcal{S}$ receives $\tau \in \mathbb{Z}_q^{nt+2ns}$ and sends $\mathbf{p} \leftarrow \mathbb{Z}_q^{nt+2ns}$ to $\mathcal{A}$. $\mathcal{S}$ checks whether the received τ satisfy the pattern that for $i \in [n]$, all the bits $\tau_j, j = (k-1)t+i, k \in [t+2s]$ are the same. Then, $\mathcal{S}$ extracts $\phi_2^i = \tau_j$.

(3) $\mathcal{S}$ sends $\mathbf{g}^R \leftarrow \mathbb{Z}_q^{\log q + 2s}$ to $\mathcal{A}$.

(4) $\mathcal{S}$ computes $A_2 \in \mathbb{Z}_q^n$ as an honest P_0 does in step 4 in $\Pi_{\text{U-MtA}}$.

4. $\mathcal{S}$ simulates the sub-protocol Π_{MtA} and acts as an honest sender of $\mathcal{F}_{\text{COT}}$ below.

(1)-(3) $\mathcal{S}$ emulates $\mathcal{F}_{\text{COT}}$ and receives $\mathbf{w} \in \mathbb{Z}_q^{nt+2s}$. $\mathcal{S}$ samples $\mathbf{q} \leftarrow \mathbb{Z}_q^{nt}, \mathbf{q}' \leftarrow \mathbb{Z}_q^{2ns}$ and sends them to $\mathcal{A}$.

(4) $\mathcal{S}$ receives $\mathbf{g}^R$ from $\mathcal{A}$. $\mathcal{S}$ computes y_i for each $i \in [n]$ as follows:

$$y_i = \sum_{j \in [t]} \mathbf{g}_j \mathbf{w}_{(i-1)t+j} + \sum_{j \in [t+1, t+2s]} \mathbf{g}_j \mathbf{w}_{nt+j}$$

(5) $\mathcal{S}$ computes $B_2 \in \mathbb{Z}_q$ as an honest P_1 does in step 5 in Π_{MtA} .

5. $\mathcal{S}$ samples $t \in [n]$, $V_S^* \leftarrow \mathbb{G}$ to $\mathcal{A}$. $\mathcal{S}$ emulates H . Once ϕ_2^t is extracted in step 3 and the received query $q = (\text{pk}^{\phi_2^t}/g^{A_2^t})$, $\mathcal{S}$ sends V_S^* to $\mathcal{A}$. Otherwise, $\mathcal{S}$ sends a random value to $\mathcal{A}$.
6. $\mathcal{S}$ sends $m_i^* \leftarrow \mathbb{Z}_q$ to $\mathcal{A}$. $\mathcal{S}$ receives u from $\mathcal{A}$ and computes v .
7. $\mathcal{S}$ sends $(\text{eval}, (y_1, \dots, y_n))$ to $\mathcal{F}_{\text{OVUF}}$ and waits to receive $(F_{\text{sk}}(y_1), \dots, F_{\text{sk}}(y_n))$. For each $i \in [n]$, $\mathcal{S}$ checks whether ϕ_2^i is extracted and $u_i = \phi_2^i \cdot y_i + A_2^i + B_2^i$. If all requirements are satisfied, $\mathcal{S}$ simulates $h_i^* = \frac{F_{\text{sk}}(y_i)}{g^{\phi_2^i/(m_i^*+u_i)}}$ and sends it to $\mathcal{A}$. Otherwise, $\mathcal{S}$ simulates $h_i^* \leftarrow \mathbb{G}$ and sends h^* to $\mathcal{A}$.
8. $\mathcal{S}$ aborts if $\mathcal{A}$ aborts and outputs what $\mathcal{A}$ outputs.

We are going to show the simulated execution is indistinguishable from the real protocol execution.

Hybrid $\mathcal{H}_0$. Same as real-world execution in $(\mathcal{F}_{\text{BB}}, \mathcal{F}_{\text{COT}})$ -hybrid.

Hybrid $\mathcal{H}_1$. This hybrid is identical to $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{\text{BB}}$, $\mathcal{F}_{\text{COT}}$, the random oracle, and simulates the messages to $\mathcal{A}$ as follows:

For **step 0**, $\mathcal{S}$ emulates $\mathcal{F}_{\text{BB}}$. Upon receiving fetch from $\mathcal{A}$, $\mathcal{S}$ sends fetch to $\mathcal{F}_{\text{OVUF}}$ and receives pk. $\mathcal{S}$ sends pk to $\mathcal{A}$. In hybrid $\mathcal{H}_0$, $\mathcal{F}_{\text{BB}}$ was initialized by an honest server with pk and sends it to $\mathcal{A}$ upon receiving fetch. Thus, the pk sends by $\mathcal{S}$ is same as the one in hybrid $\mathcal{H}_0$.

For **step 2-3**, $\mathcal{S}$ simulates sub-protocol $\Pi_{\text{U-MtA}}$. $\mathcal{S}$ emulates $\mathcal{F}_{\text{COT}}$, sends $p \leftarrow \mathbb{Z}_q^{nt+2ns}$ to $\mathcal{A}$. $\mathcal{S}$ also sends $g^R \leftarrow \mathbb{Z}_q^{\log q+2s}$ to $\mathcal{A}$. In Hybrid $\mathcal{H}_0$, p is uniformly distributed over $\mathbb{Z}_q^{nt+2ns}$ according to $\mathcal{F}_{\text{COT}}$. g^R is sampled by the client and uniformly distributed to $\mathcal{A}$. Thus, the sampled p and g^R are indistinguishable from Hybrid $\mathcal{H}_0$.

For **step 4**, $\mathcal{S}$ simulates sub-protocol Π_{MtA} . $\mathcal{S}$ emulates $\mathcal{F}_{\text{COT}}$, sends $q \leftarrow \mathbb{Z}_q^{nt}, q' \leftarrow \mathbb{Z}_q^{2ns}$ to $\mathcal{A}$. In Hybrid $\mathcal{H}_0$, q and q' are uniformly distributed according to $\mathcal{F}_{\text{COT}}$. Thus, the q and q' sampled by $\mathcal{S}$ are indistinguishable from those in Hybrid $\mathcal{H}_0$.

For **step 5**, $\mathcal{S}$ samples (t, V_S^*) to $\mathcal{A}$. $\mathcal{S}$ also programs random oracle H and sends V_S^* to $\mathcal{A}$ if it receives query $q = \text{pk}^{\phi_2^t}/g^{A_2^t}$ with ϕ_2^t extracted. Otherwise, $\mathcal{S}$ samples a random value to $\mathcal{A}$. In hybrid $\mathcal{H}_0$, an honest server uses sk that corresponding to pk in $\Pi_{\text{U-MtA}}$ and computes $V_S = H(g^{A_1^t})$ for a randomly sampled $t \leftarrow [n]$. Since $A_1^t + A_2^t = \text{sk} \cdot \phi_2^t$ holds for an honest server

and honest client, the received V_S equals to V_R if V_R is computed from $(\text{pk}^{\phi_2^t}/g^{A_2^t})$ honestly. Thus, the simulated V_R^* equals to V_S^* if $\mathcal{A}$ query random oracle honestly, which is indistinguishable from Hybrid $\mathcal{H}_0$. If $\mathcal{A}$ adds error $e \in \mathbb{Z}_q^{nt+2ns}$ in the execution of $\Pi_{\text{U-MtA}}$, $A_1^t + A_2^t = \text{sk} \cdot \phi_2^t + \mathbf{f}_t$ holds for honest server and the adversary. $\mathbf{f}_t$ is computed from Equation A.1. Since e is defined by $\mathcal{A}$ before knowing g^R , g^R is uniformly distributed over $\mathbb{Z}_q^{\log q+2s}$ to $\mathcal{A}$. For any given w and e , $\mathbf{f}_i$ is uniformly distributed over $\mathbb{Z}_q$ to $\mathcal{A}$. Thus, A_1^t is uniformly distributed over $\mathbb{Z}_q$ to $\mathcal{A}$ and V_S^* is indistinguishable from V_S in hybrid $\mathcal{H}_0$.

For step 6, $\mathcal{S}$ sends $\mathbf{m}_i^* \leftarrow \mathbb{Z}_q$ to $\mathcal{A}$. In hybrid $\mathcal{H}_0$, an honest server computes $\mathbf{m}_i = \phi_1^i \cdot \text{sk} + A_1^i + B_1^i$ and sends it to $\mathcal{A}$. If $\mathcal{A}$ acts honestly in previous steps, $\mathbf{m}_i$ satisfies the distribution that $\mathbf{m}_i + y_i \cdot \phi_2^i + A_2^i + B_2^i = \mathbf{v}_i = (\phi_1^i + \phi_2^i)(\text{sk} + y_i)$. Since ϕ_1^i is randomly sampled by an honest server, $\mathcal{A}$ has no idea about the distribution of $\mathbf{m}_i$. The simulated $\mathbf{m}_i^*$ is randomly uniform in $\mathbb{Z}_q$ as well, which is indistinguishable from hybrid $\mathcal{H}_0$. Thus, the view simulated by $\mathcal{S}$ is identical to hybrid $\mathcal{H}_0$. If there exists an error e sampled by $\mathcal{A}$ in sub-protocol $\Pi_{\text{U-MtA}}$, $\mathbf{m}_i$ satisfies the distribution that $\mathbf{m}_i + y_i \cdot \phi_2^i + A_2^i + B_2^i = \mathbf{v}_i = (\phi_1^i + \phi_2^i)(\text{sk} + y_i) + \text{diff}_i$. diff_i resulted from incorrect $\text{sk} \cdot \phi_2^i$, computed by Equation A.1. As analyzed above, diff_i is uniformly distributed over $\mathbb{Z}_q$ to $\mathcal{A}$. Thus, $\mathbf{m}_i$ is uniformly distributed over $\mathbb{Z}_q$ to $\mathcal{A}$, which is indistinguishable from the simulated $\mathbf{m}_i^*$.

Thus, the view simulated by $\mathcal{S}$ is identical to hybrid $\mathcal{H}_0$.

For step 7, $\mathcal{S}$ sends $(\text{eval}, (y_1, \dots, y_n))$ to $\mathcal{F}_{\text{OVUF}}$ and waits for $(F_{\text{sk}}(y_1), \dots, F_{\text{sk}}(y_n))$. $\mathcal{S}$ checks whether ϕ_2^i is extractable and the received $\mathbf{u}_i$ is computed from $y_i \cdot \phi_2^i + A_2^i + B_2^i$ correctly. If it is, $\mathcal{S}$ sends $\mathbf{h}_i^* = \frac{F_{\text{sk}}(y_i)}{g^{\phi_2^i/(\mathbf{m}_i^* + \mathbf{u}_i)}}$ to $\mathcal{A}$, such that $\mathbf{h}_i^* \cdot g^{\phi_2^i/(\mathbf{m}_i^* + \mathbf{u}_i)} = F_{\text{sk}}(y_i)$. Otherwise, $\mathcal{S}$ samples $\mathbf{h}_i^* \leftarrow \mathbb{G}$ and sends it to $\mathcal{A}$.

In hybrid $\mathcal{H}_0$, an honest server sends $\mathbf{h}_i = g^{\phi_1^i/(\mathbf{m}_i + \mathbf{u}_i)}$ to $\mathcal{A}$. If $\mathbf{u}_i$ is computed honestly from $y_i \cdot \phi_2^i + A_2^i + B_2^i$, where ϕ_2^i is extracted, then $\mathbf{h}_i \cdot g^{\phi_2^i/(\mathbf{m}_i + \mathbf{u}_i)} = F_{\text{sk}}(y_i)$. Thus, the view of $\mathbf{h}_i^*$ in hybrid $\mathcal{H}_0$ is identical to $\mathbf{h}_i$ in Hybrid $\mathcal{H}_0$. If $\mathbf{u}_i$ is computed honestly, but there exists an error e sampled by $\mathcal{A}$ in sub-protocol $\Pi_{\text{U-MtA}}$ in hybrid $\mathcal{H}_0$, then $\mathbf{m}_i + \mathbf{u}_i = (\text{sk} + y_i)(\phi_1^i + \phi_2^i) + \text{diff}_i$. Since diff_i is uniformly distributed over $\mathbb{Z}_q$, $\mathbf{h}_i \cdot g^{\phi_2^i/(\mathbf{m}_i + \mathbf{u}_i)} = F_{\text{sk}}(y_i)$ with negligible probability.

In Hybrid $\mathcal{H}_1$, $\mathcal{S}$ simulates $\mathbf{h}_i^* \leftarrow \mathbb{G}$. $\mathbf{h}_i^* \cdot g^{\phi_2^i / (m_i^* + u_i)} = F_{\text{sk}}(y_i)$ is negligible and indistinguishable from hybrid $\mathcal{H}_0$. If u_i is not computed honestly, then $m_i + u_i \neq (\phi_1^i + \phi_2^i)(\text{sk} + y_i)$ and thus $\mathbf{h}_i \cdot g^{\phi_2^i / (m_i + u_i)} \neq F_{\text{sk}}(y_i)$. In hybrid $\mathcal{H}_1$, $\mathcal{S}$ simulates $\mathbf{h}_i^* \leftarrow \mathbb{G}$, we have $\mathbf{h}_i^* \cdot g^{\phi_2^i / (m_i^* + u_i)} = F_{\text{sk}}(y_i)$ w.p. $2^{-\log q}$, which is indistinguishable from hybrid $\mathcal{H}_0$.

Thus, the view simulated by $\mathcal{S}$ is identical to hybrid $\mathcal{H}_0$.

Corrupted server. Let $\mathcal{S}$ access to the $\mathcal{F}_{\text{OVUF}}$ as an honest server and interact with $\mathcal{A}$ as an honest client. $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

0. $\mathcal{S}$ emulates $\mathcal{F}_{\text{BB}}$, once it receives the pk from $\mathcal{A}$, $\mathcal{S}$ stores pk and ignores subsequent messages from $\mathcal{A}$. $\mathcal{S}$ sends (init, pk) to $\mathcal{F}_{\text{OVUF}}$.

2-3. $\mathcal{S}$ simulates the sub-protocol $\Pi_{\text{U-MtA}}$ and acts as an honest sender below.

(1)-(2) $\mathcal{S}$ emulates $\mathcal{F}_{\text{COT}}$. $\mathcal{S}$ receives $\mathbf{w} \in \mathbb{Z}_q^{t+2s}$ and sends $\mathbf{q} \leftarrow \mathbb{Z}_q^{nt+2ns}$ to $\mathcal{A}$.

(3) $\mathcal{S}$ receives $\mathbf{g}^R \in \mathbb{Z}_q^{\log q + 2s}$ from $\mathcal{A}$. $\mathcal{S}$ recover sk' as: $sk' = \sum_{i \in [2 \log q + 2s]} \mathbf{g}_i \mathbf{w}_i$.

(4) $\mathcal{S}$ computes $A_1 \in \mathbb{Z}_q^n$ as an honest P_1 does in step 4 in $\Pi_{\text{U-MtA}}$.

4. $\mathcal{S}$ simulates sub-protocol Π_{MtA} and acts as an honest receiver below.

(1)-(3) $\mathcal{S}$ emulates $\mathcal{F}_{\text{COT}}$ and receives a vector $\boldsymbol{\tau} \in \mathbb{Z}_q^{n(t+2s)}$. $\mathcal{S}$ samples $\mathbf{p} \leftarrow \mathbb{Z}_q^{nt}$, $\mathbf{p}' \leftarrow \mathbb{Z}_q^{2ns}$ and sends them to $\mathcal{A}$. $\mathcal{S}$ checks whether the received $\boldsymbol{\tau} \in \mathbb{Z}_q^{n(t+2s)}$ satisfies a pattern that for $i \in [n]$, all the bits τ_j , $j \in [(i-1)t+1, it] \cup j = nt+i+(l-1)n$, $l \in [2s]$ are the same.

Then, it extracts $\phi_1^i = \tau_j$.

(4) $\mathcal{S}$ samples $\mathbf{g}^R \leftarrow \mathbb{Z}_q^{\log q + 2s}$ and sends it to $\mathcal{A}$.

(5) $\mathcal{S}$ computes $B_1 \in \mathbb{Z}_q^n$ as an honest P_0 does in step 5 in Π_{MtA} .

5. $\mathcal{S}$ emulates random oracle H and receives query q from $\mathcal{A}$. $\mathcal{S}$ samples V_S^* to $\mathcal{A}$ and records (q, V_S^*) . Once $\mathcal{S}$ receives (t, V_S^*) from $\mathcal{A}$, $\mathcal{S}$ first checks whether $g^{\text{sk}'} = \text{pk}$ and aborts if not. Then, $\mathcal{S}$ checks whether the corresponded $q = g^{A_1^t}$. If it is, $\mathcal{S}$ continue; Otherwise, $\mathcal{S}$ aborts.

6. $\mathcal{S}$ receives $\mathbf{m}$ from $\mathcal{A}$. $\mathcal{S}$ sends $\mathbf{u}_i^* \leftarrow \mathbb{Z}_q$ to $\mathcal{A}$.

7. $\mathcal{S}$ waits to receive $\mathbf{h}$.
8. For each i th iteration, if ϕ_1^i is extracted, $\mathcal{S}$ checks whether $\mathbf{h}_i = g^{\phi_1^i/v_i^*}$, $\mathbf{m}_i = \phi_1^i \cdot \text{sk}' + A_1^i + B_1^i$ and sends sk' to $\mathcal{F}_{\text{OVUF}}$. Otherwise, $\mathcal{S}$ aborts.

We are going to show the simulated execution is indistinguishable from the real protocol execution.

Hybrid $\mathcal{H}_0$. Same as real-world execution in $(\mathcal{F}_{\text{BB}}, \mathcal{F}_{\text{COT}})$ -hybrid model.

Hybrid $\mathcal{H}_1$. This hybrid is identical to $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{\text{BB}}$, $\mathcal{F}_{\text{COT}}$, the random oracle, and generates the messages to $\mathcal{A}$ as follows:

For **Step 2-3**, $\mathcal{S}$ simulates sub-protocol Π_{MtA} . $\mathcal{S}$ emulates $\mathcal{F}_{\text{COT}}$, sends $\mathbf{p} \leftarrow \mathbb{Z}_q^{nt}$, $\mathbf{p}' \leftarrow \mathbb{Z}_q^{2ns}$ to $\mathcal{A}$. $\mathcal{S}$ also samples $\mathbf{g}^R \leftarrow \mathbb{Z}_q^{\log q + 2s}$ and sends it to $\mathcal{A}$. In Hybrid $\mathcal{H}_0$, $\mathbf{p}, \mathbf{p}'$ are uniformly distributed according to $\mathcal{F}_{\text{COT}}$. $\mathbf{g}^R$ is sampled by client and uniformly distributed over $\mathbb{Z}_q^{\log q + 2s}$ to $\mathcal{A}$. Thus, the simulated $\mathbf{p}, \mathbf{p}', \mathbf{g}^R$ are indistinguishable from Hybrid $\mathcal{H}_0$.

For **step 4**, $\mathcal{S}$ simulates sub-protocol $\Pi_{\text{U-MtA}}$. $\mathcal{S}$ emulates $\mathcal{F}_{\text{COT}}$, sends $\mathbf{q} \leftarrow \mathbb{Z}_q^{n(t+2s)}$ to $\mathcal{A}$. In Hybrid $\mathcal{H}_0$, $\mathbf{q}$ is uniformly distributed over $\mathbb{Z}_q^{n(t+2s)}$ according to $\mathcal{F}_{\text{COT}}$. Thus, the sampled $\mathbf{q}$ is indistinguishable from Hybrid $\mathcal{H}_0$.

For **Step 6**, $\mathcal{S}$ receives $\mathbf{m}$ from $\mathcal{A}$, and sends $\mathbf{u}_i^* \leftarrow \mathbb{Z}_q$ to $\mathcal{A}$. In Hybrid $\mathcal{H}_0$, if $\mathcal{A}$ acts honestly in previous steps, we have $\mathbf{u}_i + \phi_1^i \cdot \text{sk} + A_1^i + B_1^i = \mathbf{v}_i = (\text{sk} + y_i)(\phi_1^i + \phi_2^i)$. Since $\phi_2^i \leftarrow \mathbb{Z}_q$ to $\mathcal{A}$, we have $\mathbf{u}_i$ is uniform distributed over $\mathbb{Z}_q$ to $\mathcal{A}$. If $\mathcal{A}$ adds error $\mathbf{e}$ in step 4, an honest client computes $\mathbf{u}_i$ such that $\mathbf{u}_i + \phi_1^i \cdot \text{sk} + A_1^i - B_1^i = \mathbf{v}_i = (\text{sk} + y_i)(\phi_1^i + \phi_2^i) + \text{diff}_i$. Since diff_i is uniform distributed over $\mathbb{Z}_q$, $\mathbf{u}_i$ is uniform distributed over $\mathbb{Z}_q$ to $\mathcal{A}$ as well. If $\mathcal{A}$ uses $\text{sk}' \neq \text{sk}$ in step 2-3, $\mathbf{u}_i + \phi_1^i \cdot \text{sk}' + A_1^i + B_1^i = \mathbf{v}_i = (\text{sk}' + y_i)(\phi_1^i + \phi_2^i)$. $\mathbf{u}_i$ is still uniformly distributed as ϕ_2^i is distributed uniformly to $\mathcal{A}$. Thus, the simulated $\mathbf{u}_i^*$ is indistinguishable from the distribution of $\mathbf{u}_i$ in Hybrid $\mathcal{H}_0$.

Hybrid $\mathcal{H}_2$. This hybrid is identical to $\mathcal{H}_1$ except $\mathcal{S}$ aborts at **step 5** in the following conditions:

- 1) the q corresponding to the received V_S^* not equal to $g^{A_1^t}$; 2) $g^{\text{sk}'} \neq \text{pk}$. $\mathcal{S}$ also aborts at **Step 8** in the following conditions: 1) $g^{\text{sk}'} \neq \text{pk}$; 2) ϕ_1^i s are not extractable; 3) $\mathbf{m}_i \neq \phi_1^i \cdot \text{sk}' + A_1^i + B_1^i$ or $\mathbf{h}_i^* \neq g^{\phi_1^i/(m_i+u_i^*)}$.

For **step 5** in hybrid $\mathcal{H}_1$, an honest client aborts if the received $V_S \neq V_R$. The client computes $V_R = H(\text{pk}^{\phi_2^t}/g^{A_2^t})$. For each $\text{pk}^{\phi_2^t}/g^{A_2^t}$, it equals to $g^{\text{sk} \cdot \phi_2^t - A_2^t} = g^{A_1^t}$, where sk is corresponded to pk . When adversary use correct sk but manipulate V_S by using inconsistent query $q \neq g^{A_1^t}$, an honest client aborts in hybrid $\mathcal{H}_1$, which is indistinguishable from condition (1) in hybrid $\mathcal{H}_2$. Adversary $\mathcal{A}$ might use invalid sk' that $g^{\text{sk}'} \neq \text{pk}$ in Π_{MtA} , then both parties holds equation $A_1^t + A_2^t = \text{sk}' \cdot \phi_2^t$. Thus, with V_S computed from A_1^t , $V_S = g^{A_1^t} = g^{\text{sk}' \cdot \phi_2^t - A_2^t} \neq V_R = \text{pk} \cdot g^{\phi_2^t - A_2^t}$. Moreover, because of the uniformity of ϕ_2^t and A_2^t , $\mathcal{A}$ is not able to construct $A_1^{t'}$ that $g^{A_1^{t'}} = \text{pk} \cdot g^{\phi_2^t - A_2^t}$ either. Thus, the client aborts with all but negligible probability, which is indistinguishable from condition (2) in hybrid $\mathcal{H}_2$.

For **step 8** in hybrid $\mathcal{H}_1$, an honest client aborts when $F_{\text{sk}}(y_i)$ does not satisfy $e(g^{y_i} \cdot \text{pk}, F_{\text{sk}}(y_i)) = e(g, g)$, where $F_{\text{sk}}(y_i) = \mathbf{h}_i \cdot g^{\phi_2^i/(m_i+u_i)}$. If $g^{\text{sk}'} \neq \text{pk}$, the probability that $m_i + u_i = (\text{sk} + y_i)(\phi_1^i + \phi_2^i)$ with negligible probability. Thus, the value of $\mathbf{h}_i \cdot g^{\phi_2^i/(m_i+u_i)}$ satisfy the verification equation with all but negligible probability. The honest client aborts with all but negligible probability which is indistinguishable from condition (1) in hybrid $\mathcal{H}_2$. For the t th iteration, if $\mathcal{A}$ adds error e to τ and ϕ_2^t is not extractable, $\text{diff}_i \neq 0$ with all but negligible probability. Thus, $m_i + u_i \neq (\text{sk} + y_i)(\phi_1^i + \phi_2^i)$ with all but negligible probability. The value of $\mathbf{h}_i \cdot g^{\phi_2^i/(m_i+u_i)}$ satisfy the verification equation with all but negligible probability. The honest client aborts with all but negligible probability which is indistinguishable from condition (2) in hybrid $\mathcal{H}_2$. If $\mathcal{A}$ sends either wrong m_i or wrong $\mathbf{h}_i$ in protocol, it will result in wrong $F_{\text{sk}}(y_i)$ that does not satisfy $e(g^{y_i} \cdot \text{pk}, F_{\text{sk}}(y_i)) = e(g, g)$. The honest client aborts with all but negligible probability which is indistinguishable from condition (3) in hybrid $\mathcal{H}_2$.

Therefore, this hybrid is identically distributed as the previous one. This completes the proof. $\square$

APPENDIX B

Appendix for Chapter 4

B.1. Hybrid Commitment Parameters based on EA-Code

Boyle et al. [97] introduced a method of choosing conservative parameters for EA-code, which is secure against the current known attacks [98]. They use a random walk on a Markov chain to determine the minimum distance for the EA code. As all variants of LPN generalize naturally to larger fields [97], we instantiate our commitment of $\mathbb{F}_{2^{128}}$ based on the conservative parameters constraints of $\mathbb{F}_2$ as follows.

B.1.1. Conservative parameters

Based on [97, Theorem 3.10], for $\mathbf{H} \leftarrow \text{EAGen}(n, N, \frac{C \ln N}{N})$, $\mathbf{H}$ achieves a minimum distance of δN with a probability of at least

$$\eta(\kappa) \geq 1 - 2RN^{-2\beta^2 C+2} = 1 - 2nN^{-2(1/2-\delta)^2 C+1} . \quad (\text{B.1})$$

For an error distribution with an expected weight of t , we have

$$\varepsilon_d \leq (1 - 2t/N)^{\delta N} / 2 \leq e^{-2t\delta} . \quad (\text{B.2})$$

We consider two scenarios: a failure probability of at most 0.01 (i.e., $\eta \geq 0.99$). Our chosen failure probability falls within the range of $[0.0000148, 0.174]$, as indicated in the initial work [97, Figure 8], and is deemed reasonable. To achieve 128-bit security against linear tests, we have $-\log(\varepsilon_d) = 128 - \log(N)$ because computing a dot-product requires N operations [97]. We first choose an appropriate value for t and then determine δ using Equation (B.2), which is $\delta \geq \frac{128 - \log(N)}{2 \log(e) \cdot t}$. Next,

to ensure $\eta \geq 0.99$, we use Equation (B.1) and choose C such that

$$\log(2n) + \log(N) \cdot (-2(1/2 - \delta)^2 C + 1) \leq \log(0.01) .$$

Finally, we check the parameters to satisfy the following conditions from [97, Theorem 3.10]:

$$R < \min \left\{ \frac{2}{\ln 2} \cdot \frac{1 - e^{-1}}{1 + e^{-1}} \cdot \beta^2, \frac{2}{e} \right\} \quad \text{and} \quad C > \frac{1}{\beta^2} .$$

shown in Theorem [97, Theorem 3.10], where $\beta = 1/2 - \delta$.

B.1.2. Aggressive parameters

[97] proposed aggressive parameters by empirically estimating the average minimum row-weight of $\mathbf{H} \leftarrow \text{EAGen}(n, N, \ell)$ using exponential regression in $\log(n)$, with $N = 5n$ and $\ell \in \{7, 9, 11\}$. Specifically, $\mathbf{H}$ achieves a minimum distance of δN , where

$$\delta \approx \begin{cases} 0.4646 \cdot e^{-0.1012 \cdot \log(n)}, & \text{if } \ell = 7 \\ 0.4415 \cdot e^{-0.0783 \cdot \log(n)}, & \text{if } \ell = 9 \\ 0.4337 \cdot e^{-0.06383 \cdot \log(n)}, & \text{if } \ell = 11 . \end{cases}$$

Then, we determine t using Equation (B.2), which is $t \geq \frac{128 - \log(N)}{2 \log(e) \cdot \delta}$.

B.2. Primal-LPN based C-VOLE

We give primal-LPN based C-VOLE construction in this section. It follows the same idea in Section 4.2.2, except minor changes regarding the format of primal-LPN commitment of vector e_u . As we introduced, generating any C-VOLE of designated vector x is start with generating C-VOLE of a designated regular- w vector e_u . As e_u is committed as $\text{com}_{e_u} = e_u \cdot \mathbf{A}_1 + r \cdot \mathbf{A}_2 + e_r$ and $\text{decom}_{e_u} = \{r, e_r\}$ under primal-LPN commitment in Section 4.2.1.1, we generate consistent $\langle e_u \rangle$ as follows: 1) generate $\langle e_u \rangle$ with P_A input e_u , 2) generate $\langle e_r \rangle$ with P_A input e_r , 3) generate

$\langle \mathbf{r} \rangle$ with P_A input $\mathbf{r}$, 4) P_A computes $H_3(M[\text{com}_{e_u}])$ and sends it to P_B for consistency check. The consistency check works as follows:

$$\begin{aligned} M[\text{com}_{e_u}] &= M[e_u] \cdot \mathbf{A}_1 + M[\mathbf{r}] \cdot \mathbf{A}_2 + M[e_r] \\ K[\text{com}_{e_u}] &= K[e_u] \cdot \mathbf{A}_1 + K[\mathbf{r}] \cdot \mathbf{A}_2 + K[e_r] \end{aligned}$$

If $\langle e_u \rangle$, $\langle e_r \rangle$, and $\langle \mathbf{r} \rangle$ are generated with P_A input consistent e_u , e_r , and $\mathbf{r}$ used in com_{e_u} , $\langle \text{com}_{e_u} \rangle$ should hold and be checked successfully by P_B that $H_3(M[\text{com}_{e_u}]) = H_3(K[\text{com}_{e_u}] + \text{com}_{e_u} \cdot \Delta)$.

Then, we can recover $\langle \mathbf{x} \rangle$ from $\langle e_u \rangle$ under $\text{Hcom}_{\mathbf{x}}$ as easily as dual-LPN-based C-VOLE does in Section 4.2.2. The detailed protocol is shown in Figure B.1. We also prove that the protocol is secure against malicious adversaries in Theorem 15.

Protocol $\Pi_{\text{C-VOLE-primal}}$

Parameters: A public hybrid commitment of $\mathbf{x} \in \mathbb{F}^m$ defined in Section 4.2.1.2, that $\text{Hcom}_{\mathbf{x}} = (\text{com}_{e_u} \in \mathbb{F}^N, \mathbf{x}^* = e_u \cdot \mathbf{H} - \mathbf{x} \in \mathbb{F}^m)$. A primal LPN commitment scheme defined in Section 4.2.1.1 that $(\text{com}_{e_u} = e_u \cdot \mathbf{A}_1 + \mathbf{r} \cdot \mathbf{A}_2 + e_r, \text{decom}_{e_u} = \{\mathbf{r}, e_r\}) \leftarrow \text{Com}(e_u)$, where $e_u \in \mathbb{F}^{w \cdot 2^L}$, $\mathbf{r} \in \mathbb{F}^k$, $e_r \in \mathbb{F}^{t \cdot 2^{L'}}$, $N = t \cdot 2^{L'}$. A Collision-Resistant Hash Function H_3 .

Inputs: P_A holds vector $\mathbf{x} \in \mathbb{F}^m$, $\text{Hdecom}_{\mathbf{x}} = (e_u \in \mathbb{F}^{w \cdot 2^L}, \mathbf{r} \in \mathbb{F}^k, e_r \in \mathbb{F}^{t \cdot 2^{L'}})$.

Compute:

1. Both P_A and P_B call $\mathcal{F}_{\text{MPVOLE}}$ with input (init). P_B receives $\Delta \in \mathbb{F}$ from $\mathcal{F}_{\text{MPVOLE}}$.
2. P_A calls $\mathcal{F}_{\text{MPVOLE}}$ with input (extend, $e_u \in \mathbb{F}^{w \cdot 2^L}$, L, w). P_B calls $\mathcal{F}_{\text{MPVOLE}}$ with input (extend, L, w). The functionality returns $M[e_u] \in \mathbb{F}^{w \cdot 2^L}$ to P_A and $K[e_u] \in \mathbb{F}^{w \cdot 2^L}$ to P_B such that $M[e_u] = K[e_u] + \Delta \cdot e_u \in \mathbb{F}^{w \cdot 2^L}$. If either party receives **abort** from $\mathcal{F}_{\text{MPVOLE}}$ in any of these executions, it aborts.
3. P_A calls $\mathcal{F}_{\text{MPVOLE}}$ with input (extend, $e_r \in \mathbb{F}^{t \cdot 2^{L'}}$, L', t). P_B calls $\mathcal{F}_{\text{MPVOLE}}$ with input (extend, L', t). The functionality returns $M[e_r] \in \mathbb{F}^{t \cdot 2^{L'}}$ to P_A and $K[e_r] \in \mathbb{F}^{t \cdot 2^{L'}}$ to P_B such that $M[e_r] = K[e_r] + \Delta \cdot e_r \in \mathbb{F}^{t \cdot 2^{L'}}$. If either party receives **abort** from $\mathcal{F}_{\text{MPVOLE}}$ in any of these executions, it aborts.
4. P_A calls $\mathcal{F}_{\text{VOLE}}$ with input (extend, $\mathbf{r} \in \mathbb{F}^k, k$); P_B calls $\mathcal{F}_{\text{VOLE}}$ with input (extend, k). The functionality returns $M[\mathbf{r}] \in \mathbb{F}^k$ to P_A and $K[\mathbf{r}] \in \mathbb{F}^k$ to P_B , such that $M[\mathbf{r}] = K[\mathbf{r}] + \mathbf{r} \cdot \Delta$. If either party receives **abort** from $\mathcal{F}_{\text{VOLE}}$ in any of these executions, it aborts.
5. P_A computes $M[\text{com}_{e_u}] = M[e_u] \cdot \mathbf{A}_1 + M[\mathbf{r}] \cdot \mathbf{A}_2 + M[e_r]$ and sends $H_3(M[\text{com}_{e_u}])$ to P_B ; P_B computes $K[\text{com}_{e_u}] = K[e_u] \cdot \mathbf{A}_1 + K[\mathbf{r}] \cdot \mathbf{A}_2 + K[e_r]$ and **value** = $K[\text{com}_{e_u}] + \text{com}_{e_u} \cdot \Delta \in \mathbb{F}^N$. P_B checks whether $H_3(M[\text{com}_{e_u}]) = H_3(\text{value})$. If not, P_B aborts.
6. P_A outputs $(\mathbf{x}, M[\mathbf{x}])$, where $M[\mathbf{x}] = M[e_u] \cdot \mathbf{H} \in \mathbb{F}^m$. P_B outputs $(K[\mathbf{x}], \Delta)$, where $K[\mathbf{x}] = K[e_u] \cdot \mathbf{H} + \mathbf{x}^* \Delta \in \mathbb{F}^m$.

Figure B.1: The committed-VOLE protocol based on primal-LPN-based hybrid commitment.

Theorem 15. *Protocol $\Pi_{C-VOLE-primal}$ in Figure B.1 securely instantiated $\mathcal{F}_{C-VOLE}$ in Figure 4.1 in $(\mathcal{F}_{MPVOLE}, \mathcal{F}_{VOLE})$ -hybrid model against any malicious adversaries.*

Proof. Let $\mathcal{A}$ be a PPT adversary that corrupt P_A or P_B . We construct a PPT simulator $\mathcal{S}$ with access to $\mathcal{F}_{C-VOLE}$ and simulates the adversary's view. We will prove that the joint distribution over the output of $\mathcal{A}$ and the honest party in the real world is indistinguishable from the joint distribution over the output of $\mathcal{S}$ and the honest party in the ideal world.

Corrupted P_A . Let $\mathcal{S}$ access $\mathcal{F}_{C-VOLE}$ as an honest P_A and interact with $\mathcal{A}$ as an honest P_B . $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

1. $\mathcal{S}$ holds public commitment of $\mathbf{x}$, that $Hcom_{\mathbf{x}} = (com_{e_u}, \mathbf{x}^*)$. $\mathcal{S}$ emulates $\mathcal{F}_{MPVOLE}$ and receives (init) from $\mathcal{A}$.
2. $\mathcal{S}$ emulates $\mathcal{F}_{MPVOLE}$ and receives (extend, $e'_u \in \mathbb{F}^{w \cdot 2^L}, L, w$) from $\mathcal{A}$. $\mathcal{S}$ receives $M[e'_u] \in \mathbb{F}^{w \cdot 2^L}$ from $\mathcal{A}$. $\mathcal{S}$ computes $\mathbf{x}' = e_u \cdot \mathbf{H} - \mathbf{x}^*$.
3. $\mathcal{S}$ emulates $\mathcal{F}_{MPVOLE}$ and receives (extend, $e'_r \in \mathbb{F}^{t \cdot 2^{L'}}, L', t$) from $\mathcal{A}$. $\mathcal{S}$ receives $M[e'_r] \in \mathbb{F}^{t \cdot 2^{L'}}$ from $\mathcal{A}$.
4. $\mathcal{S}$ emulates $\mathcal{F}_{VOLE}$ and receives (extend, $\mathbf{r}' \in \mathbb{F}^k, k$) from $\mathcal{A}$. $\mathcal{S}$ receives $M[\mathbf{r}'] \in \mathbb{F}^k$ to $\mathcal{A}$. $\mathcal{S}$ sends $(\mathbf{x}', decom_{\mathbf{x}'} = (e'_u, \mathbf{r}', e'_r))$ to $\mathcal{F}_{C-VOLE}$. If $\mathcal{S}$ receives **abort** from $\mathcal{F}_{C-VOLE}$, $\mathcal{S}$ aborts at step (5).
5. $\mathcal{S}$ receives $H_3(M[com_{e_u}])'$ from $\mathcal{A}$. $\mathcal{S}$ checks if $H_3(M[com_{e_u}])' = H_3(M[e'_u] \cdot \mathbf{A}_1 + M[\mathbf{r}'] \cdot \mathbf{A}_2 + M[e'_r])$. If not, $\mathcal{S}$ aborts.
6. $\mathcal{S}$ computes $M[\mathbf{x}'] = M[e'_u] \cdot \mathbf{H}$ and sends $M[\mathbf{x}']$ to $\mathcal{F}_{C-VOLE}$. $\mathcal{S}$ outputs what $\mathcal{A}$ outputs.

We are going to show the simulated execution is indistinguishable from real-world protocol execution.

Hybrid $\mathcal{H}_0$ Same as real-world execution in $\mathcal{F}_{MPVOLE}, \mathcal{F}_{VOLE}$ model.

Hybrid $\mathcal{H}_1$ Same as Hybrid $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{MPVOLE}$ and $\mathcal{F}_{VOLE}$. Notice that $\mathcal{S}$ does not simulate any message to $\mathcal{A}$.

Hybrid $\mathcal{H}_2$ Same as Hybrid $\mathcal{H}_1$ except $\mathcal{S}$ aborts at step (5) if $\mathcal{S}$ receives abort from $\mathcal{F}_{C-VOLE}$ or the received $H_3(M[\text{com}_{e_u}])' \neq H_3(M[e'_u] \cdot \mathbf{A}_1 + M[r'] \cdot \mathbf{A}_2 + M[e'_r])$.

In hybrid $\mathcal{H}_1$, an honest P_B aborts at step (5) if the received $H_3(M[\text{com}_{e_u}])' \neq H_3(K[e'_u] \cdot \mathbf{A}_1 + K[r'] \cdot \mathbf{A}_2 + K[e'_r] + \text{com}_{e_u} \cdot \Delta)$. There are the following abort conditions with abort probability indistinguishable from hybrid $\mathcal{H}_2$:

1) $\mathcal{A}$ uses $e'_u \neq e_u$, $e'_r \neq e_r$, and $r' \neq r$ to $\mathcal{F}_{MPVOLE}$. $\mathcal{A}$ computes $M[\text{com}_{e_u}]'$ honestly from $M[e'_u] \cdot \mathbf{A}_1 + M[r'] \cdot \mathbf{A}_2 + M[e'_r]$. Any (e'_u, e'_r, r') inconsistent with (e_u, e_r, r) will result the inequality above with the probability of statistical binding failure mentioned in Theorem 6. In this hybrid, any inconsistency will be checked by the $\mathcal{F}_{C-VOLE}$ with the same binding failure and result in an abort at step (5), which is indistinguishable from hybrid $\mathcal{H}_1$. 2) $\mathcal{A}$ uses $e'_u \neq e_u$, $e'_r \neq e_r$, and $r' \neq r$ to $\mathcal{F}_{MPVOLE}$. $\mathcal{A}$ samples $M[\text{com}_{e_u}]'$ to $\mathcal{S}$. The consistency check passes if and only if $\mathcal{A}$ samples $M[\text{com}_{e_u}]' = K[e'_u] \cdot \mathbf{A}_1 + K[r'] \cdot \mathbf{A}_2 + K[e'_r] + \text{com}_{e_u} \cdot \Delta$. However, since $K[e'_u]$, $K[e'_r]$, $K[r]$, and Δ are uniformly distributed over $\mathbb{F}^{w \cdot 2^L}$, $\mathbb{F}^{t \cdot 2^{L'}}$, $\mathbb{F}^k$ and $\mathbb{F}$, respectively, the consistency check fails with all but negligible probability. In this hybrid, given inconsistent (e'_u, e'_r, r') pair, $\mathcal{S}$ aborts except the failure probability of binding, which is also negligible and thus indistinguishable from hybrid $\mathcal{H}_1$. 3) $\mathcal{A}$ uses $e'_u = e_u$, $e'_r = e_r$, and $r' = \text{vecr}$ to $\mathcal{F}_{MPVOLE}$. $\mathcal{A}$ samples $M[\text{com}_{e_u}]'$ to $\mathcal{S}$. The consistency check passes if and only if $\mathcal{A}$ samples $M[\text{com}_{e_u}]' = K[e'_u] \cdot \mathbf{A}_1 + K[r'] \cdot \mathbf{A}_2 + K[e'_r] + \text{com}_{e_u} \cdot \Delta$, which is equivalent to checking $M[\text{com}_{e_u}]' = M[e'_u] \cdot \mathbf{A}_1 + M[r'] \cdot \mathbf{A}_2 + M[e'_r] + \text{com}_{e_u} \cdot \Delta$ in this hybrid. Thus, the abort probability is indistinguishable from hybrid $\mathcal{H}_1$.

Therefore, this hybrid is identically distributed as $\mathcal{H}_1$.

Corrupted P_B . Let $\mathcal{S}$ access $\mathcal{F}_{C-VOLE}$ as an honest P_B and interact with $\mathcal{A}$ as an honest P_A . $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

1. $\mathcal{S}$ emulates $\mathcal{F}_{MPVOLE}$ and receives (init) from $\mathcal{A}$. $\mathcal{S}$ holds public commitment of x , $\text{Hcom}_x = (\text{com}_{e_u}, x^* = e_u \cdot \mathbf{H} - x)$. $\mathcal{S}$ sends (init) to $\mathcal{F}_{C-VOLE}$. $\mathcal{S}$ receives $\Delta \in \mathbb{F}$ from $\mathcal{A}$.
2. $\mathcal{S}$ emulates $\mathcal{F}_{MPVOLE}$ and receives (extend, L, w) from $\mathcal{A}$. $\mathcal{S}$ receives $K[e_u] \in \mathbb{F}^{w \cdot 2^L}$ from $\mathcal{A}$.
3. $\mathcal{S}$ emulates $\mathcal{F}_{MPVOLE}$ and receives (extend, L', t) from $\mathcal{A}$. $\mathcal{S}$ receives $K[e_r] \in \mathbb{F}^{t \cdot 2^{L'}}$ from $\mathcal{A}$.

4. $\mathcal{S}$ emulates $\mathcal{F}_{\text{VOLUME}}$ and receives (extend, k) from $\mathcal{A}$. $\mathcal{S}$ receives $\mathbf{K}[\mathbf{r}] \in \mathbb{F}^k$ from $\mathcal{A}$.
5. $\mathcal{S}$ computes $H_3(\mathbf{M}[\text{com}_{e_u}]) = H_3(\mathbf{K}[e_u] \cdot \mathbf{A}_1 + \mathbf{K}[\mathbf{r}] \cdot \mathbf{A}_2 + \mathbf{K}[e_r] + \text{com}_{e_u} \cdot \Delta)$ and sends it to $\mathcal{A}$. $\mathcal{S}$ aborts if $\mathcal{A}$ aborts.
6. $\mathcal{S}$ computes $\mathbf{K}[\mathbf{x}] = \mathbf{K}[e_u] \cdot \mathbf{H} + \mathbf{x}^* \cdot \Delta \in \mathbb{F}^m$ and sends $(\Delta, \mathbf{K}[\mathbf{x}]) \in \mathbb{F}^{m+1}$ to $\mathcal{F}_{\text{C-VOLUME}}$. $\mathcal{S}$ outputs what $\mathcal{A}$ outputs.

We are going to show the simulated execution is indistinguishable from real-world protocol execution.

Hybrid $\mathcal{H}_0$ Same as real-world execution in $\mathcal{F}_{\text{MPVOLUME}}, \mathcal{F}_{\text{VOLUME}}$ model.

Hybrid $\mathcal{H}_1$ Same as Hybrid $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{\text{MPVOLUME}}$ and $\mathcal{F}_{\text{VOLUME}}$, and simulate messages to $\mathcal{A}$.

$\mathcal{S}$ simulates $H_3(\mathbf{M}[\text{com}_{e_u}]) = H_3(\mathbf{K}[e_u] \cdot \mathbf{A}_1 + \mathbf{K}[\mathbf{r}] \cdot \mathbf{A}_2 + \mathbf{K}[e_r] + \text{com}_{e_u} \cdot \Delta)$ and sends it to $\mathcal{A}$. In the previous hybrid, an honest $\mathbf{P}_A$ computes $\mathbf{M}[\text{com}_{e_u}] = \mathbf{K}[e_u] \cdot \mathbf{A}_1 + \mathbf{K}[\mathbf{r}] \cdot \mathbf{A}_2 + \mathbf{K}[e_r] + \text{com}_{e_u} \cdot \Delta$, which holds the property that $\mathbf{M}[e_u] = \mathbf{K}[e_u] + e_u \cdot \Delta$, $\mathbf{M}[e_r] = \mathbf{K}[e_r] + e_r \cdot \Delta$, and $\mathbf{M}[\mathbf{r}] = \mathbf{K}[\mathbf{r}] + \mathbf{r} \cdot \Delta$. Consequently, the $\mathbf{M}[\text{com}_{e_u}]$ computed by simulator equals the one computed by honest $\mathbf{P}_A$ in real world. Thus, this hybrid is identical to the previous one.

This concludes the proof. □

B.2.1. Performance of Primal-LPN based C-VOLE

Complexity Analysis. The primal-LPN-based C-VOLE protocol requires MPVOLUME instantiations of $e_u \in \mathbb{F}^{w \cdot 2^L}$ and $e_r \in \mathbb{F}^{t \cdot 2^{L'}}$, VOLE instantiation of $\mathbf{r} \in \mathbb{F}^k$, together with a $H(\mathbf{M}[\text{com}_{e_u}])$ for consistency check and a commitment with a size of $N + m$, where $N = t \cdot 2^{L'}$. According to the instantiation of MPVOLUME protocol in Section B.4, the sparse vector of length $w \cdot 2^L$ requires communication overhead of $wL\kappa$. For instantiation of VOLE protocol in Section B.5, it is dominated by $k \log |\mathbb{F}|$ for input vector with length k . Thus, the communication complexity is $(wL + tL')\kappa + (k + N + m) \log |\mathbb{F}|$ in total.

Input	Random sparse vector			Primal LPN commitment					
	$ \mathbf{x} $	$ \mathbf{e}_u $	w	L	$ \mathbf{e}_r $	t	L'	$ \mathbf{r} (k)$	$ \text{com}_{\mathbf{e}_u} $
2^{20}	$7 * 2^{20}$	224	15		1272	159	3	433	1272
2^{24}	$7 * 2^{24}$	224	19		1296	162	3	434	1296
2^{28}	$7 * 2^{28}$	224	23		1296	162	3	434	1296

Table B.1: Primal-LPN-based hybrid commitment parameters selection of $\mathbf{x}$ with different lengths. Achieves 128-bit security.

Parameter Selection of Hybrid Commitment. Same as the dual-LPN-based hybrid commitment parameter selection, we use EA code to choose the (w, L) pair of $\mathbf{e}_u \in \mathbb{F}^{w \cdot 2^L}$ for $\mathbf{x} \in \mathbb{F}^m$ with 128-bit security level. Given the (w, L) pair, we fix the structure of $\mathbf{e}_u$ and select parameters to construct a 128-bit secure commitment $\text{com}_{\mathbf{e}_u}$, where $\text{com}_{\mathbf{e}_u} = \mathbf{e}_u \cdot \mathbf{A}_1 + \mathbf{r} \cdot \mathbf{A}_2 + \mathbf{e}_r$. According to Theorem 6, we select the (k, t, L') parameters for the primal LPN commitment. For each $\mathbf{x} \in \mathbb{F}^m$ of different lengths, groups of parameters (w, L, k, t, L') are computed. Based on the complexity analysis above, we minimize the communication cost for primal-LPN-based C-VOLE protocol and present the commitment parameters that achieve 128 bit statistically binding and computational hiding for $\mathbf{x} \in \{\mathbb{F}^{2^{20}}, \mathbb{F}^{2^{24}}, \mathbb{F}^{2^{28}}\}$ in Table B.1.

Performance Evaluation. The performance of primal-LPN-based C-VOLE also consists of two parts: computing the one-time commitment of sender's input vector $\mathbf{x}$ and a online computation with any individual receiver. The time consumption and communication overhead of these two phases are shown in Figure B.2.

For one-time commitment, the sender generates a primal-LPN-based hybrid commitment of input vector $\mathbf{x} \in \mathbb{F}^m$ and publishes it to all other parties. The computation cost for the sender is 0.59 s for the computation of a vector of length $m = 2^{20}$ and 340.46 s for $m = 2^{28}$ using 4 threads, while it takes 0.30 s for $m = 2^{20}$ and 151.06 s for $m = 2^{28}$ using 16 threads. The communication overhead of publishing the primal-LPN-based hybrid commitment is 16.80 MB for $m = 2^{20}$ and 4295.0 MB for $m = 2^{28}$. For $m = 2^{20}$, the compute phase takes 0.83 s under a LAN network and 9.19 s under a WAN network with a bandwidth of 50 Mbps when using 4 threads. With 16 threads, the compute phase takes 0.61 seconds under LAN network and 8.13 seconds under WAN

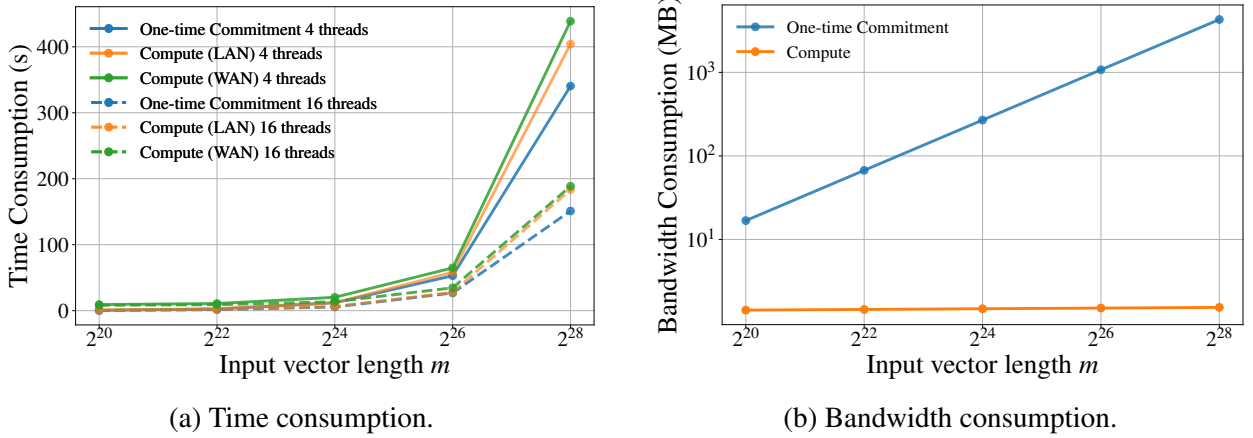


Figure B.2: Performance of the C-VOLE protocol based on the primal-LPN assumption in LAN and WAN settings.

network. For $m = 2^{28}$, the compute phase with 4 threads takes 404.11 s under a LAN network and 435.81 s under a WAN network. With 16 threads, the corresponding times are 183.47 s under a LAN network and 188.56 s under a WAN network. Regarding the online communication overhead, the cost is 1.73 Mb for $m = 2^{20}$ and 1.79 MB for $m = 2^{28}$.

Comparison with Dual-LPN-based C-VOLE. As we can see from the performance evaluation of dual-LPN-based C-VOLE and primal-LPN-based C-VOLE, both schemes perform well with the chosen hybrid commitment parameters. Considering the cumulative effect of both communication and computation overhead, the scheme based on primal-LPN shows slightly better concrete performance than the one based on dual-LPN in our experiments.

B.3. Performance of C-VOLE with Aggressive EA-code

According to Appendix B.1.2, we also evaluate the performance of our C-VOLE protocol over field $\mathbb{F}_2^{128}$ using an aggressive EA-code, which remains secure against attacks [98] applicable only to $\mathbb{F}_2$. We took $\ell = 9$ and compute the hamming weight $t = 640$. Taking dual-LPN commitment as an example, Table B.2 presents the parameters for commitments that achieve 128 bit security level.

Performance Evaluation. The performance of dual-LPN-based C-VOLE using aggressive EA-

Input	Intermediate sparse vector			Dual LPN commitment				
	$ \mathbf{x} $	$ \mathbf{e}_u $	w	L	$ \mathbf{e}_r $	t	L'	$ \text{com}_{\mathbf{e}_u} (n)$
2^{20}	$5 * 2^{20}$	640	13	1640	165	4		882
2^{24}	$5 * 2^{24}$	640	17	2688	168	4		885
2^{28}	$5 * 2^{28}$	640	21	2736	171	4		888

Table B.2: Dual-LPN-based hybrid commitment parameters selection of $\mathbf{x}$ with different lengths. Using aggressive parameters for EA-code. Commitment achieves 128-bit security level.

code also shown in two parts: computing the one-time commitment of sender's input vector $\mathbf{x}$ and a online computation with any individual receiver. The time consumption and communication overhead of these two phases are shown in Figure B.3.

For one-time commitment, the computation cost for the sender is 0.07 s for the computation of a vector of length $m = 2^{20}$ and 36.08 s for $m = 2^{28}$ using 4 threads, while it takes 0.07 s for $m = 2^{20}$ and 20.33 s for $m = 2^{28}$ using 16 threads. The communication overhead of publishing the commitment is 16.79 MB for $m = 2^{20}$ and 4294.98 MB for $m = 2^{28}$. For $m = 2^{20}$, the compute phase takes 0.36 s under a LAN network and 9.09 s under a WAN network with a bandwidth of 50 Mbps when using 4 threads. With 16 threads, the compute phase takes 0.26 seconds under LAN network and 7.84 seconds under WAN network. For $m = 2^{28}$, the compute phase with 4 threads takes 41.16 s under a LAN network and 52.79 s under a WAN network. With 16 threads, the corresponding times are 27.84 s under a LAN network and 33.89 s under a WAN network. Regarding the online communication overhead, the cost is 1.99 Mb for $m = 2^{20}$ and 2.16 MB for $m = 2^{28}$.

Comparison with Dual-LPN-based C-VOLE with conservative parameters. As we can see from the performance evaluation, the aggressive parameters result in significantly more efficient time consumption, which is roughly 10 times better for $m = 2^{28}$, although this comes at the cost of slightly worse online communication.

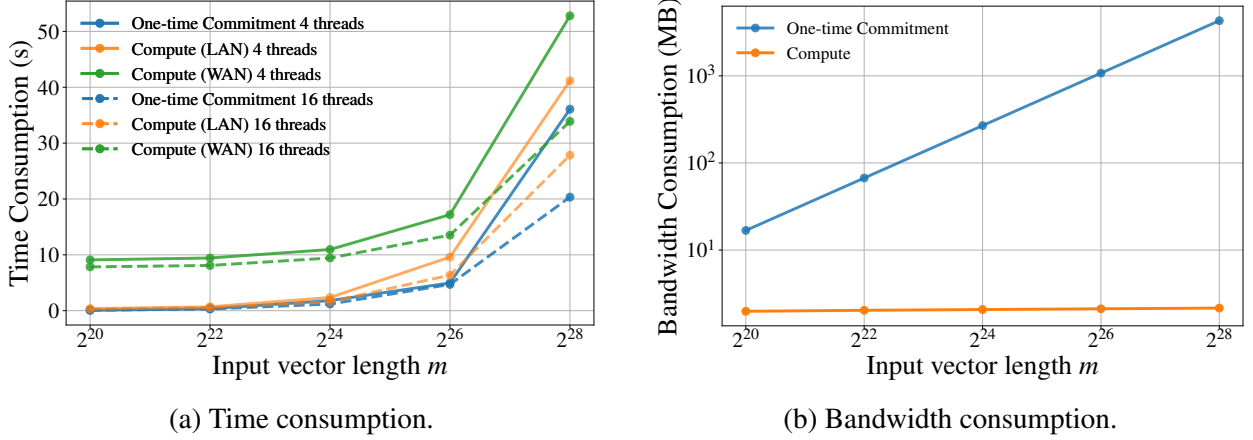


Figure B.3: Performance of the C-VOLE protocol based on the dual-LPN assumption using aggressive EA-code in LAN and WAN settings.

B.4. Protocol of MPVOLE

We detailed the GGM algorithm [100], [119], [120] and its evaluation in this section to help with the construction of Π_{MPVOLE} .

- $\text{GGM}(N, s)$: On input $N = 2^d$ for some $d \in \mathbb{N}$ and $s \in \{0, 1\}^\kappa$, this algorithm executes as follows:
 1. Define $s_0^0 := s$. For $i \in [1, d)$ and $j \in [0, 2^{i-1})$ compute $(s_{2j}^i, s_{2j+1}^i) := G(s_j^{i-1})$, where $G : \{0, 1\}^\kappa \rightarrow \{0, 1\}^{2\kappa}$ is a PRG.
 2. Compute the leaves as $(s_{2j}^d, s_{2j+1}^d) := G'(s_j^{d-1})$ for $j \in [0, 2^{d-1})$, where $G' : \{0, 1\}^\kappa \rightarrow \{0, 1\}^{2\kappa}$ is another PRG. Define a vector v as $v_h := s_{h-1}^d$ for $h \in [N]$.
 3. For $i \in [d]$, compute $S_{i,0} := \sum_{j \in [0, 2^{i-1})} s_{2j}^i$ and $S_{i,1} := \sum_{j \in [0, 2^{i-1})} s_{2j+1}^i$, where for $i \in [1, d)$, the addition is defined over $\mathbb{F}_{2^\kappa}$; for $i = d$, the addition is defined over $\mathbb{F}$.
 4. Output $(v, \{(S_{i,0}, S_{i,1})\}_{i \in [d]})$.
- $\text{GGM}'(\alpha, \{S_{i, \bar{\alpha}_i}\}_{i \in [d]})$: On input $\alpha \in [N]$ and a set $\{S_{i, \bar{\alpha}_i}\}_{i \in [d]}$ where $\bar{\alpha}_i = \alpha_i \oplus 1 \in \{0, 1\}$ for $i \in [d]$ and $\alpha = \sum_{i=1}^d \alpha_i \cdot 2^{i-1} + 1 \in [N]$, for $i \in [d]$, this algorithm does the following:
 1. Define an i -bit string $\alpha_i^* := \alpha_1 \cdots \alpha_{i-1} \bar{\alpha}_i$.

2. If $i = 1$, define $s_{\bar{\alpha}_1}^1 := S_{1, \bar{\alpha}_1}$.
3. If $i \geq 2$, for $j \in [0, 2^{i-1}), j \neq \alpha_1 \cdots \alpha_{i-1}$, compute $(s_{2j}^i, s_{2j+1}^i) := G(s_j^{i-1})$ if $i < d$ and $(s_{2j}^d, s_{2j+1}^d) := G'(s_j^{d-1})$ otherwise.
4. Compute $s_{\alpha_i^*}^i := S_{i, \bar{\alpha}_i} + \left(\sum_{j \in [0, 2^{i-1}), j \neq \alpha_1 \cdots \alpha_{i-1}} s_{2j + \bar{\alpha}_i}^i \right)$, where the addition is defined over $\mathbb{F}_{2^\kappa}$ if $i \in [1, d)$ and over $\mathbb{F}$ otherwise.
5. Define $w_h := s_{h-1}^d$ for $h \in [N], h \neq \alpha$, and output $\{w_h\}_{h \in [N] \setminus \{\alpha\}}$.

Here, we briefly introduce the construction of protocol Π_{MPVOLE} in Figure B.4, which securely instantiates the $\mathcal{F}_{\text{MPVOLE}}$ functionality in Figure 2.4. This protocol is a slightly variation of the single-point subfield-VOLE protocol from [45], where the consistency check is slightly adjust to suit the VOLE protocol. A brief description of the protocol is as follows:

The Π_{MPVOLE} generates VOLE relation of sparse multi-point vector $e_u \in \mathbb{F}^{t \cdot 2^h}, n = 2^h$, which satisfies a specific form that it is a concatenation of t single-point vectors $e_u^i, i \in [t]$, each with length n . We let α_i to represent the index of non-zero element that $\alpha_i \in [n]$ and β_i to denote the value of the non-zero element in block $e_u^i, i \in [t]$. This protocol first utilize $\mathcal{F}_{\text{VOLE}}$ over field $\mathbb{F}$ to generate VOLE of $\beta_i, i \in [t]$, as selected by P_A . Then, for each single-point block $e_u^i, i \in [t]$, this protocol follows the design of [46]. It first let P_B run GGM algorithm [121], which is an algorithm building pseudorandom function by using a PRG, to obtain pseudorandom output v_i , P_A run GGM' algorithm, to obtain $w_i, i \neq \alpha_i$. Then P_B sends d_i to help P_A recover w_i . For consistency check, this protocol first generates a random value $r_0 \leftarrow \mathbb{F}$ and its VOLE relation $M[r_0] = K[r_0] + r_0 \cdot \Delta$. With sampled $\chi_i[j], i \in [t], j \in [n]$, P_A computes linear combination of $w_i[j], i \in [t], j \in [n]$ and P_B computes linear combination of $v_i[j], i \in [t], j \in [n]$. P_A uses r_0 to mask the sum of $\beta_i \cdot \chi_i[\alpha_i], i \in [t]$ and sends the masked value to P_B . Then, the Equation B.3 holds

if both parties acts honestly.

$$\begin{aligned}
& \sum_{i \in [t]} \sum_{j=0}^{n-1} \chi_i[j] \mathbf{w}_i[j] - \mathbf{M}[r_0] \\
&= \sum_{i \in [t]} \sum_{j=0}^{n-1} \chi_i[j] \mathbf{v}_i[j] - \mathbf{K}[r_0] + \left(\sum_{i \in [t]} \beta_i \chi_i[\alpha_i] - r_0 \right) \Delta
\end{aligned} \tag{B.3}$$

We present the detailed protocol in Figure B.4. We also prove the protocol securely instantiate $\mathcal{F}_{\text{MPVOLE}}$ in $(\mathcal{F}_{\text{RVOLE}}, \mathcal{F}_{\text{VOLE}}, \mathcal{F}_{\text{OT}})$ hybrid model against malicious adversary in Theorem 16. Moreover, the construction of Π_{VOLE} is introduced in Section B.5.

Theorem 16. *Protocol Π_{MPVOLE} in Figure B.4 securely instantiated $\mathcal{F}_{\text{MPVOLE}}$ in Figure 2.4 in $(\mathcal{F}_{\text{RVOLE}}, \mathcal{F}_{\text{VOLE}}, \mathcal{F}_{\text{OT}})$ -hybrid model against malicious adversaries.*

Proof. Let $\mathcal{A}$ be a PPT adversary that corrupt P_A or P_B . We construct a PPT simulator $\mathcal{S}$ with access to $\mathcal{F}_{\text{MPVOLE}}$ and simulates the adversary's view. We will prove that the joint distribution over the output of $\mathcal{A}$ and the honest party in the real world is indistinguishable from the joint distribution over the output of $\mathcal{S}$ and the honest party in the ideal world.

Corrupted P_A . Let $\mathcal{S}$ access $\mathcal{F}_{\text{MPVOLE}}$ as an honest P_A and interact with $\mathcal{A}$ as an honest P_B . $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

For Initialize phase, $\mathcal{S}$ simulates as follows:

1. $\mathcal{S}$ emulates $\mathcal{F}_{\text{RVOLE}}$ and receives (init) from $\mathcal{A}$. $\mathcal{S}$ sends (init) to $\mathcal{F}_{\text{MPVOLE}}$.

For Extend phase, $\mathcal{S}$ simulates as follows:

2. $\mathcal{S}$ emulates $\mathcal{F}_{\text{RVOLE}}$ and receives (extend, 1) from $\mathcal{A}$. $\mathcal{S}$ receives $r_0 \in \mathbb{F}$, $\mathbf{M}[r_0] \in \mathbb{F}$ from $\mathcal{A}$.
3. $\mathcal{S}$ emulates $\mathcal{F}_{\text{VOLE}}$ and receives (extend, β , t) from $\mathcal{A}$. $\mathcal{S}$ receives $\mathbf{M}[\beta] \in \mathbb{F}^t$ from $\mathcal{A}$.
4. For each iteration $i \in [t]$, $\mathcal{S}$ emulates $\mathcal{F}_{\text{OT}}$ and receives $\bar{a}_i[k], k \in [h]$. $\mathcal{S}$ reconstructs $\alpha_i, i \in [t]$. $\mathcal{S}$ samples $K_{i, \bar{a}_i[k]}^k \leftarrow \mathbb{F}, k \in [h]$ and sends them to $\mathcal{A}$. $\mathcal{S}$ reconstructs e_u and sends (extend, e_u, h, t) to $\mathcal{F}_{\text{MPVOLE}}$. $\mathcal{S}$ also runs $\{v_i[j]\}_{j \neq \alpha_i} := \text{GGM}'(\alpha_i, \{K_{i, \bar{a}_i[k]}^k\}_{k \in [h]})$.

Protocol Π_{MPVole}

Parameters: GGM: $(\mathbf{v}, \{S_{i,0}, S_{i,1}\}_{i \in [\log N]}) \leftarrow \text{GGM}(N, s), s \leftarrow \{0, 1\}^\kappa, \mathbf{v} \in \mathbb{F}^N, S_{i,0} \in \{0, 1\}^\kappa, S_{i,1} \in \{0, 1\}^\kappa$; GGM': $\{w_h\}_{h \in [N]/\{\alpha\}} \leftarrow \text{GGM}'(\alpha, \{S_{i,\bar{\alpha}_i}\}_{i \in [\log N]}), w_h \in \{0, 1\}^\kappa$.

Inputs: For each extend sparse execution, P_A holds regular sparse multi-point vector $e_u \in \mathbb{F}^{t \cdot 2^h}, n = 2^h$. P_A partition $e_u \in \mathbb{F}^{t \cdot 2^h}$ into t blocks, that for each block $e_u^i \in \mathbb{F}^n, i \in [t]$, it has one non-zero element $\beta_i \in \mathbb{F}$ at position $\alpha_i \in [n]$. P_A represents α_i as bit vectors, namely $\mathbf{a}_i \in \{0, 1\}^h$. Both P_A and P_B have integers $t \in \mathbb{N}$ and $n = 2^h$ for some $h \in \mathbb{N}$.

Initialize:

1. Both P_A and P_B sends (init) to $\mathcal{F}_{\text{RVOLE}}$. P_B receives $\Delta \in \mathbb{F}$ from $\mathcal{F}_{\text{RVOLE}}$.

Extend:

2. Both P_A and P_B call $\mathcal{F}_{\text{RVOLE}}$ with (extend, 1). The functionality sends $r_0 \in \mathbb{F}, M[r_0] \in \mathbb{F}$ to P_A , and $K[r_0] \in \mathbb{F}$ to P_B , such that $M[r_0] = K[r_0] + r_0 \cdot \Delta$.
3. P_A call $\mathcal{F}_{\text{VOLE}}$ with (extend, β , t), where β is a concatenation of β_i s, and P_B call $\mathcal{F}_{\text{VOLE}}$ with (extend, t). The functionality sends $M[\beta] \in \mathbb{F}^t$ to P_A , and $K[\beta] \in \mathbb{F}^t$ to P_B , such that $M[\beta_i] = K[\beta_i] + \beta_i \Delta$ for each $i \in [t]$.
4. For each iteration $i \in [t]$, P_B samples seed $l_i \leftarrow \{0, 1\}^\kappa$, runs $\text{GGM}(1^n, l_i)$ to obtain $(\{v_i[j]\}_{j \in [n]}, \{K_{i,0}^k, K_{i,1}^k\}_{k \in [h]})$, and sets $\mathbf{v}_i[j] := v_i[j]$ for $j \in [n]$. For $k \in [h]$, P_B call $\mathcal{F}_{\text{OT}}$ with input $(K_{i,0}^k, K_{i,1}^k)$; P_A call $\mathcal{F}_{\text{OT}}$ with input $\bar{a}_i[k]$ and get $K_{i,\bar{a}_i[k]}^k$. Then P_A runs $\{v_i[j]\}_{j \neq \alpha_i} := \text{GGM}'(\alpha_i, \{K_{i,\bar{a}_i[k]}^k\}_{k \in [h]})$.
5. For each iteration $i \in [t]$, P_B sends $d_i := K[\beta_i] - \sum_{j \in [n]} \mathbf{v}_i[j] \in \mathbb{F}$ to P_A . Then, P_A defines $\mathbf{w}_i \in \mathbb{F}^n$ as the vector with $\mathbf{w}_i[j] := v_i[j]$ for $j \neq \alpha_i$ and $\mathbf{w}_i[\alpha_i] := M[\beta_i] - (d_i + \sum_{j \neq \alpha_i} \mathbf{w}_i[j])$. Note that $\mathbf{w}_i = \mathbf{v}_i + \Delta \cdot e_u^i$.
6. P_A and P_B check **GGM-consistency** as follows.
 - (a) P_A samples $\chi_i[j] \leftarrow \mathbb{F}$ for $i \in [t], j \in [n]$. P_A then computes $x^* := \sum_{i \in [t]} \beta_i \cdot \chi_i[\alpha_i] - r_0 \in \mathbb{F}$ and sends $(\{\chi_i[j]\}_{i \in [t], j \in [n]}, x^*)$ to P_B , who computes $y := K[r_0] - \Delta \cdot x^*$.
 - (b) P_A computes $V_A := \sum_{i \in [t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot \mathbf{w}_i[j] - M[r_0] \in \mathbb{F}$, while P_B computes $V_B := \sum_{i \in [t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot \mathbf{v}_i[j] - y \in \mathbb{F}$. Then P_B sends $H(V_B)$ to P_A . P_A aborts if $H(V_A) \neq H(V_B)$.
7. P_A outputs $M[e_u] = \mathbf{w}_1 || \dots || \mathbf{w}_t \in \mathbb{F}^{tn}$ and P_B outputs $K[e_u] = \mathbf{v}_1 || \dots || \mathbf{v}_t \in \mathbb{F}^{tn}$.

Figure B.4: Multi-point VOLE protocol.

5. For each iteration $i \in [t]$, $\mathcal{S}$ samples $d_i \leftarrow \mathbb{F}$ and sends it to $\mathcal{A}$. $\mathcal{S}$ computes $\mathbf{w}_i[j] := v_i[j]$ for $j \neq \alpha_i$ and $\mathbf{w}_i[\alpha_i] := M[\beta_i] - (d_i + \sum_{j \neq \alpha_i} \mathbf{w}_i[j])$.
6. $\mathcal{S}$ receives $(\{\chi_i[j]\}_{i \in [t], j \in [n]}, x^*)$ from $\mathcal{A}$. $\mathcal{S}$ checks whether $x^* = \sum_{i \in [t]} \beta_i \cdot \chi_i[\alpha_i] - r_0$. If it is, $\mathcal{S}$ computes $V_B = \sum_{i \in [t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot \mathbf{w}_i[j] - M[r_0]$ and sends $H(V_B)$ to $\mathcal{A}$. Otherwise, $\mathcal{S}$ sends $H(V_B) \leftarrow \mathbb{F}$ to $\mathcal{A}$.
7. $\mathcal{S}$ constructs $M[e_u] = \mathbf{w}_1 || \dots || \mathbf{w}_t$ to $\mathcal{F}_{\text{MPVole}}$. $\mathcal{S}$ outputs what $\mathcal{A}$ outputs.

We are going to show the simulated execution is indistinguishable from the real protocol execution.

Hybrid $\mathcal{H}_0$ Same as real-world execution in $(\mathcal{F}_{\text{RVOLE}}, \mathcal{F}_{\text{VOLE}}, \mathcal{F}_{\text{OT}})$ model.

Hybrid $\mathcal{H}_1$ Same as Hybrid $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{\text{RVOLE}}, \mathcal{F}_{\text{VOLE}}, \mathcal{F}_{\text{OT}}$ and simulate messages to $\mathcal{A}$.

For step (4), $\mathcal{S}$ emulates $\mathcal{F}_{\text{OT}}$ and sends $K_{i, \bar{a}_i[k]}^k \leftarrow \mathbb{F}, k \in [h]$ to $\mathcal{A}$. In Hybrid $\mathcal{H}_0$, an honest P_B computes $K_{i, \bar{a}_i[k]}^k, k \in [h]$ according to GGM algorithm with random seed $l_i \leftarrow \{0, 1\}^\kappa$. By leveraging PRG over random seed $l_i, K_{i, \bar{a}_i[k]}^k, k \in [h]$ are indistinguishable from random value over $\mathbb{F}$.

For step (5), $\mathcal{S}$ samples $d_i \leftarrow \mathbb{F}, i \in [t]$ to $\mathcal{A}$. In Hybrid $\mathcal{H}_0$, an honest P_B computes $d_i = \mathsf{K}[\beta_i] - \sum_{j \in [n]} \mathbf{v}_i[j]$ to $\mathcal{A}$. Since $\mathsf{K}[\beta_i]$ is unknown to $\mathcal{A}$, d_i is uniform distributed to $\mathcal{A}$.

For step (6), $\mathcal{S}$ receives $(\{\chi_i[j]\}_{i \in [t], j \in [n]}, x^*)$ from $\mathcal{A}$ and sends $H(V_B)$ to $\mathcal{A}$. $\mathcal{S}$ sets $H(V_B) = H(\sum_{i \in [t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot \mathbf{w}_i[j] - \mathsf{M}[r_0])$ if $x^* = \sum_{i \in [t]} \beta_i \cdot \chi_i[\alpha_i] - r_0$. $\mathcal{S}$ sets $H(V_B) \leftarrow \mathbb{F}$ if $x^* \neq \sum_{i \in [t]} \beta_i \cdot \chi_i[\alpha_i] - r_0$. In Hybrid $\mathcal{H}_0$, an honest P_B computes $V_B = \sum_{i \in [t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot \mathbf{v}_i[j] - y$, where $y = \mathsf{K}[r_0] - \Delta \cdot x^*$, and sends $H(V_B)$ to $\mathcal{A}$. If $x^* = \sum_{i \in [t]} \beta_i \cdot \chi_i[\alpha_i] - r_0 + \text{rand}, \text{rand} \in \mathbb{F}$, $y = \mathsf{M}[r_0] - \sum_{i \in [t]} \beta_i \cdot \chi_i[\alpha_i] \cdot \Delta - \text{rand} \cdot \Delta$. We have the following equation holds.

$$\begin{aligned}
V_B &= \sum_{i \in [t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot \mathbf{v}_i[j] - y \\
&= \sum_{i \in [t]} \sum_{j=0}^{n-1} \chi_i[j] \mathbf{v}_i[j] - \mathsf{M}[r_0] + \sum_{i \in [t]} \beta_i \chi_i[\alpha_i] \Delta + \text{rand} \Delta \\
&= \sum_{i \in [t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot \mathbf{w}_i[j] - \sum_{i \in [t]} \beta_i \cdot \chi_i[\alpha_i] \cdot \Delta - \mathsf{M}[r_0] \\
&\quad + \sum_{i \in [t]} \beta_i \cdot \chi_i[\alpha_i] \cdot \Delta + \text{rand} \cdot \Delta \\
&= \sum_{i \in [t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot \mathbf{w}_i[j] - \mathsf{M}[r_0] + \text{rand} \cdot \Delta \\
&= V_A + \text{rand} \cdot \Delta
\end{aligned}$$

If $rand = 0$, we have $V_A = V_B$. Thus, the simulation is identical to the real world protocol execution. Otherwise, $\mathcal{A}$ will receives $H(V_B) = H(V_A + rand \cdot \Delta)$, which is indistinguishable from a random value in the hybrid of random oracle model.

Thus, this hybrid is identical to the previous one.

Corrupted P_B . Let $\mathcal{S}$ access $\mathcal{F}_{MPVOLE}$ as an honest P_B and interact with $\mathcal{A}$ as an honest P_A . $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$. For Initialize phase, $\mathcal{S}$ simulates as follows:

1. $\mathcal{S}$ emulates $\mathcal{F}_{RVOLE}$ and receives (init) from $\mathcal{A}$. $\mathcal{S}$ receives Δ from $\mathcal{A}$.

For Extend phase, $\mathcal{S}$ simulates as follows:

2. $\mathcal{S}$ emulates $\mathcal{F}_{RVOLE}$ and receives (extend, 1) from $\mathcal{A}$. $\mathcal{S}$ receives $K[r_0] \leftarrow \mathbb{F}$ from $\mathcal{A}$.
3. $\mathcal{S}$ emulates $\mathcal{F}_{VOLE}$ and receives (extend, t) from $\mathcal{A}$. $\mathcal{S}$ receives $K[\beta] \leftarrow \mathbb{F}^t$ from $\mathcal{A}$. $\mathcal{S}$ samples $\beta_i, i \in [t]$ and computes $M[\beta_i] = K[\beta_i] + \beta_i \cdot \Delta, i \in [t]$.
4. For each iteration $i \in [t]$, $\mathcal{S}$ emulates $\mathcal{F}_{OT}$ and receives $(K_{i,0}^k, K_{i,1}^k), k \in [h]$.
5. For each iteration $i \in [t]$, $\mathcal{S}$ receives $d_i \leftarrow \mathbb{F}$ from $\mathcal{A}$. For each $\alpha_i \in [n]$, $\mathcal{S}$ computes $w_i^{\alpha_i}$ as follows:

(a) Computes $\{v_i^{\alpha_i}[j]\}_{j \neq \alpha_i} := \text{GGM}'(\alpha_i, \{K_{i,\bar{\alpha}_i[k]}^k\}_{k \in [h]})$ and sets $w_i^{\alpha_i}[j] = v_i^{\alpha_i}[j]$ for $j \neq \alpha_i$

(b) Sets $w_i^{\alpha_i}[\alpha_i] := M[\beta_i] - \left(d_i + \sum_{j \neq \alpha_i} w_i^{\alpha_i}[j]\right)$.

6. (a) $\mathcal{S}$ samples $(\{\chi_i[j] \leftarrow \mathbb{F}\}_{i \in [t], j \in [n]}, x^* \leftarrow \mathbb{F})$ to $\mathcal{A}$.

(b) $\mathcal{S}$ receives $H(V_B)$ from $\mathcal{A}$. $\mathcal{S}$ is able to extract V_B as $\mathcal{S}$ emulate random oracle H for $\mathcal{A}$.

For each $\{\alpha_i \in [n]\}_{i \in [t]}$, denoted as $\alpha \in \mathcal{N}^t, \mathcal{N} := [n]$, $\mathcal{S}$ computes V_A^α as follows: $V_A^\alpha = \sum_{i \in [t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot w_i^{\alpha_i}[j] - M[r_0]$. Define $I := \{\alpha \in \mathcal{N}^t | V_A^\alpha = V_B\}$. We also define $I = I_1 \times \cdots \times I_t$, where $I_i := \{\alpha_i \in \mathcal{N} | V_A^\alpha = V_B\}$. $\mathcal{S}$ aborts if $I = \{\}$.

$\mathcal{S}$ chooses an arbitrary $\alpha \in I$ and computes vector v_i for each $i \in [t]$ as follows:

(a) Sets $\mathbf{v}_i[j] := \mathbf{w}_i^{\alpha_i}[j]$ for $j \neq \alpha_i$

(b) Sets $\mathbf{v}_i[\alpha_i] := K[\beta_i] - d_i - \sum_{j \neq \alpha_i} \mathbf{v}_i[j]$

$\mathcal{S}$ send $\{I_1, \dots, I_t\}$ to $\mathcal{F}_{\text{MPVOLE}}$. If it receives abort, $\mathcal{S}$ aborts.

7. $\mathcal{S}$ constructs $K[e_u] = \mathbf{v}_1 || \dots || \mathbf{v}_t$ to $\mathcal{F}_{\text{MPVOLE}}$. $\mathcal{S}$ outputs what $\mathcal{A}$ outputs.

We are going to show the simulated execution is indistinguishable from the real protocol execution.

Hybrid $\mathcal{H}_0$ Same as real-world execution in $(\mathcal{F}_{\text{RVOLE}}, \mathcal{F}_{\text{VOLE}}, \mathcal{F}_{\text{OT}})$ model.

Hybrid $\mathcal{H}_1$ Same as Hybrid $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{\text{RVOLE}}, \mathcal{F}_{\text{VOLE}}, \mathcal{F}_{\text{OT}}$ and simulate messages to $\mathcal{A}$.

For step (6), $\mathcal{S}$ samples $(\{\chi_i[j] \leftarrow \mathbb{F}\}_{i \in [t], j \in [n]}, x^* \leftarrow \mathbb{F})$ to $\mathcal{A}$. In Hybrid $\mathcal{H}_0$, an honest P_A samples $\{\chi_i[j] \leftarrow \mathbb{F}\}_{i \in [t], j \in [n]}$, and computes $x^* = \sum_{i \in [t]} \beta_i \cdot \chi_i[\alpha_i] - r_0$. Since r_0 is output by $\mathcal{F}_{\text{VOLE}}$ to P_A and is uniformly distributed against $\mathcal{A}$, x^* is uniformly distributed against $\mathcal{A}$, which is indistinguishable from this hybrid. Thus, this hybrid is identical to the previous one.

Hybrid $\mathcal{H}_2$ Same as Hybrid $\mathcal{H}_1$ except $\mathcal{S}$ aborts at step (6) if $I = \{\}$ or received abort from $\mathcal{F}_{\text{MPVOLE}}$.

For step (6), $\mathcal{S}$ extracts set I and send it to $\mathcal{F}_{\text{MPVOLE}}$, which is equivalent to $\mathcal{A}$ samples selective failure attack on P_A 's chosen non-zero index $\alpha_i, i \in [t]$. If $\mathcal{S}$ receives abort from $\mathcal{F}_{\text{MPVOLE}}$, there exist $i, i \in [t]$, $\alpha_i \notin I_i$, that is equivalent to $\alpha \notin I$. In the real world protocol execution, if $V_B \neq V_A^\alpha$, P_A aborts. This is equivalent to $\alpha \notin I$. Thus, $\mathcal{F}_{\text{MPVOLE}}$ aborts if and only if the real world protocol execution aborts. This hybrid is identical to the previous one.

Then, we will prove the outputs of $\mathcal{A}$ and honest party in the real world is indistinguishable from the outputs of $\mathcal{S}$ and honest party in the ideal world.

We will prove that except with probability $1/|\mathbb{F}|$, all choices of $\alpha_i \in I_i$ in the above simulation lead to the same vector $\mathbf{v}_i$.

Since $V_A^\alpha = V_A^{\alpha'} = V_B$, we have:

$$V_A^\alpha = V_A^{\alpha'}$$

$$\begin{aligned}
& \sum_{i \in [t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot \mathbf{w}_i^{\alpha_i}[j] - \mathbf{M}[r_0] = \sum_{i \in [t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot \mathbf{w}_i^{\alpha'_i}[j] - \mathbf{M}[r_0] \\
& \sum_{i \in [t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot \mathbf{w}_i^{\alpha_i}[j] - y - \sum_{i \in [t]} \beta_i \chi_i[\alpha_i] \Delta = \\
& \sum_{i \in [t]} \sum_{j=0}^{n-1} \chi_i[j] \cdot \mathbf{w}_i^{\alpha'_i}[j] - y - \sum_{i \in [t]} \beta_i \chi_i[\alpha'_i] \Delta \\
& \sum_{i \in [t]} \left(\sum_{j \neq \alpha_i, \alpha'_i} \chi_i[j] (\mathbf{w}_i^{\alpha_i}[j] - \mathbf{w}_i^{\alpha'_i}[j]) + \chi_i[\alpha_i] (\mathbf{w}_i^{\alpha_i}[\alpha_i] \right. \\
& \quad \left. - \mathbf{w}_i^{\alpha'_i}[\alpha_i] - \beta_i \Delta) + \chi_i[\alpha'_i] (\mathbf{w}_i^{\alpha_i}[\alpha'_i] - \mathbf{w}_i^{\alpha'_i}[\alpha'_i] + \beta_i \Delta) \right) = 0
\end{aligned}$$

Since $\{\chi_i[j]\}_{i \in [t], j \in [n]}$ are uniformly distributed and independent from $\mathbf{w}_i^{\alpha_i}$, $\mathbf{w}_i^{\alpha'_i}$, and Δ , we have except with probability $1/|\mathbb{F}|$,

$$\mathbf{w}_i^{\alpha_i}[j] = \mathbf{w}_i^{\alpha'_i}[j], j \neq \alpha_i, \alpha'_i \quad (\text{B.4})$$

$$\mathbf{w}_i^{\alpha_i}[\alpha_i] - \mathbf{w}_i^{\alpha'_i}[\alpha_i] = \mathbf{w}_i^{\alpha'_i}[\alpha'_i] - \mathbf{w}_i^{\alpha_i}[\alpha'_i] = \Delta \beta_i \quad (\text{B.5})$$

Thus, we are able to obtain for any $\alpha_i, \alpha'_i \in I_i$, $\mathbf{v}_i^{\alpha_i}[j] = \mathbf{v}_i^{\alpha'_i}[j]$, $j \neq \{\alpha_i, \alpha'_i\}$ from Equation B.4. According to $\mathbf{w}_i^{\alpha_i}[\alpha_i] = \mathbf{v}_i^{\alpha_i}[\alpha_i] + \Delta \beta_i$ and Equation B.5, we are able to get $\mathbf{v}_i^{\alpha_i}[\alpha_i] = \mathbf{w}_i^{\alpha'_i}[\alpha_i] = \mathbf{v}_i^{\alpha'_i}[\alpha_i]$ and similarly $\mathbf{v}_i^{\alpha'_i}[\alpha'_i] = \mathbf{w}_i^{\alpha_i}[\alpha'_i] = \mathbf{v}_i^{\alpha_i}[\alpha'_i]$. $\square$

B.5. Protocol of VOLE

We present a simple VOLE protocol constructed from $\mathcal{F}_{\text{RVOLE}}$. In this protocol, $\mathcal{P}_A$ holds the designated input vector $\mathbf{b} \in \mathbb{F}^m$. After calling $\mathcal{F}_{\text{RVOLE}}$ on both sides, $\mathcal{P}_A$ obtains random $(\mathbf{r}, \mathbf{M}[\mathbf{r}]) \in \mathbb{F}^{2m}$ and $\mathcal{P}_B$ obtains $(\Delta, \mathbf{K}[\mathbf{r}]) \in \mathbb{F}^{m+1}$, such that $\mathbf{M}[\mathbf{r}] = \mathbf{K}[\mathbf{r}] + \mathbf{r} \cdot \Delta$. Then, $\mathcal{P}_A$ sends the difference $\mathbf{f}$ between $\mathbf{r}$ and $\mathbf{b}$. Consequently, both parties are able to obtain the VOLE correlation of the designated vector $\mathbf{b}$, where $\mathbf{M}[\mathbf{b}] = \mathbf{M}[\mathbf{r}]$ and $\mathbf{K}[\mathbf{b}] = \mathbf{K}[\mathbf{r}] + \mathbf{f} \cdot \Delta$.

This protocol is able to securely instantiate $\mathcal{F}_{\text{VOLE}}$ in the $\mathcal{F}_{\text{RVOLE}}$ hybrid model. A simple proof sketch is provided here. For corrupted $\mathcal{P}_A$, a simulator $\mathcal{S}$ emulates $\mathcal{F}_{\text{VOLE}}$ and receives $\mathbf{r} \in \mathbb{F}^m$, $\mathbf{M}[\mathbf{r}] \in \mathbb{F}^m$ from $\mathcal{A}$. $\mathcal{S}$ receives $\mathbf{f} \in \mathbb{F}^m$ from $\mathcal{A}$. It is able to recover $\mathbf{b} = \mathbf{r} - \mathbf{f}$, and

$M[b] = M[r]$, and sends $M[b]$ to $\mathcal{F}_{\text{VOLE}}$. The simulator does not need to simulate any message to the adversary. For corrupted P_B , $\mathcal{S}$ receives $K[r] \in \mathbb{F}^m$ from $\mathcal{A}$ and samples $f \leftarrow \mathbb{F}^m$ to $\mathcal{A}$, which is indistinguishable from the f in real protocol. Then, $\mathcal{S}$ is able to compute $K[b] = K[r] + f\Delta$ to $\mathcal{F}_{\text{VOLE}}$. The above simulation could simulate the view of adversary in real protocol. Also, $\mathcal{S}$ output what $\mathcal{A}$ outputs could ensure the same output distribution of the real protocol.

B.6. Proof of Theorem 9

Proof. Let $\mathcal{A}$ be a PPT adversary that corrupt P_1 or P_j . We construct a PPT simulator $\mathcal{S}$ with access to $\mathcal{F}_{\text{PSI}}$ and simulates the adversary's view. We will prove that the joint distribution over the output of $\mathcal{A}$ and the honest party in the real world is indistinguishable from the joint distribution over the output of $\mathcal{S}$ and the honest party in the ideal world.

Corrupted P_1 . Let $\mathcal{S}$ access $\mathcal{F}_{\text{PSI}}$ as an honest P_1 and interact with $\mathcal{A}$ as an honest P_2 . $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

- (1) $\mathcal{S}$ emulates random oracle H_1 and receives elements $x_i, i \in [n]$ from $\mathcal{A}$. $\mathcal{S}$ samples $H_1(x_i) \leftarrow \mathbb{F}, i \in [n]$ and sends them to $\mathcal{A}$.
- (2) $\mathcal{S}$ receives $\text{Hcom}_p \in \mathbb{F}^N$ from $\mathcal{A}$.
- (3) $\mathcal{S}$ samples $c^s \leftarrow \mathbb{F}$ and sends it to $\mathcal{A}$.
- (4) $\mathcal{S}$ receives $w^r \in \mathbb{F}, r \in \{0, 1\}^\kappa$ from $\mathcal{A}$.
- (5) $\mathcal{S}$ emulates $\mathcal{F}_{\text{C-VOLE}}$ and receives $(\text{init}, p, \text{Hdecom}_p)$, where $p \in \mathbb{F}^m, \text{Hdecom}_p \in \mathbb{F}^{m'}$ from $\mathcal{A}$. $\mathcal{S}$ checks whether $(\text{Hcom}_p, \text{Hdecom}_p)$ opens to p . If not, $\mathcal{S}$ sends **abort** to $\mathcal{A}$. Otherwise, $\mathcal{S}$ waits to receive $M[p] \leftarrow \mathbb{F}^m$ from $\mathcal{A}$. $\mathcal{S}$ checks whether $\text{Decode}(p, x_i, r) = H(x_i)$. If it is, $\mathcal{S}$ inserts x_i to set X . $\mathcal{S}$ sends (init, X) to $\mathcal{F}_{\text{PSI}}$ and waits to receive (setid, Z) .
- (6) $\mathcal{S}$ samples $w^s \leftarrow \mathbb{F}$ and programs random oracle H_1 that $c^s = H_1(w^s)$. $\mathcal{S}$ sends w^s to $\mathcal{A}$. $\mathcal{S}$ computes $w = w^s + w^r$.

- (7) $\mathcal{S}$ samples $ey_i \leftarrow \{0, 1\}^{out}$ for each $y_i \in Y$ and inserts ey_i to set EY . $\mathcal{S}$ sends EY to $\mathcal{A}$.
- (8) $\mathcal{S}$ emulates random oracle H_2 and receives query (q_i, x_i) from $\mathcal{A}$. $\mathcal{S}$ first checks whether the query is fresh.
- (a) If it is, $\mathcal{S}$ checks whether $x_i \in Z$ and $q_i = \text{Decode}(\mathbf{M}[\mathbf{p}], x_i, r) + w$. If it is, $\mathcal{S}$ samples ex_i from EY/EX and sends it to $\mathcal{A}$. $\mathcal{S}$ records the triple (q_i, x_i, ex_i) and inserts ex_i to EX . If not, $\mathcal{S}$ samples $ex_i \leftarrow \{0, 1\}^{out}/(EY \cup EX)$ to $\mathcal{A}$, records the triple, and inserts ex_i to EX .
 - (b) If not, $\mathcal{S}$ checks records and sends corresponding ex_i to $\mathcal{A}$.

$\mathcal{S}$ aborts if $\mathcal{A}$ aborts and outputs what $\mathcal{A}$ outputs.

While $\mathcal{S}$ access $\mathcal{F}_{\text{PSI}}$ as an honest P_1 and interact with $\mathcal{A}$ as an honest $P_j, j \in [3, n]$. $\mathcal{S}$ simulates the **Extend** phase. The simulation is same as above except step (5):

- (5) $\mathcal{S}$ emulates $\mathcal{F}_{\text{C-VOLE}}$ and receives $(\text{init}, \mathbf{p}, \text{Hdecom}_{\mathbf{p}})$, where $\mathbf{p} \in \mathbb{F}^m, \text{Hdecom}_{\mathbf{p}} \in \mathbb{F}^{m'}$ from $\mathcal{A}$. $\mathcal{S}$ checks whether $(\text{Hcom}_{\mathbf{p}}, \text{Hdecom}_{\mathbf{p}})$ opens to $\mathbf{p}$. If not, $\mathcal{S}$ sends **abort** to $\mathcal{A}$. Otherwise, $\mathcal{S}$ waits to receive $\mathbf{M}[\mathbf{p}] \leftarrow \mathbb{F}^m$ from $\mathcal{A}$. $\mathcal{S}$ sends $(\text{init}, \text{setid})$ to $\mathcal{F}_{\text{PSI}}$ and waits to receive Z .

We are going to show the simulated execution is indistinguishable from real-world protocol execution.

Hybrid $\mathcal{H}_0$ Same as real-world execution in $(\mathcal{F}_{\text{C-VOLE}}, H_1, H_2)$ model.

Hybrid $\mathcal{H}_1$ Same as Hybrid $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{\text{C-VOLE}}$, random oracle H_1, H_2 , and simulates the messages to $\mathcal{A}$ as follows:

For step (1), $\mathcal{S}$ emulates H_1 and samples $H_1(x_i) \leftarrow \mathbb{F}, i \in [n]$ to $\mathcal{A}$. In Hybrid $\mathcal{H}_0$, $H_1(x_i) \leftarrow \mathbb{F}, i \in [n]$ are generated and outputted by a random oracle, which is uniform distributed and indistinguishable from this hybrid.

For step (3), $\mathcal{S}$ samples $c^s \leftarrow \mathbb{F}$ to $\mathcal{A}$. In hybrid $\mathcal{H}_0$, Random oracle H_1 outputs c^s , which is uniformly distributed over $\mathbb{F}$ and indistinguishable from this hybrid.

For step (6), $\mathcal{S}$ samples $w^s \leftarrow \mathbb{F}$ to $\mathcal{A}$. $\mathcal{S}$ also programs random oracle H_1 that $c^s = H_1(w_s)$. In hybrid $\mathcal{H}_0$, an honest P_2 samples $w^s \leftarrow \mathbb{F}$ and computes $c^s = H_1(w_s)$. The distribution of w^s is indistinguishable from this hybrid.

For step (7), $\mathcal{S}$ programs EY to size $|Y|$ with random elements in $\{0, 1\}^{out}$. In hybrid H_0 , an honest P_2 computes ey_i from random oracle for each element $y_i \in Y_2$. Thus, EY is indistinguishable from this hybrid.

For step (8), $\mathcal{S}$ emulates H_2 and programs H_2 that for $x_i \in Z$, $q_i = \text{Decode}(\mathbf{M}[\mathbf{p}], x_i, r) + w$, it sends $ex_i \in EY$ to $\mathcal{A}$. Thus, the output of $\mathcal{A}$ is same as interacting with an honest P_2 .

Thus, this hybrid is identical to the previous one.

Corrupted P_j . Let $\mathcal{S}$ access $\mathcal{F}_{\text{PSI}}$ as an honest P_j and interact with $\mathcal{A}$ as an honest P_1 . $\mathcal{S}$ passes all communication between $\mathcal{A}$ and environment $\mathcal{Z}$.

(1-2) $\mathcal{S}$ samples $\text{Hcom}_p \leftarrow \mathbb{F}^N$. $\mathcal{S}$ publishes Hcom_p .

(3) $\mathcal{S}$ emulates H_1 and receives w^s from $\mathcal{A}$. $\mathcal{S}$ samples $c^s \leftarrow \mathbb{F}$ to $\mathcal{A}$. $\mathcal{S}$ receives $c'^s \leftarrow \mathbb{F}$ from $\mathcal{A}$. If $c'^s \neq c^s$, $\mathcal{S}$ aborts in step (6).

(4) $\mathcal{S}$ samples $w^r \leftarrow \mathbb{F}$, $r \leftarrow \{0, 1\}^\kappa$ and sends them to $\mathcal{A}$.

(5) $\mathcal{S}$ emulates $\mathcal{F}_{\text{C-VOLE}}$ and receives (init) from $\mathcal{A}$. $\mathcal{S}$ waits to receive $(\Delta, \mathbf{K}[\mathbf{p}]) \leftarrow \mathbb{F}^{m+1}$ from $\mathcal{A}$.

(6) $\mathcal{S}$ receives w'^s from $\mathcal{A}$ and checks whether $w'^s = w^s$. If not, $\mathcal{S}$ aborts.

(7) $\mathcal{S}$ emulates H_1 , H_2 , and receives query as follows:

(a) $\mathcal{S}$ receives y_i from $\mathcal{A}$. If the query is fresh, $\mathcal{S}$ samples $H_1(y_i) \leftarrow \mathbb{F}$ to $\mathcal{A}$ and stores the pair $(y_i, H_1(y_i))$. If not, $\mathcal{S}$ checks the records and sends corresponding $H_1(y_i)$ to $\mathcal{A}$.

(b) $\mathcal{S}$ receives query (q_i, y_i) from $\mathcal{A}$. If the query is fresh, $\mathcal{S}$ samples $ey_i \leftarrow \{0, 1\}^{out}$ to $\mathcal{A}$ and records triple (y_i, q_i, ey_i) . If not, $\mathcal{S}$ checks the records and sends corresponding ey_i to $\mathcal{A}$.

$\mathcal{S}$ receives EY from $\mathcal{A}$. For each $ey_i \in EY$, $\mathcal{S}$ checks whether it is recorded and $q_i = \text{Decode}(\mathbf{K}[\mathbf{p}], y_i, r) + \Delta H_1(y_i) + w$. If it is, $\mathcal{S}$ inserts y_i to set Y and sends (intersect, Y)

to $\mathcal{F}_{\text{PSI}}$. $\mathcal{S}$ receives setid from $\mathcal{F}_{\text{PSI}}$.

While $\mathcal{S}$ access $\mathcal{F}_{\text{PSI}}$ as an honest P_j and interact with $\mathcal{A}$ as an honest P_1 . $\mathcal{S}$ simulates the **Extend** phase. The simulation is same as above except: $\mathcal{S}$ extracts set Y and sends $(\text{intersect}, Y, \text{setid})$ to $\mathcal{F}_{\text{PSI}}$ at step (7).

We are going to show the simulated execution is indistinguishable from real-world protocol execution.

Hybrid $\mathcal{H}_0$ Same as real-world execution in $(\mathcal{F}_{\text{C-VOLE}}, H_1, H_2)$ model.

Hybrid $\mathcal{H}_1$ Same as Hybrid $\mathcal{H}_0$ except $\mathcal{S}$ emulates $\mathcal{F}_{\text{C-VOLE}}$, random oracle H_1, H_2 , and simulates the messages to $\mathcal{A}$ as follows:

For step (1-2), $\mathcal{S}$ samples $\text{Hcom}_p \leftarrow \mathbb{F}^N$, and publishes it. In Hybrid $\mathcal{H}_0$, Hcom_p generated from commitment scheme. According to Theorem 7, Hcom_p is computational indistinguishable from pseudorandom vector over $\mathbb{F}^N$, which is indistinguishable from this hybrid.

For step (3), $\mathcal{S}$ emulates H_1 and samples $c^s \leftarrow \mathbb{F}$ for each w^s to $\mathcal{A}$. In hybrid $\mathcal{H}_0$, c^s is uniformly distributed over $\mathbb{F}$ and indistinguishable from this hybrid.

For step (4), $\mathcal{S}$ samples $w^r \leftarrow \mathbb{F}, r \leftarrow \{0, 1\}^k$ as an honest P_1 does in hybrid $\mathcal{H}_0$.

For step (7), $\mathcal{S}$ emulates H_1, H_2 , and samples $H_1(y_i) \leftarrow \mathbb{F}, ey_i \leftarrow \{0, 1\}^{\text{out}}$ to $\mathcal{A}$, which is indistinguishable from the output of a random oracle. $\mathcal{S}$ records all the programmed pairs and triple. $\mathcal{S}$ receives EY from $\mathcal{A}$ and checks each $ey_i \in EY$ whether been recorded and the previous query is computed correctly to extract $\mathcal{A}$'s private set elements and sends them to $\mathcal{F}_{\text{PSI}}$. Thus, the output of $\mathcal{S}$ and honest party is same as the output of $\mathcal{A}$ and honest party.

Thus, this hybrid is identical to the previous one.

Hybrid $\mathcal{H}_2$ Same as Hybrid $\mathcal{H}_1$ except $\mathcal{S}$ aborts at step (6) if $c'^s \neq c^s$ or $w'^s \neq w^s$. It is indistinguishable from P_1 aborts if $c^s \neq H_1(w^s)$ in hybrid $\mathcal{H}_1$. Thus, this hybrid is identical to the previous one.

This concludes the proof. □

APPENDIX C

Appendix for Chapter 5

C.1. Proof of Theorem 10

Theorem 17 (Theorem 10, restated). *Given a binary matrix $\mathbf{A} \in \{0, 1\}^{N \times n}$, we define distribution $\mathcal{D}_{\mathbf{A}}$ over $\{-1, +1\}^N$ as follows: 1) sample a binary vector $\mathbf{s} \in \{0, 1\}^n$ uniformly at random and 2) output $\mathbf{r} = 2(\mathbf{A}\mathbf{s} \bmod 2) - \mathbf{1}$. Let $d > 1$ is a constant, $n \geq 4d$ and $N \leq n^{d/2}$. For an overwhelming fraction of possible choices of $\mathbf{A}$ with each row having Hamming weight d and $\mathbf{A}_i \neq \mathbf{A}_j$ for all $i \neq j$, we have distribution $\mathcal{D}_{\mathbf{A}}$ is $\log \log(n)$ -inadmissible.*

Proof. We begin by analyzing the properties under the condition that $\text{HW}(\mathbf{A}_i) = \text{HW}(\mathbf{A}_j) = d$ and $\mathbf{A}_i \oplus \mathbf{A}_j = \mathbf{v}$, where $t = \text{HW}(\mathbf{v}) \neq 0$. More specifically, the following properties must hold:

1. Since both $\mathbf{A}_i$ and $\mathbf{A}_j$ have Hamming weight d , the Hamming weight of their XOR satisfies

$$t = \text{HW}(\mathbf{A}_i \oplus \mathbf{A}_j) \leq \text{HW}(\mathbf{A}_i) + \text{HW}(\mathbf{A}_j) = 2d .$$

2. On the non-zero coordinates of $\mathbf{v}$, the vectors $\mathbf{A}_i$ and $\mathbf{A}_j$ must differ at every position. Consequently, among these t coordinates, each vector contains exactly $t/2$ ones, which contributes a factor of $\binom{t}{t/2}$ possible choices.
3. On the remaining $n - t$ coordinates, $\mathbf{A}_i$ and $\mathbf{A}_j$ must be identical and together contain exactly $d - t/2$ ones to preserve the Hamming weight constraint $\text{HW}(\mathbf{A}_i) = \text{HW}(\mathbf{A}_j) = d$, contributing a factor of $\binom{n-t}{d-t/2}$.
4. Consequently, $\text{HW}(\mathbf{v})$ must be even, since $\mathbf{A}_i$ and $\mathbf{A}_j$ have the same Hamming weight, implying that the number of positions where the two vectors differ is even.

We now bound the correlation by analyzing the pairwise probability that two random weight- d vectors $\mathbf{A}_i$ and $\mathbf{A}_j$ satisfy $\mathbf{A}_i \oplus \mathbf{A}_j = \mathbf{v}$. The total sample space of $(\mathbf{A}_i, \mathbf{A}_j)$ has size $\binom{n}{d}^2 - \binom{n}{d}$ conditioned on $\text{HW}(\mathbf{A}_i) = \text{HW}(\mathbf{A}_j) = d$ and $\mathbf{A}_i \neq \mathbf{A}_j$. The number of valid pairs $(\mathbf{A}_i, \mathbf{A}_j)$ that $\mathbf{A}_i \oplus \mathbf{A}_j = \mathbf{v}$ is $\binom{t}{t/2} \binom{n-t}{d-t/2}$. Let

$$\mathcal{A} \stackrel{\text{def}}{=} \{\mathbf{a} \in \{0, 1\}^n \mid \text{HW}(\mathbf{a}) = d\}.$$

We define p_v as

$$p_v \stackrel{\text{def}}{=} \Pr_{\mathbf{A}_i, \mathbf{A}_j \leftarrow \mathcal{A}, \mathbf{A}_i \neq \mathbf{A}_j} [\mathbf{A}_i \oplus \mathbf{A}_j = \mathbf{v}] = \frac{\binom{t}{t/2} \binom{n-t}{d-t/2}}{\binom{n}{d}^2 - \binom{n}{d}}.$$

We show that for constant $2d \geq t$ and $n \geq 4d$, and for any nonzero vector $\mathbf{v}$, the probability p_v is maximized when $t = 2$. That is,

$$p_{\max} \stackrel{\text{def}}{=} \max_{\mathbf{v} \in \mathbb{F}_2^n \setminus \{\mathbf{0}_n\}} p_v = \frac{2 \binom{n-2}{d-1}}{\binom{n}{d}^2 - \binom{n}{d}} = \frac{2d(n-d)}{n(n-1)(\binom{n}{d} - 1)} \leq \frac{2d^{d+1}}{n^{d+1}}.$$

Since the denominators of p_v do not depend on t , maximizing p_v is equivalent to maximizing $F(t) \stackrel{\text{def}}{=} \binom{t}{t/2} \binom{n-t}{d-t/2}$. We now show that for every even $t \geq 2$,

$$\begin{aligned} \frac{F(t+2)}{F(t)} &= \frac{\binom{t+2}{t/2+1}}{\binom{t}{t/2}} \cdot \frac{\binom{n-t-2}{d-t/2-1}}{\binom{n-t}{d-t/2}} = \frac{4(t+1)}{t+2} \cdot \frac{(d-t/2)(n-d-t/2)}{(n-t)(n-t-1)} \\ &\leq 4 \cdot \frac{d-t/2}{n-t-1} \leq 4 \cdot \frac{d-t/2}{4d-t-1} \leq 1. \end{aligned}$$

For any non-zero $\mathbf{v} \in \{0, 1\}^n$, we define

$$\mathcal{S}_v \stackrel{\text{def}}{=} \{(i, j) \in [N]^2 \mid \mathbf{A}_i \oplus \mathbf{A}_j = \mathbf{v}\}$$

Thus, we obtain a partition of $\{(i, j) \in [N]^2 \mid i \neq j\}$ as

$$\{(i, j) \in [N]^2 \mid i \neq j\} = \bigcup_{v \neq \mathbf{0}} \mathcal{S}_v.$$

For nonzero $\mathbf{v}$, each $\mathcal{S}_{\mathbf{v}}$ is an inadmissible set. By definition of $\mathbf{A}$, for any $i \neq j$, we have $\mathbf{A}_i \neq \mathbf{A}_j$. Therefore, for any $i \neq j$, we have

$$\begin{aligned}\mathbb{E}_{\mathbf{r} \leftarrow \mathcal{D}}[\mathbf{r}[i]\mathbf{r}[j]] &= \mathbb{E}_{\mathbf{s} \leftarrow \{0,1\}^n} [(-1)^{\mathbf{A}_i \mathbf{s}} (-1)^{\mathbf{A}_j \mathbf{s}}] \\ &= \mathbb{E}_{\mathbf{s} \leftarrow \{0,1\}^n} [(-1)^{\langle \mathbf{A}_i \oplus \mathbf{A}_j, \mathbf{s} \rangle}] = 0.\end{aligned}$$

Moreover, for any $(i, j), (k, l) \in \mathcal{S}_{\mathbf{v}}$,

$$\begin{aligned}\mathbb{E}_{\mathbf{r} \leftarrow \mathcal{D}}[\mathbf{r}[i]\mathbf{r}[j]\mathbf{r}[k]\mathbf{r}[l]] &= \mathbb{E}_{\mathbf{s} \leftarrow \{0,1\}^n} [(-1)^{(\mathbf{A}_i \oplus \mathbf{A}_j \oplus \mathbf{A}_k \oplus \mathbf{A}_l) \mathbf{s}}] \\ &= \mathbb{E}_{\mathbf{s} \leftarrow \{0,1\}^n} [(-1)^{\langle \mathbf{v} \oplus \mathbf{v}, \mathbf{s} \rangle}] = 1.\end{aligned}$$

Finally, for any $\mathbf{u} \neq \mathbf{v}$ and any $(i, j) \in \mathcal{S}_{\mathbf{u}}, (k, l) \in \mathcal{S}_{\mathbf{v}}$,

$$\begin{aligned}\mathbb{E}_{\mathbf{r} \leftarrow \mathcal{D}}[\mathbf{r}[i]\mathbf{r}[j]\mathbf{r}[k]\mathbf{r}[l]] &= \mathbb{E}_{\mathbf{s} \leftarrow \{0,1\}^n} [(-1)^{(\mathbf{A}_i \oplus \mathbf{A}_j \oplus \mathbf{A}_k \oplus \mathbf{A}_l) \mathbf{s}}] \\ &= \mathbb{E}_{\mathbf{s} \leftarrow \{0,1\}^n} [(-1)^{\langle \mathbf{v} \oplus \mathbf{u}, \mathbf{s} \rangle}] = 0.\end{aligned}$$

We then consider the tail probability $\Pr[|\mathcal{S}_{\mathbf{v}}| \geq k]$ for a fixed nonzero $\mathbf{v}$ and any positive integer k . Let $(x)_k \stackrel{\text{def}}{=} x(x-1) \cdots (x-k+1)$ denote the falling factorial. Define the indicator variable $X_{(i,j)}$ such that $X_{(i,j)} = 1$ if and only if $\mathbf{A}_i \oplus \mathbf{A}_j = \mathbf{v}$, and $X_{(i,j)} = 0$ otherwise. Then

$$|\mathcal{S}_{\mathbf{v}}| = \sum_{i \neq j} X_{(i,j)}.$$

If $|\mathcal{S}_{\mathbf{v}}| \geq k$, then the falling factorial $(|\mathcal{S}_{\mathbf{v}}|)_k \geq k!$. Therefore, by Markov's inequality,

$$\Pr[|\mathcal{S}_{\mathbf{v}}| \geq k] \leq \Pr[(|\mathcal{S}_{\mathbf{v}}|)_k \geq k!] \leq \frac{\mathbb{E}[(|\mathcal{S}_{\mathbf{v}}|)_k]}{k!}.$$

For any collection of indicator variables $\{X_a\}_a$, the combinatorial identity

$$\left(\sum_a X_a \right)_k = \sum_{(a_1, \dots, a_k) \text{ distinct}} X_{a_1} \cdots X_{a_k}$$

holds. This identity follows since $\sum_a X_a$ counts the number of indices such that $X_a = 1$. Consequently, the falling factorial $(\sum_a X_a)_k$ equals the number of ways to choose k ordered distinct indices from those satisfying $X_a = 1$. The right-hand side enumerates exactly these choices, because the product $X_{a_1} \cdots X_{a_k}$ equals 1 if and only if all selected indices are distinct and each satisfies $X_{a_r} = 1$; otherwise, the product is 0. Therefore, the two sides of the identity count the same quantity. Applying this identity with $a_r = (i_r, j_r)$ with $i_r \neq j_r$, we have

$$(|\mathcal{S}_v|)_k = \sum_{(a_1, \dots, a_k) \text{ distinct}} X_{a_1} \cdots X_{a_k}.$$

Taking the expectation gives

$$\mathbb{E} \left[(|\mathcal{S}_v|)_k \right] = \sum_{((i_1, j_1), \dots, (i_k, j_k)) \text{ distinct}} \Pr \left[\bigwedge_{r=1}^k (\mathbf{A}_{i_r} \oplus \mathbf{A}_{j_r} = \mathbf{v}) \right].$$

Let us fix k distinct pairs for the analysis that follows. Note that for any pair (i, j) satisfying $\mathbf{A}_i \oplus \mathbf{A}_j = \mathbf{v}$, the reversed pair (j, i) also satisfies the same condition. Hence, each unordered pair $\{i, j\}$ corresponds to at most two ordered pairs. Without loss of generality, we may restrict attention to a set of $k/2$ unordered pairs, denoted by $\{i_1, j_1\}, \dots, \{i_{k/2}, j_{k/2}\}$ with $i_t > j_t$ for all t . Let set $\mathcal{I} \stackrel{\text{def}}{=} \{i_1, j_1, \dots, i_{k/2}, j_{k/2}\}$ and $m \stackrel{\text{def}}{=} |\mathcal{I}|$ that denotes the number of distinct indices involved. The subsequent analysis separates the contribution into two cases. We first analyze the good case, corresponding to $m = k$, where all indices are distinct. The bad case, where $m < k$ due to index collisions, will be ruled out later based on the structural property that for all $i \neq j$, we have $\mathbf{A}_i \neq \mathbf{A}_j$.

Good Case: When $m = 2k$, all indices in $(i_1, j_1, \dots, i_{k/2}, j_{k/2})$ are distinct. Since the rows of $\mathbf{A}$

are sampled without replacement (and hence are pairwise distinct), they are not independent. Nevertheless, we can upper-bound the joint probability using a sequential-exposure argument. Specifically,

$$\Pr \left[\bigwedge_{r=1}^{k/2} (\mathbf{A}_{i_r} \oplus \mathbf{A}_{j_r} = \mathbf{v}) \right] = \prod_{r=1}^{k/2} \Pr \left[\mathbf{A}_{i_r} \oplus \mathbf{A}_{j_r} = \mathbf{v} \mid \{\mathbf{A}_{i_s} \oplus \mathbf{A}_{j_s} = \mathbf{v}\}_{s < r} \right].$$

For each r , the conditional probability is at most $p_{\max}$, since conditioning only restricts the set of available rows. Hence,

$$\Pr \left[\bigwedge_{r=1}^{k/2} (\mathbf{A}_{i_r} \oplus \mathbf{A}_{j_r} = \mathbf{v}) \right] \leq p_{\max}^{k/2}.$$

The number of ordered tuples $(i_1, j_1, \dots, i_{k/2}, j_{k/2})$ is at most N^k . Therefore, the total contribution from all good tuples satisfies

$$\begin{aligned} \mathbb{E}[(|\mathcal{S}_{\mathbf{v}}|)_k \mid m = 2k] &\leq N^k \cdot p_{\max}^{k/2} \leq n^{d/2 \cdot k} \cdot \left(\frac{2d^{d+1}}{n^{d+1}} \right)^{k/2} \\ &\leq 2^{k/2} \cdot d^{(d+1)k/2} \cdot n^{-k/2}. \end{aligned}$$

Bad Case. Suppose $m < k$, i.e., some indices appear in multiple pairs. We show that this case cannot occur under the assumption that $\mathbf{A}_i \neq \mathbf{A}_j$ for all $i \neq j$.

Construct an undirected graph G over vertex set $[N]$, where each constraint $\mathbf{A}_{i_r} \oplus \mathbf{A}_{j_r} = \mathbf{v}$ introduces an edge (i_r, j_r) .

If some connected component of G contains at least two edges, then there exists a path of length at least 2, say (u, u_1) and (u_1, u_2) . These imply

$$\mathbf{A}_{u_1} = \mathbf{A}_u \oplus \mathbf{v} \quad \text{and} \quad \mathbf{A}_{u_2} = \mathbf{A}_{u_1} \oplus \mathbf{v} = \mathbf{A}_u.$$

Since $u_2 \neq u$, this contradicts the assumption that $\mathbf{A}_i \neq \mathbf{A}_j$ for all $i \neq j$.

Therefore, every connected component contains at most one edge, which implies $m = k$. Hence, the bad case cannot occur.

Combining the good contribution and the bad contribution, we have

$$\begin{aligned}\mathbb{E}\left[(|\mathcal{S}_v|)_k\right] &= \mathbb{E}\left[(|\mathcal{S}_v|)_k \mid m = 2k\right] + \mathbb{E}\left[(|\mathcal{S}_v|)_k \mid m \leq 2k\right] \\ &\leq 2^{k/2} \cdot d^{(d+1)k/2} \cdot n^{-k/2}.\end{aligned}$$

By the factorial moment method and Markov's inequality, setting $k = \log \log(n)$ yields $\frac{\mathbb{E}[(|\mathcal{S}_v|)_k]}{k!}$ is negligible, and consequently

$$\begin{aligned}\Pr\left[|\mathcal{S}_v| \geq \log \log(n)\right] &\leq \Pr\left[(|\mathcal{S}_v|)_{\log \log(n)} \geq (\log \log(n))!\right] \\ &\leq \text{negl}(n).\end{aligned}$$

Note that $\text{HW}(\mathbf{A}_i \oplus \mathbf{A}_j)$ is at most $2d$ and must be even. Hence, the number of nonzero vectors $\mathbf{v} \neq 0$ that can appear is

$$\sum_{1 \leq t \leq 2d, t \text{ is even}} \binom{n}{t} \leq n^{2d}.$$

Applying the union bound and using the fact that d is constant, we get

$$\begin{aligned}\Pr\left[\Delta(\mathcal{D}_{\mathbf{A}}) \geq \log \log(n)\right] &\leq \sum_{\mathbf{v} \neq 0} \Pr\left[|\mathcal{S}_v| \geq \log \log(n)\right] \\ &\leq n^{2d} \cdot \text{negl}(n) \leq \text{negl}(n).\end{aligned}$$

□

C.2. Proof of Theorem 12

Theorem 18 (Theorem 12, restated). *Let H be a random oracle, and assume that the SWHE scheme satisfies ciphertext pseudorandomness. Then, Protocol Π_{PJc} (Figure 5.2) securely realizes the functionality for private join and compute over distributed input sets in the presence of semi-honest adversaries in the $(\mathcal{F}_{\text{Rand}}, \mathcal{F}_{\text{ThSWHE}}, \mathcal{F}_{\text{2PC}})$ -hybrid model.*

Proof. Let $\mathcal{A}$ be a probabilistic polynomial-time (PPT) adversary that corrupts at most one of the servers, namely Agg or CSP, possibly colluding with a subset of the parties. We construct a PPT simulator $\mathcal{S}$, which runs the PPT adversary $\mathcal{A}$ as a subroutine and emulates functionalities $\mathcal{F}_{\text{Rand}}$, $\mathcal{F}_{\text{ThSWHE}}$ and $\mathcal{F}_{2\text{PC}}$. $\mathcal{S}$ simulates the random oracle H by responding to random strings while keeping the consistency of responses. Let $\mathcal{C} \subseteq \{P_i\}_{i \in [w]}$ be the corrupted parties. $\mathcal{S}$ proceeds as follows.

Initialization:

1. $\mathcal{S}$ emulates the $\mathcal{F}_{\text{ThSWHE}}$ by executing $(\text{params}', \text{sk}', \text{pk}', \text{ek}') \leftarrow \text{KeyGen}(1^\lambda, 1^L)$, and sending (ek', pk') to $\mathcal{A}$. $\mathcal{S}$ emulates $\mathcal{F}_{\text{Rand}}$ by sampling a random $\text{seed} \leftarrow \{0, 1\}^\lambda$. $\mathcal{S}$ sends seed to $\mathcal{A}$.
2. For each $\ell \in [tk]$, $\mathcal{S}$ samples $\mathbf{s}^\ell \in \{0, 1\}^n$ and generates a fresh encryption $\llbracket 0 \rrbracket$. If the corrupted server is CSP, $\mathcal{S}$ sends $\llbracket 0 \rrbracket$ to $\mathcal{A}$; otherwise, it publishes $\llbracket 0 \rrbracket$.
3. For each $\ell \in [tk]$, $\mathcal{S}$ generates two counters, c_1^ℓ and c_2^ℓ , initialized to 0.

Union operation over the set of P on (input, p , $\mathcal{S}$):

1. If $P \notin \mathcal{C}$ and Agg is corrupted, $\mathcal{S}$ sends a fresh $\llbracket 0 \rrbracket$ to $\mathcal{A}$.
2. $\mathcal{S}$ samples a matrix $\mathbf{A} \in \{0, 1\}^{N \times n}$ uniformly at random subject to the constraint that each row has Hamming weight exactly d and all rows are pairwise distinct.
3. For $\ell \in [tk]$, $\mathcal{S}$ computes its sketching result:

$$z^\ell = \sum_{i \in [m]} b_i \mathbf{r}^\ell[\mathbf{A}_i],$$

where $\mathbf{r}^\ell[i] = 2(\mathbf{A}_i \mathbf{s}^\ell \bmod 2) - 1$.

4. For $\ell \in [tk]$, $\mathcal{S}$ updates $c_p^\ell \leftarrow c_p^\ell + z^\ell$, where $p \in \{1, 2\}$.

Join Computation via 2PC on Servers:

1. For each $\ell \in [tk]$, $\mathcal{S}$ emulates $\mathcal{F}_{\text{ThSWHE}}$'s Dec command by uniformly sampling $d_1^\ell, d_2^\ell \in \mathcal{M}$ and sending (d_1^ℓ, d_2^ℓ) to $\mathcal{A}$.
2. If the corrupted server is Agg, $\mathcal{S}$ emulates $\mathcal{F}_{2\text{PC}}$ by sending $z = \text{MoM}(\{c_1^\ell c_2^\ell\}^{\ell \in [tk]})$ to $\mathcal{A}$.

We use a sequence of hybrids to show that real-world and ideal-world executions are computationally indistinguishable.

Hybrid 0. This hybrid (denoted by $\mathcal{G}_0$) is the real-world execution.

Hybrid 1. This hybrid (denoted by $\mathcal{G}_1$) is the same as $\mathcal{G}_0$, except that emulating $\mathcal{F}_{\text{Rand}}$, $\mathcal{F}_{\text{ThSWHE}}$ and $\mathcal{F}_{2\text{PC}}$, and simulating random oracle H honestly.

It is obvious that $\mathcal{G}_1$ is perfectly indistinguishable from $\mathcal{G}_0$. Hybrid $\mathcal{G}_1$ can obtain all secrets of $\mathcal{A}$ by emulating $\mathcal{F}_{\text{ThSWHE}}$.

Hybrid 2. This hybrid (denoted by $\mathcal{G}_2$) is identical to $\mathcal{G}_1$, except that $z = \text{MoM}(\{c_1^\ell c_2^\ell\}^{\ell \in [tk]})$ is computed in the plaintext.

We prove that $\mathcal{G}_2$ is statistically indistinguishable from $\mathcal{G}_1$ by the decryption correctness of the underlying SWHE scheme.

Hybrid 3. This hybrid (denoted by $\mathcal{G}_3$) is identical to $\mathcal{G}_1$, except that every ciphertext sent to $\mathcal{A}$ is replaced with a fresh encryption of 0. This corresponds to the ideal-world execution.

We prove that $\mathcal{G}_3$ is computationally indistinguishable from $\mathcal{G}_2$ via a reduction to the ciphertext pseudorandomness property of the underlying SWHE scheme.

In conclusion, the joint distribution of the outputs of $\mathcal{S}$ and honest participants in the ideal-world execution is computationally indistinguishable from that of the outputs of $\mathcal{A}$ and honest participants in the real-world execution, which completes the proof. $\square$

C.3. Our 4-Wise Independent Construction

Theorem 19. (*Our 4-Wise Independent Construction*) Let $m = n^2$, and let $\mathbf{c} \leftarrow \mathbb{F}_2^n$, $\mathbf{d} \leftarrow \mathbb{F}_2^n$, and $\mathbf{e} \leftarrow \mathbb{F}_2^{2n}$ be independent uniformly random bit vectors. Define $\mathbf{f} \in \mathbb{F}_2^m$, such that for all $i, j \in [n], k = (i-1)n + j \in [\ell]$, $\mathbf{f}_k = \mathbf{c}_i \oplus \mathbf{d}_j \oplus \mathbf{e}_{i+j}$. Then, $\mathbf{f}$ is four-wise independent over $\mathbb{F}_2^m$.

Proof. Let $k_\ell = (i_\ell - 1)n + j_\ell$ for $\ell = 1, 2, 3, 4$ be four queried positions in $\mathbf{f}$. From the definition of $\mathbf{f}$, we have: $\mathbf{f}_{k_\ell} = \mathbf{c}_{i_\ell} \oplus \mathbf{d}_{j_\ell} \oplus \mathbf{e}_{i_\ell + j_\ell}$, for $\ell = 1, 2, 3, 4$. Define $\mathbf{F}_{k_\ell} \in \mathbb{F}^{4n}$ as a row vector with element one at indexes i_ℓ , $j_\ell + n$, and $i_\ell + j_\ell + 2n$, respectively. All other entries are zero. We have

$$\mathbf{f}_{k_\ell} = \mathbf{F}_{k_\ell} \cdot \begin{bmatrix} \mathbf{c} & \mathbf{d} & \mathbf{e} \end{bmatrix}^T \quad (\text{C.1})$$

To establish 4-wise independence, for any four distinct indices $k_1, k_2, k_3, k_4 \in [\ell]$, the joint distribution of the corresponding outputs is uniform over $\mathbb{F}_2^4$. We observe that, if $\text{rank}(\mathbf{F}_{k_1, k_2, k_3, k_4}) = 4$, $\{\mathbf{f}_{k_\ell}\}_{\ell \in [4]}$ is uniformly distributed over $\mathbb{F}_2^4$. We prove that, in our construction, $\text{rank}(\mathbf{F}_{k_1, k_2, k_3, k_4})$ is always equal to 4. We distinguish the cases as follows:

1. If there are four i_ℓ s, $\ell \in \{1, 2, 3, 4\}$ that composes $\mathbf{F}_{k_\ell}$ respectively, no matter how j_ℓ is selected in each $\mathbf{F}_{k_\ell}$, $\text{rank}(\mathbf{F}_{k_1, k_2, k_3, k_4}) = 4$.
2. If there are three i_ℓ s, $\ell \in \{1, 2, 3\}$, that composes $\mathbf{F}_{k_\ell}$, $\ell \in \{1, 2, 3, 4\}$, we assume i_3 appears twice in $\mathbf{F}_{k_3}$ and $\mathbf{F}_{k_4}$. According to our construction, index $k_\ell = (i_\ell - 1)n + j_\ell$. Thus, given $i_3 = i_4$, $\mathbf{F}_{k_3}$ and $\mathbf{F}_{k_4}$ are constructed by two different j_3 and j_4 . We observe that, no matter how j_1 and j_2 are selected in $\mathbf{F}_{k_1}$ and $\mathbf{F}_{k_2}$, $\text{rank}(\mathbf{F}_{k_1, k_2, k_3, k_4}) = 4$.
3. If there are two i_ℓ s, $\ell \in \{1, 2\}$, that determines $\mathbf{F}_{k_\ell}$, $\ell \in \{1, 2, 3, 4\}$, we assume two cases: 1) i_1 appears once in $\mathbf{F}_{k_1}$ and i_2 appears three times in $\mathbf{F}_{k_\ell}$, $\ell \in \{2, 3, 4\}$. Same as above analysis, given $i_2 = i_3 = i_4$, $\mathbf{F}_{k_2}$, $\mathbf{F}_{k_3}$ and $\mathbf{F}_{k_4}$ are constructed by three different j_2 , j_3 and j_4 . We observe that, no matter how j_1 is selected in $\mathbf{F}_{k_1}$, $\text{rank}(\mathbf{F}_{k_1, k_2, k_3, k_4}) = 4$. 2) i_1 appears twice in $\mathbf{F}_{k_1}$ and $\mathbf{F}_{k_3}$ and i_2 appears twice in $\mathbf{F}_{k_2}$ and $\mathbf{F}_{k_4}$. Same as above, given $i_1 = i_3$, $\mathbf{F}_{k_3}$ and $\mathbf{F}_{k_1}$ are constructed by two different j_1 , j_3 . Given $i_2 = i_4$, $\mathbf{F}_{k_4}$ and $\mathbf{F}_{k_2}$ are constructed by two different j_4 , j_2 . We observe that, no matter how j_4 and j_2 are selected, even $j_3 = j_4$, $j_1 = j_3$, $\text{rank}(\mathbf{F}_{k_1, k_2, k_3, k_4}) = 4$.
4. If there are one i_ℓ for $\ell \in \{1\}$, that determines $\mathbf{F}_{k_\ell}$, $\ell \in \{1, 2, 3, 4\}$, same as above, four different j_ℓ for $\ell \in \{1, 2, 3, 4\}$ are needed to achieve four different entries k_ℓ , $\ell \in \{1, 2, 3, 4\}$. We have

$\text{rank}(\mathbf{F}_{k_1, k_2, k_3, k_4}) = 4$ in this case.

Thus, in our construction, for any distinct indices k_1, k_2, k_3, k_4 , $\text{rank}(\mathbf{F}_{k_1, k_2, k_3, k_4})$ is always equal to 4. We conclude that for any distinct indices k_1, k_2, k_3, k_4 and any $a_1, a_2, a_3, a_4 \in \{0, 1\}$,

$$\Pr \left[\mathbf{f}_{k_1} = a_1 \wedge \mathbf{f}_{k_2} = a_2 \wedge \mathbf{f}_{k_3} = a_3 \wedge \mathbf{f}_{k_4} = a_4 \right] = \frac{1}{16}.$$

□

C.4. Performance under Large Polynomial Modulus Degree

We evaluate the performance under larger polynomial modulus degree 16384, where ciphertexts become larger and each homomorphic multiplication becomes more expensive. We report the results for plain modulus 2^{24} across the three protocol phases in Table C.3. In the server initialization phase, when $n = 128$, the LAN runtime is 3.1s, with 149.2 MB of server communication and 155.5 MB of broadcast communication. The cost scales approximately linearly with n and is roughly $3\times$ higher than that under polynomial modulus degree 8192. For client local sketching, the amortized cost is 90.7 ms per element when $d = 8$, also about $3\times$ higher than the cost under degree 8192. The client-to-server communication increases to 0.71 MB. In the server aggregation and output phase, both homomorphic aggregation and threshold decryption become more expensive due to the increased ciphertext size. For encrypted sketch aggregation over 2000 clients, the aggregation time is 0.83 s, while LAN threshold decryption takes 0.04 s. The subsequent 2PC computation incurs additional overhead due to the use of a multi-limb representation to prevent overflow under large payloads, resulting in a $4\times$ cost increase, corresponding to 0.1 s of LAN time and 12.9 MB of communication.

		$n = 128$	$n = 256$	$n = 512$	$n = 1024$
Comp. (s)	LAN	3.1	5.7	10.8	20.7
	WAN	16.9	31.8	60.6	118.7
Comm. (MB)		149.2	298.4	596.7	1193.4
Publish (MB)		155.5	304.7	603.0	1199.7

Table C.1: Server initialization costs under polynomial modulus degree 16384 for different seed lengths n .

m	2^0	2^4	2^8	2^{12}	2^{16}
$d = 4$	100.8	95.4	86.3	41.2	27.5
$d = 5$	127.6	122.6	108.6	66.5	54
$d = 6$	161.6	155.7	127.6	65.9	53.8
$d = 7$	188.8	182.2	149.9	91.6	79.8
$d = 8$	223.6	211.4	160.8	90.7	80.1

Table C.2: Amortized per-element client sketching cost (ms/elem.) under polynomial modulus degree 16384.

		Aggregate Encrypted Sketch	Threshold Decryption	2PC	Total
Comp. (s)	LAN	0.83	0.04	0.1	0.97
	WAN	0.83	0.05	2.39	3.27
Comm. (MB)		0	2.6	12.9	15.5

Table C.3: Server aggregation and output costs under polynomial modulus degree 16384.