TOWARDS PRIVACY-PRESERVING AND ROBUST WEB OVERLAYS

JIA YAOQI

(*B.Eng., Huangzhong University of Science and Technology*)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
DEPARTMENT OF COMPUTER SCIENCE
NATIONAL UNIVERSITY OF SINGAPORE

2017

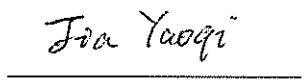
Supervisor:
Associate Professor Liang Zhenkai

Examiners:
Associate Professor Chang Ee Chien
Assistant Professor Kang Minsuk
Professor Duan Haixin, Tsinghua University

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.



Jia Yaoqi

30 March 2017

ACKNOWLEDGEMENTS

This thesis would not be possible without the help of my advisors Zhenkai Liang and Prateek Saxena. First, I would like to thank Zhenkai for his mentorship and encouragement for exploring new directions. No road of flowers leads to glory, which also applies to the journey pursuing my Ph.D.. Zhenkai's earnest advice motivates me to move forward continuously in spite of the difficulties and twists being incurred in my journey. Besides, Prateek taught me to think big and aim high since we first collaborated. His enthusiasm and creativity inspire me to broaden my vision and explore distinct security research areas. In the meantime, I have learned a lot about the technical skills of security research from both Zhenkai and Prateek, and have benefited tremendously from the discussions over these years.

I would also like to thank my thesis committee members, Ee-Chien Chang, Min Suk Kang and Haixin Duan, for their valuable feedback on this thesis. Many thanks to Haifeng Yu, Seth Gilbert, Erik-Oliver Blass, Travis Mayberry and other professors for their insightful discussions and feedback on my work in this thesis.

I am indebted to all my collaborators, both at NUS and outside, over these years for all of hard work and fun time. The solid work in this thesis is inseparable from their remarkable contributions. Many thanks to Hong Hu (introducing me to the lab), Xinshu Dong, Shruti Tople, Tarik Moataz, Guangdong Bai, Xiaolei Li, Enrico Budianto, Zheng Leong Chua, Shuo Chen, Roland Yap, Deli Gong, Jian Mao, Yue Chen, Behnaz Hassanshahi, and Ziqi Yang.

I would like to thank all my colleagues in the NUS security group: Chunwang Zhang, Ting Dai, Shweta Shinde, Loi Luu, Hung Dang, Inian Parameshwaran, Shiqi Shen and many others. I have learned a lot from them and shared numerous happy memories with them during these years. Thanks to my friends across the world for your unforgettable support on my life and research.

Finally, I would like to thank my parents and grandparents for their unconditional love. I am indebted to my Mom for her unwavered love and support; my Dad for his earnest instructions and encouragement to stimulate me to pursue my dreams. I am especially grateful to my beloved, Meng Guo, for her continuous love, encouragement and support.

Contents

SUMMARY **x**

LIST OF TABLES **xi**

LIST OF FIGURES **xiii**

1 Introduction **1**

1.1 Thesis Overview 5

1.1.1 APAC: An Anonymous Peer-assisted CDN 5

1.1.2 OBLIVP2P: An Oblivious P2P Content Sharing System . 6

1.1.3 Robust Synchronous P2P Primitives Using SGX Enclaves 7

2 Background **11**

2.1 Web Overlays 11

2.2 Related Work 13

2.2.1 Anonymous Communication Systems Against Partial Adversaries 13

2.2.2 Long-term Traffic Analysis of Global Adversaries 14

2.2.3 Robust P2P Primitives Against Byzantine Adversaries . 15

3 Anonymity in Peer-assisted CDNs: Inference Attacks and Mitigation **17**

3.1 Introduction 17

3.2 Motivation & Problem Statement 22

3.2.1 Inference Attacks & Real-world Examples 22

3.2.2 Problem Statement 24

3.3 Anonymous Peer-assisted CDN 26
3.3.1 Design of APAC 27
3.3.2 Circuit Construction 30
3.3.3 Parameters Selection 32
3.3.4 Anonymity Analysis 34
3.3.4.1 Analysis of Initiator Anonymity 34
3.3.4.2 Analysis of Responder Anonymity 37
3.4 Implementation of APAC 38
3.4.1 Components in APAC 38
3.4.2 Content Delivery in APAC 40
3.5 Performance Evaluation 43
3.5.1 Measurement Setup 44
3.5.2 Bandwidth Saving 45
3.5.3 Network Latency 46
3.5.4 Performance under Churn 50
3.5.5 Load on Peers 51
3.6 Security Analysis 52
3.6.1 Degree of Initiator/Responder Anonymity in APAC . . . 52
3.6.2 Degree of Anonymity in Current Peer-assisted CDNs . . 55
3.6.3 Analysis of Churn in APAC 56
3.7 Related Work 60
3.7.1 Security & Privacy in Peer-assisted CDNs 60
3.7.2 Anonymous Communication Systems 61
3.8 Summary 62

4 OBLIVP2P: An Oblivious Peer-to-Peer Content Sharing System 63
4.1 Introduction 63
4.1.1 Approach 65
4.1.2 System and Results 66
4.2 Problem 67

4.2.1 BitTorrent: A P2P Protocol . . . 68
4.2.2 Threat Model . . . 68
4.2.3 Insufficiency of Existing Approaches . . . 70
4.2.4 Problem Statement . . . 71
4.3 Our Approach . . . 73
4.3.1 Background: Tree-Based ORAM . . . 73
4.3.2 Mapping an ORAM to a P2P setting . . . 74
4.3.3 OBLIVP2P-0 : Centralized Protocol . . . 75
4.3.4 OBLIVP2P-0 Analysis . . . 76
4.4 OBLIVP2P-1: Distributed Protocol . . . 77
4.4.1 Challenges . . . 77
4.4.2 Oblivious Selection . . . 79
4.4.2.1 Definitions . . . 79
4.4.2.2 OblivSel Overview . . . 81
4.4.2.3 Base Primitives . . . 82
4.4.2.4 OblivSel Instantiation . . . 84
4.4.3 OBLIVP2P-1: Complete Design . . . 85
4.4.4 Optimization: Handling Bursts . . . 88
4.5 Implementation and Evaluation . . . 91
4.5.1 Linear Scalability with Peers . . . 92
4.5.2 Latency Overhead and Breakdown . . . 97
4.5.3 Optimization Measurements . . . 98
4.6 OBLIVP2P-1 Analysis . . . 99
4.6.1 Performance . . . 99
4.6.2 Security Analysis . . . 101
4.7 Discussion . . . 104
4.8 Related Work . . . 106
4.9 Summary . . . 107

5 Robust Synchronous P2P Primitives Using SGX Enclaves 109

5.1 Introduction 109
5.2 Problem 114
5.2.1 Problem Definition 114
5.2.2 Attacker Model 115
5.2.3 Strawman Solution & Attacks 116
5.3 Solution Overview 120
5.3.1 SGX Features and Security Properties 120
5.3.2 Overview of Our Results 124
5.4 Enclaved Reliable Broadcast Protocol 126
5.4.1 Preliminaries 126
5.4.2 ERB details 127
5.4.3 Analysis 128
5.5 Enclaved Random Number Generation 130
5.5.1 Unoptimized ERNG 130
5.5.2 Optimized ERNG 132
5.5.3 Analysis 135
5.6 Evaluation 136
5.6.1 ERB Evaluation 137
5.6.2 ERNG Evaluation 139
5.6.3 Byzantine case 140
5.7 Primitives and Formal Definitions 142
5.7.1 Peer Channel 142
5.7.2 Failure Modes 144
5.7.3 Core Primitives 146
5.7.4 Implementing Blinded Channel using SGX 148
5.8 Rethinking Reliable Broadcast Protocols 151
5.8.1 Digital Signature Schemes 152
5.8.2 Early Stopping Schemes 155
5.9 Security Analysis 157

5.9.1 ERB Analysis 157
5.9.2 P2P Sanitization & Analysis 161
5.9.3 Unoptimized ERNG Analysis 164
5.9.4 Optimized ERNG 165
5.10 Discussions 168
5.10.1 Are Assumptions Reasonable? 168
5.10.2 Applications 170
5.11 Related Work 172
5.12 Summary 174

6 Conclusion **175**

SUMMARY

The World Wide Web gradually becomes an essential part of our daily life in the digital age. The web architecture used to be a client-server model, in which clients (or browsers) request and fetch web contents, such as HTML, JavaScript and CSS, from web servers. Recently peer-to-peer (P2P) techniques have been introduced into the web infrastructure, which empower browsers to directly communicate with each other and form a P2P web overlay. On one hand, this web overlay decentralizes the web to provide better availability of web resources and efficiency of transferring these resources. On the other hand, this also brings the open and unsolved problems like privacy issues to the new web architecture. In this thesis, we analyze privacy and robustness issues in web overlays, and propose solutions to address these issues using cryptographic and hardware primitives.

First, we present inference attacks in peer-assisted content delivery networks (CDNs) on top of web overlays, which can infer user's online activities such as browsing history. To thwart such attacks, we propose an anonymous peer-assisted CDN (APAC), which employs onion-routing techniques to conceal users' identities and uses region-based circuit selection algorithm to reduce performance overhead. Second, previous studies have shown that a global adversary is realistic, and it can reveal users' online activities (or access patterns) using long-term traffic analysis. Against such adversaries, we design an oblivious peer-to-peer content sharing system (OBLIVP2P), which uses new primitives such as distributed oblivious RAM (ORAM) in the P2P setting.

Lastly, we propose solutions to ensure the robustness of P2P primitives, as all the utilities and security / privacy properties provided by P2P protocols (including the aforementioned two protocols) are relied on the robustness of the correct execution of these protocols. Recent evidence suggests that malicious (or byzantine) nodes can easily join the open P2P systems and perniciously dis-

rupt the execution of the given protocol to weaken the core utility (e.g., Bitcoin) or the anonymity guarantee (e.g., Tor). To ensure the robustness of protocols against such byzantine adversaries in contrast to the honest-but-curious ones in the prior two works, we leverage a new hardware primitive, Intel software guard extensions (SGX). By enforcing our properties, we reduce the byzantine model to the general-omission model, where byzantine nodes have no extra advantage than omitting messages. We further propose new algorithms realizing two fundamental primitives and improve the efficiency of P2P protocols in the synchronous setting. For future work, we are planning to leverage SGX features to re-design OBLIVP2P to achieve decentralization and better throughput with low latency against byzantine adversaries. Furthermore, we can integrate our new OBLIVP2P into WebRTC-enabled web browsers as primitives for new proposed protocols.

LIST OF TABLES

2.1 Low-latency anonymous communication systems. 13

3.1 Notations for anonymity analysis. 31

3.2 Comparison with low-latency anonymous systems. 61

4.1 Various meta-information contained in the state s, for OBLIVP2P-0 and OBLIVP2P-1. B is the block size in bits, N_P the number of peers, N_B number of blocks, L the path length, and z the bucket size. 79

4.2 Comparison of OBLIVP2P instantiation per access. B the block size, N the number of blocks in the network, E the overhead of a block encryption, $\mathcal{E}$ a multiplication in elliptic curve group, burst the number of versions. 85

5.1 Round complexity and communication complexity for reliable broadcast in synchronous network. 113

5.2 Round / communication complexity for random number generation protocols in synchronous distributed systems. 113

LIST OF FIGURES

2.1 Illustration of content delivery in current peer-assisted CDNs. Peer v_A sends requests to the peer server for resources R_1 and R_2 (in ①). The server responds to v_A with a list of peers, e.g., v_B and v_C having R_1 and R_2 respectively (in ②). v_A connects with v_B/v_C, and fetches resources from them (in ③ ④). 12

3.1 Illustration of an inference attack to infer the responder of a request. The adversarial peer v_A sends a request for a resource R to the peer server (in ①). Based on locality, the server replies v_A with the nearby victim v_B having R (in ②). v_A fetches R from v_B (in ③), thus the adversary infers that the victim has viewed R recently. 21

3.2 Illustration of an inference attack to infer the initiator of a request. The user v_A fetches resources from peer v_B and v_C which are controlled by the adversary (in ① ②). Therefore, by passively observing the requests from the victim, the adversary can determine that the victim is looking for (interested in) the requested content. 21

3.3 The overview for the deployment of APAC. 28

3.4 A 8-node (6-relay) circuit from the initiator to the responder in APAC. 32

3.5 The network latency reduction (based on the loading time without APAC) decreases when increasing the degree of anonymity of the system. 32

3.6 Effect of varying L_{max}: Increasing the maximum number of intermediate nodes increases the degree of initiator anonymity. . . 34

3.7 Effect of varying N: Increasing the total number of joined peers increases the degree of initiator anonymity. 34

3.8 Effect of varying f_R: Increasing the number of adversarial peers having the requested resource decreases the degree of initiator anonymity. 35

3.9 Overview of the communication channels in APAC. 38

3.10 Overview of how a peer fetches content from another peer via three nodes in APAC. 40

3.11 Total outgoing network traffic size of a server in response to a request in APAC setting ($APAC$) and in the client-server setting ($BASELINE$), measured in KB. 43

3.12 When the number of requests increases, the total outgoing network traffic for $BASELINE$ significantly increases, but the traffic for $APAC$ only slowly increases. 45

3.13 Network latency reduction (NLR) of an initiator peer in Swarmify and APAC under three network configurations, as compared to the baseline. All data are based on a resource with size of 2 MB and averaged over 30 runs with 95% confidence intervals for each of them. 47

3.14 The percentage of the network latency for setup overhead of a circuit in three configurations. 48

3.15 Network latency reduction (NLR) varies when intermediate nodes are in different regions. "a-b-c" means the number of nodes in the initiator's region ($a = \lfloor \alpha_{init} l \rfloor$), nodes randomly chosen ($b = l - \lfloor \alpha_{init} l \rfloor - \lfloor \alpha_{res} l \rfloor$) and nodes in the responder's region ($c = \lfloor \alpha_{res} l \rfloor$). 48

3.16 The network latency reduction (NLR) for 50 requests(sorted in ascending order) when 100 joined peers are in APAC. 49

3.17 The success rate decreases when the length of the circuit increases. 50

3.18 The success rate increases when the stay time follows the Weibull distribution with larger λ and smaller k. (The circuit has 2 intermediate nodes.) 51

3.19 λ becomes larger and k turns smaller, when more users stay longer on the page. 57

3.20 The stay rate decreases when the duration of a circuit and k increase, as well as λ decreases. 57

3.21 The success rate increases when the number of created circuits increases. 58

4.1 Mapping of a client / server ORAM model to a P2P system . . . 74

4.2 Oblivious Selection protocol using IT-PIR and Seed Homomorphic PRG as base primitives 80

4.3 Theoretical (Th) and experimental (Ex) comparison of OBLIVP2P-0 and OBLIVP2P-1 parameters for block size of 512 KB. The throughput for OBLIVP2P-1 linearly scales with the increase in network size. 93

4.4 The latency for fetching a block for OBLIVP2P-1 reduces up to 2^{13} and then becomes constant. 93

4.5 The latency for sync operation for OBLIVP2P-1 reduces up to 2^{13} and then becomes constant. 94

4.6 The data transferred through the tracker for OBLIVP2P-0 increases linearly with the number of peers 94

4.7 The throughput for blocksize 128 KB and 1 MB increases with increase in the network size. 95

4.8 Impact of optimizations (O1-O3) on the throughput of OBLIVP2P-1 for 2^{14} peers and blocksize of 512 KB. 95

5.1 Each peer consists of two entities: an Enclave and an OS. The OS models the operating system and memory. The Enclave models the isolated memory and the secure execution of a program. The sender $\mathsf{Enclave}_s$ can send a message via a secure channel to the receiver $\mathsf{Enclave}_r$. The grey areas are secure against malicious OSes of byzantine nodes. 121

5.2 Termination time in seconds for ERB slightly increase with the number of peers. 138

5.3 (Th) theoretical and (Ex) experimental comparisons of network overall communication bandwidth in MB for ERB in function of the number of nodes in $\mathcal{P}$. 138

5.4 Termination time of ERNG in function of the number of nodes in $\mathcal{P}$. 139

5.5 Communication overhead of ERNG in function of the number of nodes in $\mathcal{P}$. 140

5.6 Time termination of ERB linearly increase with the number of byzantine nodes in $\mathcal{P}$. 141

5.7 Communication overhead of ERB in function of different byzantine peers in $\mathcal{P}$. 141

5.8 $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$: SGX-based Peer channel. 149

Chapter 1

Introduction

The World Wide Web (or the web) provides assorted web resources and services for users to conduct essential activities online, such as reading news, paying bills and accessing medical records. In our digital era, the web has become an integral part of everyone's daily life. The traditional web architecture adopts the client-server model, where a web browser sends requests and retrieves web contents from a web server. The server is responsible for hosting all the web resources and delivering relevant resources to clients for every request.

This way, the server becomes the centralized bottleneck, which is a major factor influencing the efficiency of fetching resources for clients. To address such challenge, conventional CDN operators, e.g., Akamai [1] and CloudFlare [11], distribute web contents around the world-wide servers (dubbed *edge* servers) for faster loading of content resources. CDN operators can increase the geo-density of edge servers up to an extent (e.g., placing edge servers in major cities of various countries), but fetching resources for users far away from an edge server has major latency. In the meantime, owing to the tremendous cost of renting CDN servers, small companies cannot afford a large-scale deployment (say worldwide) of servers. To resolve these issues, various approaches have been proposed utilizing proxies or client-side software as CDN servers to deliver resources to clients [2, 33, 37, 45, 57, 62, 130, 134, 135, 149, 218, 222, 228, 231].

Previous studies have shown advantages of such peer-assisted CDNs systems: they can significantly reduce the cost of all parties including Internet Service Providers (ISPs) [147, 156]. For example, NetSession has over 25 million users in 239 countries and territories, and offloads $70-80\%$ of the traffic to peers without the trade-off of reliability [232].

With the recent advances of peer-to-peer (P2P) techniques (supported by real-time communications [59]) in web browsers, browsers can also serve as CDN servers to deliver the fetched contents to other browsers, which helps to offload the burden of servers and improve the efficiency of data transfer. These P2P techniques empower browsers to directly communicate with each other for data transmission, such as voice / video streaming. As a result, browsers act both as clients to send requests for resources and as servers to provide web contents. Users can retrieve web contents from multiple browsers simultaneously, instead of the centralized server in the traditional model. This evolves the web from the client-server model towards a new peer-to-peer model, which decentralizes the web to provide better availability of web resources. Following the new trend of pursuing the peer-to-peer (P2P) web, mainstream browsers, such as Google Chrome, Firefox and Internet Explorer, have already supported real-time communication (RTC) [27, 59], which enables one browser to directly communicate with another, forming a *web overlay* consisting of browsers as peers in a P2P network. Moreover, various emerging browsers, including Maelstrom from BitTorrent [28] and Mist from Ethereum [29], are designed to be decentralized and support peer-to-peer communication.

However, web overlays bring not only the benefits of P2P techniques, but also new threats. A web overlay can be abstracted as a P2P network, in which browsers are the nodes connecting to others. Thus, the open problems in P2P systems, such as privacy issues, are introduced into web overlays. One of most critical threats is that due to the open (permissionless) nature of the new web infrastructure, malicious nodes (or browsers) can easily join an overlay by con-

necting to another browser. Joining as legitimate peers in the overlay, malicious nodes can either passively monitor the traffic to infer a benign user's online activities (e.g., requesting particular resources), or behave arbitrarily to disrupt the execution of the given protocol. For instance, in a web overlay or a peer-assisted CDN, a user's online activities such as sending / requesting personalized web pages and relevant resources are exposed to the other peers. Revealing a user's browsing history will significantly leak the user's privacy. For example, a user's digital identity can be revealed when visiting social network websites [226]; visiting map service/political websites reflects a user's geolocation/political orientation [152]. Moreover, long-term traffic analysis through global monitor (i.e., observing all traffic of the whole network) is feasible in real P2P systems. For example, many copyright-enforcement organizations and service providers are reported to globally monitor BitTorrent traffic to identify illegal actions. It has been shown that monitoring of BitTorrent traffic can reveal the data sent and requested by the peers in the network [164, 200, 212]. Therefore, users in various P2P systems have a risk of leaking private information (such as the resources they upload or download) to a global adversary.

In the meantime, the robustness of P2P protocols is the basis of the core utilities and security / privacy properties provided by these protocols. However, the presence of malicious (byzantine) adversaries is a major security concern in P2P systems, as they can perform malicious behaviors, like forging, delaying and dropping messages, to disrupt the protocol execution (robustness). For example, recently, researchers have demonstrated that in a popular cryptocurrency — Bitcoin — byzantine nodes can collude to eclipse or partition the honest nodes leading to double-spending and selfish mining attacks [146, 189]. Further, byzantine nodes in anonymous P2P networks can become the entry and exit nodes of an honest node's communication circuit, by advertising high-bandwidth connections and high-uptimes falsely [70]. These byzantine entry/exit nodes can selectively deny service or severely weaken the core anonymity properties of such

systems as Tor, Cashmere and Hydra-Onions [47, 84].

To preserve privacy, like hiding their online traces, users typically employ anonymous networks to conceal their digital identities. These anonymous systems include Mix network [100,117,183], Onion-routing/Tor-based systems [64, 123, 150, 204, 223], and other P2P anonymous systems [136, 178, 179, 187, 205, 207], which allow a user to be *anonymous*, so that the user is unidentifiable within a set of users [199]. However, these existing applications rarely impose stringent performance demands, such as in a real-time caching system. Their primary goal is to preserve a high level of anonymity. Directly adopting these approaches may introduce non-negligible performance overhead for clients, e.g., the client-side communication overhead for circuit setup in onion routing-based and Crowds-based systems. On the other side, these anonymous systems are susceptible to long-term traffic analysis, such as intersection or statistical disclosure attacks [63, 159] and end-to-end correlation attacks [49, 51]. Furthermore, these systems only consider passive adversaries but not malicious (or byzantine) nodes, which can arbitrarily forge, delay, replay and drop messages to violate the given protocols and weaken the privacy guarantee [47, 84]. Therefore, designing robust P2P protocols continues to be an important research problem due to the attacks possible in a byzantine setting.

Thesis Statement. *The emerging web architecture that is based on web overlays results in new privacy and robustness issues. In this thesis, we design new privacy-preserving P2P systems and robust P2P protocols using cryptographic / hardware primitives to resolve these issues.*

Specifically, in this thesis, we aim to answer the following research questions: What are the real-world attacks on privacy in web overlays? How to preserve adequate privacy as well as balance performance overhead in web overlays? Can we design privacy-preserving P2P systems against honest-but-curious adversaries with global views? Further, can we devise robust P2P protocols against byzantine adversaries?

Our Approach. We first employ onion-routing techniques by adding intermediate hops in between browsers (users) to enable a user to be unidentifiable among a set of users [150]. With preserving anonymity of users, we also propose region-based circuit selection algorithm to achieve desired performance gains. Second, to hide access patterns of users (or links between users and resources) against long-term traffic analysis of global adversaries, which the first system APAC does not provide, we propose OBLIVP2P– a construction for a scalable distributed ORAM protocol in the P2P setting [153]. Lastly, in contrast to dealing with passive adversaries, we ensure the robustness of P2P protocols against active adversaries (or byzantine nodes), which are not handled by APAC and OBLIVP2P. We leverage a recent trusted computing mechanism, called Intel SGX, to reduce the byzantine model into the general-omission model, in which faulty nodes can only omit to send / receive messages [154]. We show two examples of fundamental P2P protocols are realizable securely using SGX features. Since our approaches are generic and these issues are common in conventional P2P systems, our solutions can also be smoothly ported to P2P systems.

1.1 Thesis Overview

Next, we present the overview of the works constituting this thesis.

1.1.1 APAC: An Anonymous Peer-assisted CDN

As the first step, we systematically study the representative applications of web overlays called peer-assisted CDNs. We present that the current designs of peer-assisted CDNs expose clients to privacy-invasive attacks, enabling one client to infer the set of browsed resources of another client. To resolve this privacy issue, we propose an *anonymous peer-assisted CDN (*APAC*)*, which employs P2P content delivery while providing *initiator anonymity* (i.e., hiding who sends

the resource request) and *responder anonymity* (i.e., hiding who responds to the request) for peers. Previous anonymous systems [18, 22, 48, 117, 123, 136, 179, 183, 204, 205] focus on preserving high level of anonymity, but rarely impose stringent performance demands, such as in a real-time caching system. In contrast to existing works, our *locality-aware* APAC is tunable to select intermediate nodes nearby the initiator/responder, which reduces the resource fetching time (network latency) for peers. APAC can be a web service, compatible with current browsers and requiring no client-side changes. Our anonymity analysis shows that our APAC design can preserve a higher level of anonymity than state-of-the-art peer-assisted CDNs. In addition, our evaluation demonstrates that APAC can achieve desired performance gains.

1.1.2 OBLIVP2P: An Oblivious P2P Content Sharing System

In APAC, the adversary is considered to be non-global, which cannot monitor all the traffic in the network. However, a global adversary is feasible in real P2P systems, and previous anonymous systems are susceptible to long-term global analysis, such as intersection attacks [63, 159] and end-to-end correlation attacks [49,51]. To overcome these challenges, we propose a new approach to protecting against persistent, global traffic analysis in P2P content-sharing systems. Our approach advocates for hiding data access patterns, making P2P systems *oblivious*. We propose OBLIVP2P— a construction for a scalable distributed ORAM protocol, usable in a real P2P setting. Our protocol achieves the following results. First, we show that our construction retains the (linear) scalability of the original P2P network w.r.t the number of peers. Second, the experiments simulating about 16,384 peers on 15 Deterlab nodes can process up to 7 requests of 512KB each per second, suggesting usability in moderately latency-sensitive applications as-is. The bottlenecks remaining are purely computational (not bandwidth). Third, the experiments confirm that in our construction, no centralized infrastructure is a bottleneck — essentially, ensuring that the

network and computational overheads can be completely offloaded to the P2P network. Finally, the construction is highly parallelizable, which implies that remaining computational bottlenecks can be drastically reduced if OBLIVP2P is deployed on a network with many real machines.

1.1.3 Robust Synchronous P2P Primitives Using SGX Enclaves

In APAC and OBLIVP2P, we consider that an adversary can only passively monitor the traffic in the network, but in real scenarios, malicious (byzantine) nodes can easily join the network and perform arbitrarily, like disrupting the execution of the given protocol. Peer-to-peer (P2P) systems such as BitTorrent, Bitcoin, APAC and OBLIVP2P, are susceptible to serious attacks from byzantine nodes that join as peers. Due to well-known impossibility results for designing P2P primitives in unrestricted byzantine settings, research has explored many adversarial models with additional assumptions, ranging from mild (such as pre-established PKI) to strong (such as the existence of common random coins). One such widely-studied model is the *general-omission* model, which yields simple protocols with good efficiency, but has been considered impractical or unrealizable since it artificially limits the adversary only to omitting messages.

In this work, we study the setting of a synchronous network wherein peer nodes have CPUs equipped with a recent trusted computing mechanism called Intel SGX. In this model, we observe that the byzantine adversary reduces to the adversary in the general-omission model. As a first result, we show that by leveraging SGX features, we eliminate any source of advantage for a byzantine adversary beyond that gained by omitting messages, making the general-omission model *realizable*. Second, we present new protocols that improve the communication complexity of two fundamental primitives — reliable broadcast and common random coins (or beacons) — over the best-known results in the synchronous general-omission model, by utilizing SGX features. Specifically,

we present a reliable broadcast protocol that achieves $O(N^2)$ communication complexity in bits and worst-case $\frac{N}{2} + 2$ round complexity while tolerating up to $\frac{N}{2}$ byzantine nodes. As another example, we show a protocol to generate a distributed common random coin that can be used as a random beacon. Our protocol that generates an *unbiased* random number in presence of $\frac{N}{3}$ byzantine peers having worst-case round and communication complexity of $O(\log N)$ and $O(N \log N)$. We present proofs of correctness and security and confirm theoretical efficiency claims with experimental evaluation on over 1000 nodes, running on 40 machines on DeterLab. We discuss the application of both these protocols in other P2P operations, such as random walks, shared key generation, random beacons and load balancing protocols.

Summary of Contributions:

- We systematically analyze inference attacks on real-world web overlays or peer-assisted CDNs, i.e., Swarmify, BemTV and P2PSP. We develop an anonymous peer-assisted CDN (APAC) for web applications, which involves browsers as peers to distribute content. Our prototype implementation is available online [4]. From our analysis, APAC can preserve high level of initiator/responder anonymity even if 35% peers are compromised. Our evaluation demonstrates that APAC can bring desired network latency reduction for peers and bandwidth savings for deployed sites.
- We propose OBLIVP2P— a first candidate for an oblivious peer-to-peer protocol in content sharing systems against long-term traffic analysis of global adversaries. Our main building block is a primitive which we refer to as oblivious selection that makes a novel use of recent advances in Oblivious RAM combined with private information retrieval techniques. Our prototype implementation is available online [32]. We experimentally evaluate our protocol to measure the overall throughput of our system, latency for accessing resources and the impact of optimizations on the system throughput.
- To deal with malicious nodes, we leverage SGX features to reduce the byzan-

tine model to the general-omission model, where byzantine nodes have no extra advantage than omitting messages. By enforcing our properties, we can improve the efficiency of P2P protocols. As the first attempt, we propose efficient protocols for reliable broadcast (ERB) and unbiased random number generation (ERNG) in synchronous settings. We provide security analysis and proof for our protocol constructions. Our prototype implementation is open source and available online [34]. Our experimental evaluation confirms the theoretical expectations of our solutions.

Statement of Joint Work: The development of techniques, protocols and systems presented in Chapter 3 and Chapter 5 was led by Yaoqi Jia. In addition to Yaoqi Jia, contributors to the work presented in Chapter 3 include Guangdong Bai, Prateek Saxena and Zhenkai Liang. Contributors in addition to Yaoqi Jia for the work presented in Chapter 5 include Shruti Tople, Tarik Moataz, Deli Gong, Prateek Saxena and Zhenkai Liang. The development of OBLIVP2P protocol in Chapter 4 was joint work of Yaoqi Jia, Tarik Moataz and Shruti Tople. The development of OBLIVP2P system was led by Yaoqi Jia. The contributors to the work presented in Chapter 4 also include Prateek Saxena.

Organization: We first discuss the background of web overlays as well as related work on privacy-preserving systems and robust P2P protocols in Chapter 2. We present inference attacks against privacy on representatives of web overlays (peer-assisted CDNs), and propose APAC that preserve certain privacy and balance performance overhead in Chapter 3. In Chapter 4, we design OBLIVP2P to introduce ORAM to P2P systems to conceal data access patterns for the first time. In contrast to the honest-but-curious setting, we present using SGX features to devise robust P2P primitives against byzantine adversaries and improve the efficiency of existing protocols in Chapter 5. Finally, we conclude and discuss future work in Chapter 6.

Chapter 2

Background

In this chapter, we first illustrate the background of web overlays. Then we review related work on privacy-preserving systems and robust P2P protocols including anonymous communication systems against partial adversaries, long-term traffic analysis of global adversaries and robust P2P primitives against byzantine adversaries.

2.1 Web Overlays

With the advent of P2P techniques in the web infrastructure, web browsers start to be coordinated by servers to connect with each other. Each browser can act as a client to send requests for web resources, such as HTML and JavaScript, and also behave like a server to deliver these resources to other browsers. In terms of distinct P2P web applications, all the involved browsers and servers form various web overlays, which can be abstracted as P2P networks consisting of nodes (represented by browsers / servers) and edges (represented by connections in between nodes). Web overlays assist servers to offload resource-delivery tasks to browsers, improving the availability of web resources and the efficiency of transferring these resources. With the benefits brought by P2P overlays, browser vendors provide active and substantial support to push forward the development of the P2P-based web infrastructure. Until now, all of the mainstream browsers

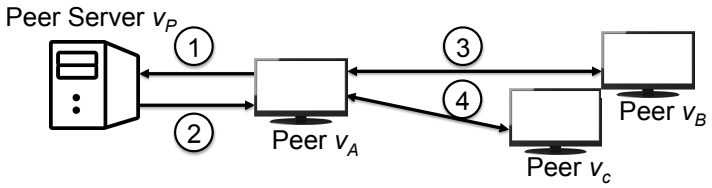


Figure 2.1: Illustration of content delivery in current peer-assisted CDNs. Peer v_A sends requests to the peer server for resources R_1 and R_2 (in ①). The server responds to v_A with a list of peers, e.g., v_B and v_C having R_1 and R_2 respectively (in ②). v_A connects with v_B/v_C, and fetches resources from them (in ③ ④).

such as Chrome and IE except Safari support WebRTC [27, 59], which empowers browsers with P2P real-time communication. In the meantime, assorted emerging browsers are designed to support P2P communication including Maelstrom from BitTorrent [28] and Mist from Ethereum [29].

The representative example of web overlays is the peer-assisted CDN, engaging clients (like browsers) to deliver contents to other peers. Numerous peer-assisted CDNs have been proposed in recent years [2, 37, 45, 62, 130, 134, 135, 149, 218, 222, 228, 231]. In contrast to traditional infrastructure-based CDNs, peer-assisted CDNs offload content delivery tasks on clients (peers) to save the bandwidth of servers [147, 156], and reduce the latency of fetching content at the client side. For instance, NetSession can offload $70 - 80\%$ of the traffic to the peers [232]. For the thorough evaluation on Etsy, Maygh is able to reduce the 95th-percentile bandwidth due to image content at the operator by over 75% [231]. In peer-assisted CDNs, the peers are the clients that fetch and distribute contents, e.g., browsers in Maygh [231] and client programs in NetSession [2]. In such design, a peer server coordinates peers to distribute contents. As Figure 2.1 illustrates, after communicating with the peer server, peers can fetch contents from other nodes. To reduce the network latency, the peer server typically assigns the initiator a peer who has the requested resource as the responder, based on the distance between their IP addresses or geolocations (called a *locality-aware* peer selection algorithm). Alternatively, the server directly responds with a list of peers having the requested content, the initiator

System's Name	No Installation	Initiator Anonymity.	Responder Anonymity.	Locality-aware
Onion Routing/Tor-based Systems [64, 123, 204, 223]	✗	✓	✗	Partially
Crowds [205], & Morphmix [207], etc. [136, 178–181, 187]	✗	✓	✗	✗
Hidden Service [48], I2P [22] & Freenet [18], etc. [53, 71, 148, 209]	✗	✓	✓	✗

Table 2.1: Low-latency anonymous communication systems.

will contact these peers in parallel. After receiving the content, the initiator can serve it to other peers.

Analogous to peer-assisted CDNs, other web overlays have the similar topology, where a coordinator server instructs peers or browsers to communicate with each other, like BitTorrent's tracker model.

2.2 Related Work

2.2.1 Anonymous Communication Systems Against Partial Adversaries

For peer-assisted CDNs, researchers have proposed solutions to preserve the integrity and authenticity of contents [62, 218, 231]. For example, FireCoral introduces *signing service* and *tracker* components to authenticate and verify contents [218]. On the other hand, adversarial nodes in the network can infer a victim node's browsing history, i.e., what resources the victim has requested and served. However, no systematic studies of inference attacks on peer-assisted CDNs have been conducted yet and few defenses against such attacks are deployed on peer-assisted CDNs.

Anonymous Communication Systems. One typical way to protect a user's privacy (like unlinking her identity and her browsing history) is to make her

unidentifiable among a large set of users. To achieve such anonymity, anonymous communication systems are often the primary choice, as they are designed to hide users' identities from third parties. Anonymous communication systems can be classified into high-latency and low-latency systems. High-latency anonymous communication systems, e.g., Mixminion [117] and Mixmaster [183], are designed to be against a global passive adversary who can observe all traffic in the network. However, the high-latency message transmission (e.g., several hours) makes them unsuitable to be implemented in peer-assisted CDNs.

As Table 2.1 shows, there are four design considerations for low-latency communication systems from the system's perspective. *No installation* indicates that an approach does not require the client to install any software. *initiator / responder anonymity* means that the initiator / responder is unidentifiable among an set of peers. A system is *locality-aware* if its algorithm is based on the locations of peers. Onion routing/Tor-based systems [64, 123, 204, 223], semi-centralized systems [85, 86], and P2P low-latency anonymous systems [18, 22, 48, 53, 71, 136, 148, 178–181, 187, 205, 207, 209] provide initiator anonymity. For instance, MorphMix creates paths on an unstructured overlay to forward communications; ShadowWalker is based on a random walk over redundant structured topologies and Pisces uses social networks to achieve anonymous communications. Further, Hidden Service [48], I2P [22], Freenet [18], and other approaches [53, 71, 91, 148, 209] also preserve responder anonymity.

2.2.2 Long-term Traffic Analysis of Global Adversaries

In reality, apart from the partial adversary in the previous section, a global adversary capable of performing long-term analysis in the network is proven to exist and not rare. For instance, globally monitoring of BitTorrent traffic has shown to reveal the data requested and sent by the peers in the network [164, 200, 212]. Nevertheless, anonymous systems, like mix networks and

onion routing, are susceptible to long-term traffic analysis as shown in Section 4.2. Statistical disclosure attacks, proposed by Danezis and enhanced by other researchers, improve the likelihood of de-anonymizing users on these systems [114, 116, 119, 120, 173, 175, 197, 219]. Moreover, existing traffic analysis attacks on onion routing based approaches [127, 128, 191, 224, 225] can reveal users' identities with observing multiple communication rounds. Other P2P systems like Crowds [205], Tarzan [136], MorphMix [207], AP3 [178], Salsa [187], ShadowWalker [179], Freenet [18] offer anonymity for users. However, these systems show limits against global adversary with long-term traffic analysis capabilities.

2.2.3 Robust P2P Primitives Against Byzantine Adversaries

As one of the fundamental P2P primitives, reliable broadcast has been extensively studied in the byzantine setting.

Reliable Broadcast. Reliable broadcast has been extensively studied since the 1980s, and is closely related to the problem of byzantine agreement (BA). Several excellent surveys on the problem are available [160, 220]. Byzantine agreement can also achieve reliable broadcast [88, 92, 94, 155, 176, 185, 201, 220]. For the asynchronous network, Bracha's classic reliable broadcast protocol requires $O(N^2)$ communication complexity and tolerates up to $\frac{N}{3}$ byzantine nodes [89, 90]. Cachin and Tessaro [95] leverage erasure codes to improve efficiency and reduce communication complexity. However, as the time is not bounded, messages may incur arbitrary delays, and most protocols do not guarantee terminating runs, except under some special assumptions such as sharing a "common coin" [88, 201].

Without any extra assumptions, reliable broadcast and byzantine agreement in the synchronous setting can tolerate $\frac{N}{3}$ byzantine nodes at most, and with $\min\{f+2, t+1\}$ round complexity [124, 162, 195]. Lamport *et al.* and Pease *et al.* propose protocols terminating within $t+1$ rounds and tolerating up to

$\frac{N}{3}$ byzantine nodes, but with exponential communication complexity [162, 195]. Berman *et al.* achieve $O(\mathsf{poly}(N))$ communication complexity but only tolerating upto $\frac{N}{4}$ byzantine nodes [76]. Garay *et al.* later present a byzantine agreement protocol terminating within $\mathsf{min}\{f + 5, t + 1\}$ rounds [138, 139].

To tolerate a larger fraction of byzantine nodes, additional assumptions are often needed. A common assumption is that of having a one-time trusted dealer that pre-deploys PKI in the infrastructure. This assumption, for instance, allows digital signatures to be used for *authentication*, wherein a message claimed to be sent by a node A can be assured to be originating from A [125, 133, 157, 162]. This weakens the capabilities of the byzantine adversary, which cannot forge messages on behalf of honest nodes. Researchers have proposed protocols to use digital signatures to boost the resilience from $\frac{N}{3}$ to $N - 1$, but the communication complexity is still large, i.e., $O(\mathsf{exp}(N))$ and $O(N^3)$ [125, 162]. Katz *et al.* extend the work of Feldman and Micali [132] to employ authenticated channels, and present protocols tolerating $\frac{N}{2}$ byzantine nodes with $O(\mathsf{poly}(N))$ complexity [157]. Fitzi *et al.* also give an authenticated BA protocol that beats this bound ($\frac{N}{2}$) but under specific number-theoretic assumptions [133]. Abraham *et al.* provide a solution with early stopping ($\mathsf{min}\{f + 2, t + 1\}$) and polynomial complexity [61].

Chapter 3

Anonymity in Peer-assisted CDNs: Inference Attacks and Mitigation

3.1 Introduction

Web overlays are used for various systems, and peer-assisted CDNs are one of the representative applications. The peer-assisted CDN is a new content distribution paradigm supported by CDNs (e.g., Akamai), which enables clients to cache and distribute web content on behalf of a website. Content Delivery Networks (CDNs) were introduced a decade ago. To distribute data to end users in a fast way, CDN operators (e.g., Akamai [1] and CloudFlare [11]) assist sites to deliver content to end users with their multiple data centers across the world. Complementary to these infrastructure-based approaches, numerous CDNs have adopted peer-to-peer techniques to distribute content, e.g., Swarmify [45], PeerCDN [37] Akamai NetSession [2], Squirrel [149], CoralCDN [134, 135], FlowerCDN [130] and Maygh [231].

On one hand, involving end users as peers (client-side CDNs) to distribute data in peer-assisted CDNs reduces the fetching time of resources and saves the bandwidth of CDNs' servers. For instance, Akamai NetSession can offload over 70% of the traffic to peers with high reliability [232]. On the other hand, in

contrast to infrastructure-based systems in which trusted centralized servers are deployed, untrusted peers in peer-assisted CDNs can easily join the system. The compromised or malicious peers can then a) modify content and inject unauthorized content; b) delay or deny content delivery to other peers; c) misreport their contributions to manipulate the accounting for commercial services [62, 232]; d) infer which peer they deliver/fetch a resource to/from, and what content the peers have requested.

In practice, peer-assisted CDNs introduce various measures to tackle these security issues. For example, to protect the authenticity and integrity of the content in peers, FireCoral introduces the *signing service* to authenticate content, and peers can verify the content with the hash information supplied by the *tracker* [218]. In addition, Aditya *et al.* proposed RCA, a reliable client accounting system for NetSession to discover all misreportings and protocol violations by faulty or malicious clients and quarantine these potentially colluding clients [62]. However, privacy leakage in peer-assisted CDNs has not been actively studied yet, and anonymity is also seldom considered by existing peer-assisted CDNs.

Although the majority of resources (e.g., images) served on CDNs are publicly accessible for all users, a user's online activities (e.g., visiting personalized web pages and fetching relevant resources) have been proven to be privacy-sensitive. For example, a user's digital identity can be revealed when visiting social network websites [226]; visiting map service/political websites reflects a user's geolocation/political orientation [152]. Revealing a user's browsing history will significantly leak the user's privacy. To demonstrate this threat, we present *inference attacks* on peer-assisted CDNs: by placing controlled peers and observing the requested contents in the system, the adversary can effectively infer content-access activities of benign peers.

To demonstrate the effectiveness of the proposed inference attacks, we conduct inference attacks against widely used peer-assisted CDNs, including Swarmify,

BemTV and P2PSP. From our experiments, we find that in any of these systems, when a peer (i.e., an initiator) sends a request for a specific resource, the server assists the initiator to directly fetch the content from another nearby peer (i.e., the responder) based on locality, without any mechanism to conceal the initiator's or the responder's identity. Therefore, the adversary who controls a number of peers is capable of mounting inference attacks on existing peer-assisted CDNs to identify the initiator or responder of a forwarded request. Through the observed communication, the adversary can infer the user's browsing history and preferences. The adversary can then use this information for spear phishing, personal targeted advertisements, and social engineering attacks [163].

The primary goal of peer-assisted CDNs is to save the server's bandwidth and reduce client-side network latency. It is quite challenging to conceal peers' identities (IP addresses) to mitigate inference attacks as well as to preserve reasonable responsiveness for the deployed site. On the other hand, although anonymous mechanisms have been well studied in communication systems [18, 22, 48, 117, 123, 136, 179, 183, 204, 205], their previous applications rarely impose stringent performance demands such as in a real-time caching system. Their primary goal (e.g., onion routing [204], Tor [123], Mixminion [117] and Crowds [205]) is to preserve high level of anonymity. Directly adopting these approaches may introduce non-negligible performance overhead for clients, e.g., the client-side communication overhead for circuit setup in onion routing-based and Crowds-based systems. For instance, Mixminion can introduce several-hour latency for message transmission [117].

To address these challenges, we develop an *Anonymous Peer-assisted CDN (*APAC*)* for web applications. APAC involves client-side browsers as peers to distribute content without requiring any software installation for users. Once users visit the deployed site, APAC's client-side code will assist their browsers to join APAC as peers and serve resources to other users. APAC is compatible with mainstream browsers and requires negligible modification on the deployed

websites. In practice, the adversarial peers can exist uniformly in the system or densely surround the victim. To achieve an adequate level of anonymity in different scenarios, we introduce a new *region-based circuit selection algorithm* to construct a path (denoted as *circuit*) consisting of peers. Our APAC encapsulates the request with standard layered encryption and the initiator can fetch the content via the constructed circuit.

Our goal is not to build a perfect anonymity-preserving system, as it may introduce huge performance overhead, but we aim to enable the system to make trade-off between performance and anonymity, which is significantly beyond that of current peer-assisted CDNs. Using a standard measure of *degree of anonymity* [122], we show that APAC can preserve a high degree of initiator/responder anonymity (i.e., at 0.8 degree of anonymity recommended for anonymous communication systems by Diaz *et al.* [122]), with only two intermediate peers for a circuit even if 35% of all peers are under the adversary's control. Our *locality-aware* APAC is tunable to select intermediate nodes nearby the initiator/responder, which reduces the resource fetching time (network latency) for peers. Our evaluation in a city-wide network shows that with two intermediate peers for a circuit, APAC can reduce 44.1% client-side network latency and save 97.3% server-side bandwidth when fetching 2 MB[1] resources via peers, with 0.8 degree of anonymity.

Contributions. Compared to regular CDN services, APAC saves the bandwidth of the CDNs edge server, and reduces the latency for peer when the edge server and peers are not in the same city. In contrast to single-hop peer-assisted CDNs such as Swarmify, APAC provides an adequate level of anonymity for peers, but it trades off the performance, e.g., network latency. In summary, we provide the following contributions:

- We systematically analyze inference attacks on real-world services, i.e., Swarmify, BemTV and P2PSP.

[1] Since the averaged total size of transferred data when loading a site is 2268 KB [50], we use the fetching time of 2 MB resources as the representative.

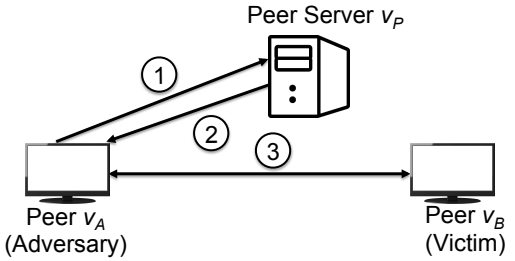


Figure 3.1: Illustration of an inference attack to infer the responder of a request. The adversarial peer v_A sends a request for a resource R to the peer server (in ①). Based on locality, the server replies v_A with the nearby victim v_B having R (in ②). v_A fetches R from v_B (in ③), thus the adversary infers that the victim has viewed R recently.

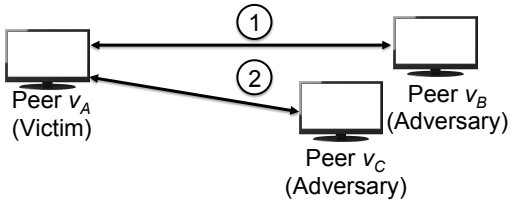


Figure 3.2: Illustration of an inference attack to infer the initiator of a request. The user v_A fetches resources from peer v_B and v_C which are controlled by the adversary (in ① ②). Therefore, by passively observing the requests from the victim, the adversary can determine that the victim is looking for (interested in) the requested content.

- We develop an anonymous peer-assisted CDN (APAC) for web applications, which involves browsers as peers to distribute content. Our prototype implementation is available online [4]. From our analysis, APAC can preserve high level of initiator/responder anonymity even if 35% peers are compromised.
- APAC is compatible with current browsers, and requires no client-side installation. Our evaluation demonstrates that APAC can bring desired network latency reduction for peers and bandwidth savings for deployed sites. APAC can customize and balance between three considerations: anonymity, performance and compatibility with browsers.

3.2 Motivation & Problem Statement

3.2.1 Inference Attacks & Real-world Examples

As shown in Figure 2.1, a typical peer-assisted CDN does not conceal the initiator's/responder's identity (IP address) for each request. By observing the forwarded requests from the controlled peers in the system, the adversary can effectively infer the initiator/responder of each request, and further infer the victim's browsing history and preferences (as shown in Figure 3.1 and 3.2). We term such attacks as *inference attacks*. In an inference attack, when any of the adversarial peers is the responder/initiator of a request, the adversary can definitely determine which peer is the initiator/responder of the request. By profiling a user's browsing history and preferences with the inference attack, the adversary can infer the victim's digital identity [226] and precise geolocation [152], as well as further abuse the sensitive information for spear phishing, personally targeted advertisements, or even social engineering attacks [163].

Inference attacks in peer-assisted CDNs have not been carefully studied. We analyze inference attacks on three real-world services, including Swarmify, BemTV and P2PSP, to show the prevalence and effectiveness of such attacks[2].

Swarmify. Swarmify [45] assists the deployed site to deliver content to users from other peers based on locality. It requires sites to include a service-specific library based on WebRTC [59] and deploy optional changes on resources. Although all communications between peers in Swarmify are encrypted, our study demonstrates that it does not guarantee anonymity for peers.

We have mounted an inference attack as follows. We first deployed Swarmify on a website and set 10 images and 2 videos as the targeted resources. For the first time, we launched a Chrome browser as the victim's peer. The victim's peer randomly fetched part of resources (unknown to the attacker) from the remote

[2] Previous work [231, 232] mentions that peers can learn the IP addresses of the connect peers in numerous peer-assisted CDNs, e.g., NetSession, FireCoral, FlowerCDN and Maygh. Thus, these services are conceptually vulnerable to inference attacks too.

server. To infer what content the victim has requested, we located the adversarial node nearby the victim's peer in the same local area network (LAN), and used Wireshark [58] to eavesdrop on the network traffic from/to the controlled node. When the malicious peer requested all resources, we clearly observed that the peer server replied with the victim's IP address and instructed the malicious node to fetch particular resources from the victim's peer. Therefore, the attacker can infer the responder's identity (the victim's IP address) and what content the responder holds. For the second time, the adversarial peer first fetched and buffered all content from the site. When the victim's peer requested several resources, the controlled peer was instructed to distribute the particular resources to the victim's peer. Observing the traffic from the controlled peer, the adversary identified the victim's IP address and the requested resources. Hence we conclude that Swarmify is vulnerable to inference attacks.

BemTV & P2PSP. BemTV [5] is a hybrid CDN and P2P infrastructure for streaming live videos over HTTP, which is also built upon WebRTC with the aim of utilizing clients' web browsers to relay the streamed media files. BemTV requires additional setup on the server side to manage connections between peers, but this is completely transparent to peers.

We have tested BemTV to live-stream a sample video that had been accessed by several computers located within a university network infrastructure. While streaming out the media files, we observed the incoming and outgoing traffic of requests from a computer that acted as an adversary. For each media file, the attacker was able to figure out which specific users (determined by the mapping of IP address) fetched/delivered the media content from/to the adversarial peer. Based on this information, the adversary is capable of determining who is the initiator/responder and further infer the exact video that the victim is watching. In addition, it turns out that BemTV does not guarantee the content integrity between peers in the network, which opens up possibilities for content pollution attacks [137, 151, 186]. Besides BemTV, we have also carried out inference

attacks on another video streaming service called P2PSP [35]. We find that P2PSP suffers the same anonymity issues as BemTV does.

Key findings. Our analysis (in Section 3.6.2) reveals that existing peer-assisted CDNs (including Swarmify, BemTV and P2PSP) cannot preserve an adequate level of anonymity when over 20% peers in the network are malicious. Our analysis is conservative, as we assume that the peer-assisted CDN's server *randomly* assigns a peer having the requested resource as the responder for the request. In reality, if the CDN is locality-aware and instructs the peers to fetch content from nearby peers, it is even easier for the adversary to identify the peers near her controlled peers. Hence the existing peer-assisted CDNs provide much worse anonymity for users than that in our conservative analysis.

3.2.2 Problem Statement

In this chapter, for a particular message (i.e., resource request), we call the peer who initiates the request the *initiator*, and call the peer who responds the request the *responder*. We define *initiator/responder anonymity* to mean that the adversary cannot identify the initiator/responder among peers for a resource request. The assumptions on the adversary are:

- *Internal*: The adversary controls some of peers, which are part of the system and can observe the information about forwarded packets.
- *Partial*: The adversary controls a limited number of peers (e.g., a fraction f), and cannot perform any traffic analysis on the rest of the system.
- *Non-adaptive*: The adversary places nodes first, and then the system constructs a circuit for the request. Once the request is in progress, the adversary cannot alter the placement of peers.

We consider an honest-but-curious adversary, which follows the protocol and places nodes randomly in the network or nearby a victim to increase their chance to determine whether the victim initiates/responds a request. We assume that existing accounting mechanisms [62] can be deployed to discover protocol-

violating peers, e.g., massively sending or delaying/denying content delivery requests [62]. Sybil attacks [118,126,229] and denial-of-service attacks (DOS) [196] are out of scope in this chapter.

The adversary's goal is to identify the initiator/responder of a request with high probability. At the beginning of the request, any benign peer is indistinguishable as the initiator/responder. When the request is in progress, the adversarial peers may be chosen as hops for the request. Hence the adversary can make some observations of the request and gain more knowledge to infer the initiator/responder with higher probability. An anonymous system is required to guarantee that the adversary's observations give minimal *advantage* to determine the initiator/responder. Such an adversary's advantage can be quantified as an entropy metric — the amount of uncertainty in determining the initiator/responder of a request to be a specific victim node. Considering the initiator anonymity, we define the system's entropy as:

Definition 3.2.1. Given a request in the system, where Ψ is the set of peers, and p_u is the probability that the peer u is the initiator of the request, the entropy $H(I)$ for the system is defined by:

$$H(I) = -\sum_{u \in \Psi} p_u \log_2 (p_u) \tag{3.1}$$

If the adversary has no a priori information on the request, the system preserves the maximum entropy H_M. With some observations, e.g., the responder is adversarial and monitors the forwarded packets, the adversary has higher probability to infer the initiator, as well as the system's entropy decreases. Let O be the set of all observations for the adversary. When an observation $o \in O$ occurs with the probability $P(o)$, the corresponding entropy is $H(I|o)$.

Definition 3.2.2. The conditional entropy of the system on observing O is defined by:

$$H(I|O) = \sum_{o \in O} P(o) \cdot H(I|o) \tag{3.2}$$

The advantage gained by the adversary with observation O is the difference in the entropy before and after O, that is: $H_M - H(I|O)$. The degree of anonymity is defined as the normalized value of this difference in certainty of the adversary's guesses about a victim being the initiator:

Definition 3.2.3. The *degree of initiator anonymity* provided by a system is defined by:

$$D(I|O) = 1 - \frac{H_M - H(I|O)}{H_M} = \frac{H(I|O)}{H_M} \tag{3.3}$$

The larger $D(I|O)$ is, the higher level of anonymity a system provides. $D(I|O) = 0$ or $D(R|O) = 0$ means absolutely no anonymity, i.e., the adversary knows with 100% the initiator or responder of a request; When the initiator or responder is not identifiable among all peers, $D(I|O) = D(R|O) = 1$. To preserve an adequate level of anonymity, the degree of anonymity for a system is at least ϵ ($\epsilon = min\{D(I|O)\}$). In this chapter, we set ϵ as 0.8, which is suggested by a previous study [122]. Hence, if $D(I|O)$ and $D(R|O)$ in a system are over 0.8, we consider the system preserve an adequate level of initiator/responder anonymity. Analogous to $H(I|O)$ and $D(I|O)$, we define $H(R|O)$ and $D(R|O)$ to quantify the responder anonymity in a system.

3.3 Anonymous Peer-assisted CDN

In this section, we present our anonymous CDN system APAC that provides protections against inference attacks as well as balances the performance overhead. To build practical anonymous peer-assisted CDNs, we have three goals below.

Anonymity: In a peer-assisted CDN, the adversary may control a fraction f of peers. The adversarial peers can be uniformly scattered in the network or densely surround the victim. When a benign peer sends/responds a request to another peer to fetch/deliver a resource, our system should conceal its identity and the linkability to the requested resource. Therefore, other peers do not know

who is the initiator/responder of the request.

Performance: Peer-assisted CDNs are designed to assist customers (e.g., sites) to save bandwidth of servers and reduce latency of delivering content to users. Our system should not sacrifice this merit. Therefore, one important goal for us is to balance performance and anonymity.

Compatibility: Our design should introduce no (or minor) changes on websites and clients, such that our system can be easily deployed on various web applications and is user-friendly. Compared with other peer-assisted CDNs (e.g., NetSession) that require the end users to install standalone software, our system can attract more users to join as peers for content delivery.

3.3.1 Design of APAC

Overview. APAC consists of two primary components: a peer server run by the site operator, and the client-side code implemented in JavaScript and executed in each peer, i.e., a user's web browser. Independent of the site's content server (or the CDN's edge server), the peer server does not directly deliver content, but maintains the connections with peers and coordinates peers to fetch data either from the content server or other peers. At the beginning, when peers request resources, the peer server instructs them to retrieve the content from the content server. The peers who have retrieved the content will store it and be ready to distribute it. When the total number of joined peers and the number of peers having resources are sufficient to preserve the required anonymity based on proper configurations, the peer server will construct circuits for new requests. The server chooses nodes based on our selection algorithm, and arranges them into a path (or circuit), through which the request and content will be transmitted. Then the initiators of requests fetch content from other peers via the circuits. After receiving the content from other peers, peers first verify its integrity, then store and serve it to other peers. We detail the functionalities and implementation in Section 3.4. As Figure 3.3 illustrates, analogous to current peer-assisted CDNs,

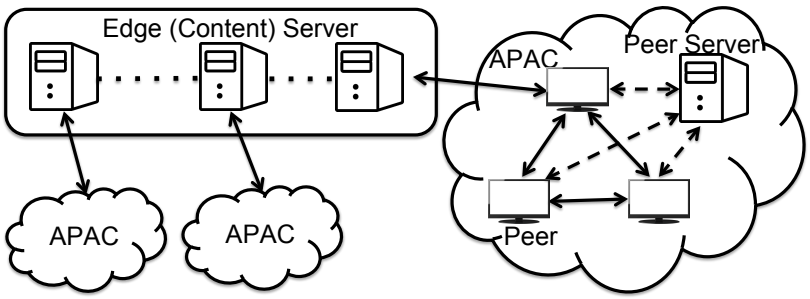


Figure 3.3: The overview for the deployment of APAC.

APAC can be deployed for different edge (content) servers, and serves users far from the nearest edge server. We show the performance of APAC in a city-wide scale in Section 3.5. Next, we demonstrate how APAC achieves the three design goals.

Anonymity: We introduce a new *region-based circuit selection algorithm* that APAC's peer server uses to construct circuits for requests. A circuit consists of three categories of nodes: nodes nearby the initiator and the responder, and nodes chosen globally (not included in the first two categories). To provide initiator/responder anonymity, APAC's peer server communicates with peers and constructs a circuit for each requests, instead of letting the initiator to construct the circuit in conventional onion routing-based approaches, which only preserve initiator anonymity. Each request in APAC is encrypted as a layered encryption packet with the keys of selected nodes for the circuit in the peer server, and is transferred through the circuit. In this way, the initiator and responder know what resource is requested, but cannot identify each other. Any intermediate node only knows its predecessor and successor, but does not know the transmitted content, or determine which peer is the initiator or responder. In contrast to the current peer-assisted CDNs, in which controlling one node in a request is enough to identify the initiator/responder, it is difficult for the adversary to infer the initiator when only controlling the responder or several intermediate peers. To preserve higher level of anonymity, APAC can set a longer length for the circuit. APAC can also be adjusted to select all intermediate nodes globally to avoid choosing disproportionate number of adversarial peers.

For a given circuit, by controlling the first relay, the responder and at least half of the intermediate peers alternatively in the circuit, the adversary can determine the initiator [230]. We show that an adversarial placement of peers (e.g., placing more peers nearby the victim) does not give significant advantage over a randomized placement by an honest-but-curious adversary (details in Section 3.6.1). Even in the analysis of the worst case that the adversary can learn the distance between any two controlled peers, our system can still preserve an adequate level of anonymity with proper configurations (details in Section 3.3.4.1).

Performance: We design APAC to be *locality-aware* to achieve good performance. With APAC's peer server, we can avoid the non-negligible overhead for the circuit setup on the client. Furthermore, based on the locality information of all peers maintained by the server, we utilize the *region-based circuit selection algorithm* to balance the anonymity and performance. Instead of randomly choosing intermediate peers, our algorithm can reduce the network latency by selecting peers nearby the initiator/responder. By adjusting the maximum length of the circuit and the distribution factors (controlling the number of intermediate nodes in each region), APAC can provide significant network latency reduction and bandwidth savings as well as preserve a high degree of anonymity. We discuss the design of circuit construction in the next section, and evaluate the performance in Section 3.5.

Compatibility: We provide a web overlay with WebRTC [59] supported by mainstream browsers (e.g., Firefox and Chrome) to achieve the peer-to-peer communication and data transmission among different peers (browsers). Thus when visiting a deployed website as usual, the user's browser will automatically join APAC as peers to fetch/distribute content from/to other peers in a transparent manner. Meanwhile, APAC's client-side code only requires the retrofitted website to specify the targeted resources. No specification means all static resources on the page can be distributed via peers. We demonstrate the details in

Algorithm 1: Region-based Circuit Selection Algorithm in APAC

input : $N-$ number of peers, $v_{init}-$ initiator, $v_{res}-$ responder, $R-$ requested resource, $L_{max}-$ the maximum number of intermediate nodes (or relays), $\alpha_{init},\ \alpha_{res}(0 \leq \alpha_{init} + \alpha_{res} \leq 1)-$ distribution factors, $N_{init}-$ number of peers nearest to v_{init}, $N_{res}-$ number of peers nearest to v_{res}, $\mathbb{S}_R-$ set of peers having R, $N_R = |\mathbb{S}_R|$, $G-$ topology graph of peers

output: $cir-$selected circuit

$v_{res} \leftarrow randomSelect(\mathbb{S}_R)$
$l \leftarrow random(1, L_{max})$
$cir_{init} \leftarrow \langle\rangle$; $cir_{im} \leftarrow \langle\rangle$; $cir_{res} \leftarrow \langle\rangle$
if $\lfloor \alpha_{init} l \rfloor \geq 1$ **then**
    $G' \leftarrow removeFrom(G, \{v_{res}\})$
    $\mathbb{S}_{init} \leftarrow selectNearest(G', v_{init}, N_{init})$
    $cir_{init} \leftarrow randomSelectNodes(\mathbb{S}_{init}, \lfloor \alpha_{init} l \rfloor)$
end
if $\lfloor \alpha_{res} l \rfloor \geq 1$ **then**
    $G' \leftarrow removeFrom(G, \mathbb{S}_{init})$
    $\mathbb{S}_{res} \leftarrow selectNearest(G', v_{res}, N_{res})$
    $cir_{res} \leftarrow randomSelectNodes(\mathbb{S}_{res}, \lfloor \alpha_{res} l \rfloor)$
end
if $l - \lfloor \alpha_{init} l \rfloor - \lfloor \alpha_{res} l \rfloor \geq 1$ **then**
    $G' \leftarrow removeFrom(G, \mathbb{S}_{init} \cup \mathbb{S}_{res})$
    $\mathbb{S}_{im} \leftarrow getNodes(G')$
    $cir_{im} \leftarrow randomSelectNodes(\mathbb{S}_{im}, l - \lfloor \alpha_{init} l \rfloor - \lfloor \alpha_{res} l \rfloor)$
end
$cir \leftarrow concatenate(v_{init}, cir_{init}, cir_{im}, cir_{res}, v_{res})$
return cir

Section 3.4.

3.3.2 Circuit Construction

The key technique of our approach is the locality-aware circuit selection algorithm. Given a set of parameters of the anonymity requirements and threat levels, the algorithm selects a circuit with optimized performance. In this section we discuss the algorithm, and discuss parameter selection in the next section.

When other peers have already cached the requested resources, a peer can fetch content from them. We propose the circuit selection algorithm for APAC to choose intermediate peers from three categories: near-initiator, globally random (not nearby the initiator/responder) and near-responder nodes as a circuit for each request. Every request in APAC is encapsulated in layers of encryption (like onions) and transferred via the circuit.

Algorithm 1 describes our region-based circuit selection algorithm. In APAC,

Notation	Description
l_{cir}	The circuit's length (number of nodes including the initiator and responder in the circuit - 1)
l, L_{max}	Number of intermediate peers or relays ($l = l_{cir} - 1$), the maximum l
$\mathbb{S}_{init}$	The set of N_{init} peers nearest to the initiator
$\mathbb{S}_{res}$	The set of N_{res} peers nearest to the responder
$\mathbb{S}, \mathbb{S}_R, \mathbb{S}_{im}$	The set of all peers in the system, the set of peers having a resource R, $\mathbb{S} - \mathbb{S}_{init} - \mathbb{S}_{res}$
N, N_{init}, N_{res}, N_R	Number of peers in $\mathbb{S}$, $\mathbb{S}_{init}$, $\mathbb{S}_{res}$, $\mathbb{S}_R$
α_{init}, α_{res}	Fraction of nodes for the circuit in $\mathbb{S}_{init}$, $\mathbb{S}_{res}$
f, f_R, f_{init}, f_{res}, f_{im}	Fraction of peers are compromised in the system, $\mathbb{S}_R$, $\mathbb{S}_{init}$, $\mathbb{S}_{res}$, $\mathbb{S}_{im}$

Table 3.1: Notations for anonymity analysis.

the peer server maintains certain information about the current network (e.g., the total number of peers). The input parameters L_{max}, α_{init} , α_{res}, N, N_R, N_{init}, and N_{res} are decided by the peer server based on the trade-off between anonymity and performance, defined in Table 3.1.

- The peer server randomly chooses one peer as the responder from the set of peers having the requested resource R (line 1),
- Based on the range from 1 to L_{max}, the number of intermediate nodes (l) is determined (line 2).
- According to the distribution factor α_{init}, the peer server randomly picks nodes nearby the initiator as the first part of the circuit[3] (line 4 - 7).
- According to another distribution factor α_{res}, the peer server selects nodes nearby the responder as the last part of the circuit (line 8 - 11).
- The remaining nodes of the circuit are randomly chosen from the rest of all peers (line 12 - 15).

[3]The distances between the chosen peers and the initiator/responder are measured based on the peers' geographical coordinates, which can be obtained via *navigator.geolocation* [19] or using the GeoIP service to map network addresses to physical locations.

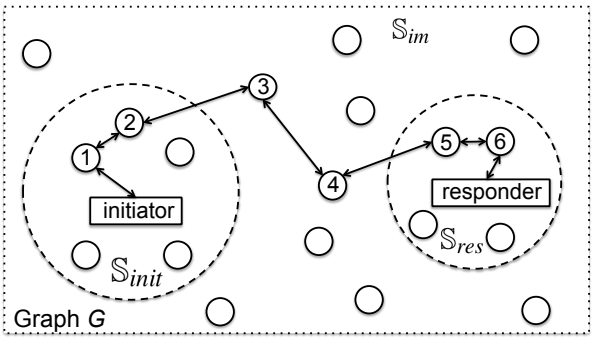


Figure 3.4: A 8-node (6-relay) circuit from the initiator to the responder in APAC.

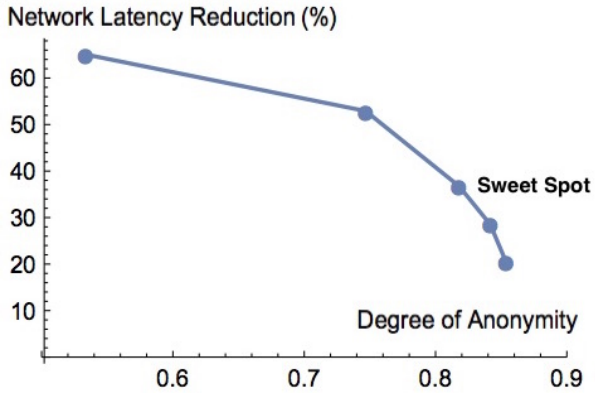


Figure 3.5: The network latency reduction (based on the loading time without APAC) decreases when increasing the degree of anonymity of the system.

- The peer server concatenates all the nodes (including the initiator, intermediate nodes and responder) in sequence as the circuit for the request (line 16).

Figure 3.4 demonstrates a 8-node circuit constructed by the peer server in APAC.

3.3.3 Parameters Selection

Depending on the requirement of anonymity/performance for the deployed web application, the input parameters for Algorithm 1 can be adjusted based on the analysis in Section 3.3.4 and 3.5. In this section, we briefly illustrate the primary factors to select these parameters.

The maximum number of intermediate nodes L_{max}. The distribution of peers in the network, the required anonymity, the fraction of adversarial peers and other factors affect the selection of L_{max}. As an in-advance conclusion, to achieve better anonymity, the system is supposed to select larger L_{max} (as shown

in Figure 3.6, 3.7 & 3.8). On the other hand, to achieve better performance of the system, the smaller L_{max} is preferred (as shown in Figure 3.13). As Figure 3.5 shows, by adjusting the setting of APAC at the sweet spot, the system can preserve an adequate level of anonymity (i.e., 0.8 degree of anonymity) with minor performance overhead[4]. In our experiment, setting two intermediate nodes for the circuit makes the system at the sweet spot.

Distribution factors α_{init} & α_{res}. To provide higher level of anonymity for peers even when adversarial peers are densely near the initiator/responder, the system can select smaller $\alpha_{init}/\alpha_{res}$ and N_{init}/N_{res} to diversify the intermediate nodes on the circuit. Thus the probability to choose an adversarial node as the intermediate node is approximate to f, which does not give the adversary significant advantage. APAC can enlarge α_{init} & α_{res} to reduce client-side network latency for peers (as shown in Figure 3.15 & 3.16).

The other parameters N, N_R, N_{init} & N_{res}. By increasing the total number of peers N and the number of peers having requested resources N_R, the system can preserve higher level of anonymity, but it requires more users to join the system to start the content delivery via peers. To reduce the client-side network latency, the system can set small N_{init} & N_{res} to let intermediate nodes nearby the initiator/responder.

The security parameters such as the maximum number of intermediate nodes L_{max} and distribution factors α_{init} & α_{res} can be tuned to adjust the anonymity level of APAC. To achieve higher level of anonymity for peers, the developer can select larger L_{max} to increase the length of a circuit and larger N/N_R to increase the threshold of starting using APAC, as well as choose smaller $\alpha_{init}/\alpha_{res}$ and N_{init}/N_{res} to diversify the intermediate nodes on the circuit. To preserve initiator/responder anonymity for peers, currently peers cannot overrule the security parameters set by the developer; otherwise, adversarial nodes may choose

[4] The figure is based on our anonymity analysis in Section 3.3.4 (Figure 3.6) and performance analysis in Section 3.5 (Figure 3.13). We consider that 100 peers (35% are adversarial) join APAC in one city, increasing the circuit's length can provide higher level of anonymity to benign peers but introduce more network latency.

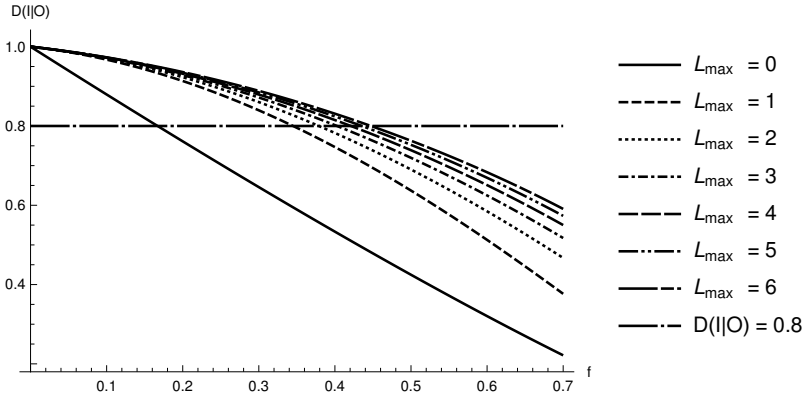


Figure 3.6: Effect of varying L_{max}: Increasing the maximum number of intermediate nodes increases the degree of initiator anonymity.

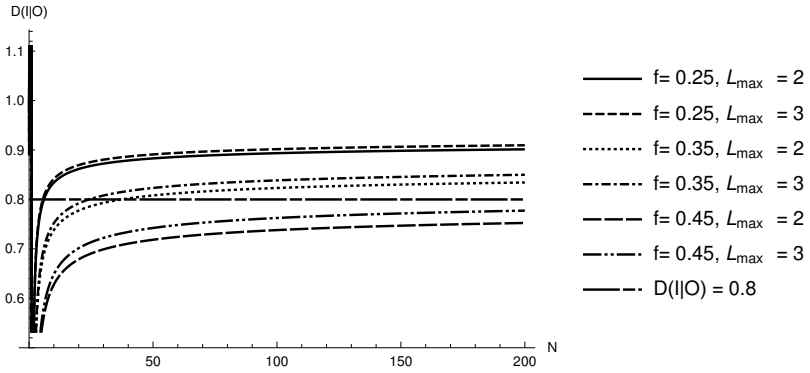


Figure 3.7: Effect of varying N: Increasing the total number of joined peers increases the degree of initiator anonymity.

low level of anonymity to easily infer the identity of the initiator/responder for a circuit. In the extreme scenario where a peer seeks an even higher privacy guarantee, it can opt out APAC to directly fetch resources from the resource server as fallback.

3.3.4 Anonymity Analysis

In this section, we analyze the anonymity of APAC in a mathematical way and demonstrate the way to choose proper parameters for the circuit selection algorithm based on the requirement of anonymity.

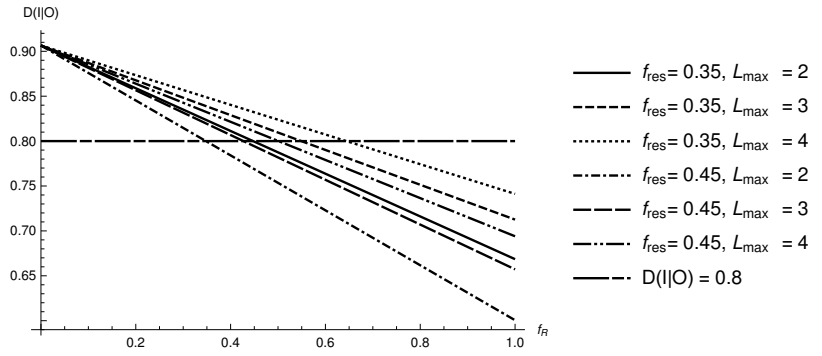


Figure 3.8: Effect of varying f_R: Increasing the number of adversarial peers having the requested resource decreases the degree of initiator anonymity.

3.3.4.1 Analysis of Initiator Anonymity

Considering the initiator v_{init} issues a request for a resource R, the peer server sets up a $(l+2)$-node circuit based on the selection algorithm (in Algorithm 1), and the responder v_{res} delivers R. Our analysis follows the definitions in Section 3.2.2 and the notations in Table 3.1.

In APAC, each intermediate node in a circuit only knows its predecessor and successor. To identify the initiator of a circuit and link the requested resource to it, the adversary has to infer the circuit's length, control the responder (i.e., to know which resource is requested) and the first relay (i.e., to identify the predecessor as the initiator). The padding for each encryption layer makes the adversary difficult to infer the relative positions of the controlled peers in a circuit. Thus to confirm one of the controlled nodes is the first relay (only when the adversary knows the length), the adversary has to control the responder and some intermediate nodes in the circuit to reconstruct the circuit. For a given circuit, by controlling the first relay, the responder and at least half of the intermediate peers alternatively in the circuit, the adversary can determine the initiator [230]. However, we consider the worst case that the adversary can learn the distance between any two controlled peers by passively logging forwarded requests and transmitted data as well as timing attacks[5]. Thus by determining

[5] To precisely determine the distance between two peers is difficult but possible for the adversary with timing attacks [66, 97]. To show the effectiveness of APAC against the adversary, we analyze the initiator anonymity in this worst case.

the correct circuit's length as well as compromising the first relay and the responder, the adversary can infer that the first relay's predecessor is the initiator.

We assume that O_I is the observation that the adversary can identify the initiator for a circuit with the probability $P(O_I)$ that O_I occurs. Based on the derivation in Section 3.6.1, the degree of initiator anonymity can be computed as:

$$D(I|O) = \frac{H(I|O)}{H_M} = (1 - P(O_I)) \frac{\log_2 ((1-f)N)}{\log_2 N} \tag{3.4}$$

In practice, the adversarial peers can exist uniformly in the system or densely around the initiator/responder. Next, we discuss the degree of initiator anonymity in these two scenarios.

Randomly placing peers in the system. In this case, the fractions of adversarial peers are equal in three different regions, i.e., $f_{init} = f_{res} = f$. If $f_R = f$, the probability for the adversary to identify the initiator of the circuit (proved in Section 3.6.1) $P(O_I) = \frac{1}{L_{max}} \sum_{l=1}^{L_{max}} \frac{f^2}{L_{max}-l+1}$. Therefore, the degree of initiator anonymity can be represented as:

$$D(I|O) = \left(1 - \frac{1}{L_{max}} \sum_{l=1}^{L_{max}} \frac{f^2}{L_{max} - l + 1}\right) \frac{\log_2 ((1-f)N)}{\log_2 N} \tag{3.5}$$

To quantify the effects of L_{max} and N to the degree of initiator anonymity, we plot Equation 3.6.2 as Figure 3.6 and 3.7. Let $N = 100$, $L_{max} = 0, 1, 2, 3, 4, 5, 6$, Figure 3.6 shows that increasing the maximum circuit length can increase the degree of initiator anonymity. The first plot ($L_{max} = 0$) represents the degree of anonymity for current peer-assisted CDNs (derivation in Section 3.6.2). We can see that the system does not hold 0.8 degree of anonymity even when $f < 0.2$. On the contrary, APAC preserves 0.8 degree of anonymity when $L_{max} = 2$ and over 35% peers are compromised. As shown in Figure 3.7, increasing the total number of joined peers can slowly increase the initiator anonymity. From Figure 3.6 and 3.7, we can see that when $L_{max} \geq 2$ and $N \geq 100$, APAC can preserve the suggested degree of anonymity (i.e., 0.8), even if 35% of all peers are under the adversary's control.

Placing peers nearby the initiator/responder. For a targeted peer, the adversary may increase f_R by storing the requested resource in more controlled peers and enlarge f_{init}/f_{res} by locating controlled peers nearby the initiator/responder. As we discussed in Section 3.3.1, without considering the worst case, the placement of peers nearby the initiator/responder does not help the adversary to gain significant advantage. Next we discuss the influence of this placement of peers in the worst case. If the first relay is in the initiator's region or the responder's region ($l_1 > 0$ or $l_1 = l_2 = 0$ & $l_3 > 0$ as shown in Equation 3.6.1), then $P(O_I) = \frac{1}{L_{max}} \cdot \sum_{l=1}^{L_{max}} \frac{f_{init} f_R}{L_{max}-l+1}$ or $P(O_I) = \frac{1}{L_{max}} \sum_{l=1}^{L_{max}} \frac{f_{res} f_R}{L_{max}-l+1}$. The degree of initiator anonymity can be computed as (using f_{res} as the representative):

$$D(I|O) = \left(1 - \frac{1}{L_{max}} \sum_{l=1}^{L_{max}} \frac{f_{res} f_R}{L_{max} - l + 1}\right) \frac{\log_2 \left((1-f)\, N\right)}{\log_2 N} \tag{3.6}$$

To quantify the effects of f_R and L_{max} to the degree of initiator anonymity, we plot Equation 3.3.4.1 as Figure 3.8. Let $N = 100, f = 0.35$, as shown in Figure 3.8, increasing f_R and f_{res} can decrease the initiator anonymity. However, if we set α_{init} and α_{res} properly, which makes $l_1 = 0$ and $l_3 = 0$, then $P(O_I) = \frac{1}{L_{max}} \cdot \sum_{l=1}^{L_{max}} \frac{f_{im} f_R}{L_{max}-l+1}$. If we also set $N_{init} = N_{res} = 0$, then $f_{im} = f$. Hence increasing f_{res} does not directly decrease $D(I|O)$. On the contrary, when f_{res} or f_{init} increases, f_{im} may decrease accordingly as f is the same. In this case, if $f_{im} = 0.35$, the first, second and fourth plots in Figure 3.8 can represent the degree of initiator anonymity accordingly. Thus when $L_{max} = 2$, APAC can preserve over 0.8 degree of the initiator anonymity if f_R is around 0.45.

3.3.4.2 Analysis of Responder Anonymity

Analogous to initiator anonymity, in order to identify the responder of a given circuit, the adversary has to determine the circuit's length as well as control the last relay and the initiator. We assume that O_R is the observation that the adversary can identify the responder for a circuit with the probability $P(O_R)$ that O_R occurs. The degree of responder anonymity can be computed as (details

of the derivation in Section 3.6.1):

$$D(R|O) = \frac{H(R|O)}{H_M} = (1 - P(O_R)) \frac{\log_2 ((1-f)N)}{\log_2 N} \tag{3.7}$$

The analysis for the two scenarios: adversarial peers uniformly exist in the system and more adversarial peers near the initiator/responder, is similar to the analysis for the degree of initiator anonymity. As a result, if we properly tune the parameters, e.g., $L_{max} = 2, N = 100, \lfloor \alpha_{init} l \rfloor = 0$ and $\lfloor \alpha_{res} l \rfloor = 0$, APAC can preserve 0.8 degree of initiator/responder anonymity even if 35% of all peers are adversarial and have different distributions.

3.4 Implementation of APAC

In this section, we discuss the implementation details of APAC. We implement APAC with 2600+ lines of code as well as several libraries and frameworks. APAC's client-side code is written in JavaScript with 1000+ lines of code, apart from three libraries [13, 38] The peer server is also written in JavaScript with 1600+ lines of code based on Node.js platform and PeerServer [38]. Our prototype implementation is available online [4]. APAC is compatible with mainstream browsers (e.g., Chrome, Firefox and Opera) and various operating systems (e.g., Mac OS, Linux and Windows). Similar to other peer-assisted CDNs (e.g., Swarmify), APAC is designed not to violate the same-origin policy — APAC is deployed "per website" along with the client-side scripts.

3.4.1 Components in APAC

In APAC, the communications between the peer server and peers are over HTTPS (e.g., TLS protocol [52]), and the data transmission among peers are over WebRTC which uses DTLS protocol [15] as shown in Figure 3.9. TLS and DTLS protocols ensure that communication channels are secure and the adversary can-

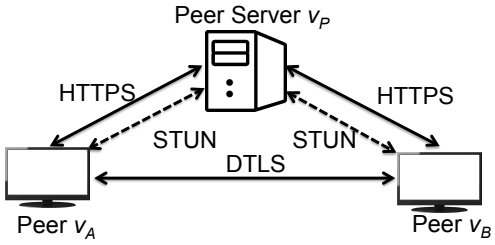


Figure 3.9: Overview of the communication channels in APAC.

not eavesdrop or tamper with any message in the communications among peers and servers. APAC utilizes Session Traversal Utilities for NAT (STUN) [43] to deal with Network Address Translator (NAT) [30] traversal. In this way, even behind NAT a peer can communicate with another peer with a public IP address or also behind NAT.

Resources in APAC**.** Similar to other CDNs, e.g., Swarmify and NetSession, only static resources (e.g., files, images and videos) are delivered in APAC. To ensure content integrity, we use flat naming mechanism for resources, i.e., naming the resource with its hash value. Alternatively, we can attach the hash value for each resource request, then the peer can verify the content with the value at the client side. Since the content server is trusted, the peer only needs to verify the integrity of the resource fetched from other peers. To preserve authenticity, peers are limited to fetch resources provided/authenticated by the content server from other peers. The content server in APAC is the website's server or the CDN's edge server, which stores the website's pages and resources. To retrofit a website to deploy APAC, developers only need to append APAC's client-side code on pages. Optionally, developers can explicitly specify several shared resources instead of sharing all resources in the page.

Peer Server in APAC**.** The peer server is in charge of maintaining the connections with peers, the properties of each peer (e.g., identity, IP address, peer key, locality and names of stored resources) and the properties of each resource (e.g., location in the content server and identities of peers having the resource). Meanwhile, when a peer requests a resource, the peer server either assists the

peer to fetch the content from the server or constructs a circuit for the peer to retrieve the resource from another peer. Key management and path choosing are done by the peer server, and peer servers can be distributed to improve fault tolerance.

Peers in APAC**.** To achieve our third goal (compatibility), APAC is designed to support a plug-and-play style joining/leaving without any solicited action. When a user visits the retrofitted site, APAC's client-side code automatically assists the user's browser to join APAC as peers to fetch/deliver content from/to other peers, without any additional installation of extensions or software. Users leave APAC's network when closing tabs, like Swarmify.[6] The client-side code is purely based on JavaScript and compatible with all mainstream browsers. With WebRTC [59], the code enables browser-based real-time communication and data transmission for peers. It also helps peers to store the requested contents in indexedDB [24] (i.e., a high-performance client-side storage), which enables peers to directly load the same resources from indexedDB without issuing new requests. With data URI scheme [14] [7], the client-side code appends the content fetched from the server, another peer or indexedDB to the specific location in the visiting page.

3.4.2 Content Delivery in APAC

In this section, we illustrate how peers fetch content from the content server, other peers and indexedDB in APAC.

Initiation of peers. When a user first visits the site deployed with APAC, the client-side code will negotiate with the peer server to assist the user's browser to join the system as a peer. The user's browser issues an initiation request to the peer server with the current IP address or coordinates obtained from *nav-*

[6] To support opt-out in APAC for users, the website can host two copies, i.e., the original one and the retrofitted one. A user can explicitly choose which one to visit via setting preferences in the site's first page.

[7] The data URI scheme encodes data with base64, which introduces certain overhead. We use other serialization techniques for data transmission to reduce the overhead, and only use URI scheme when appending the data to web pages.

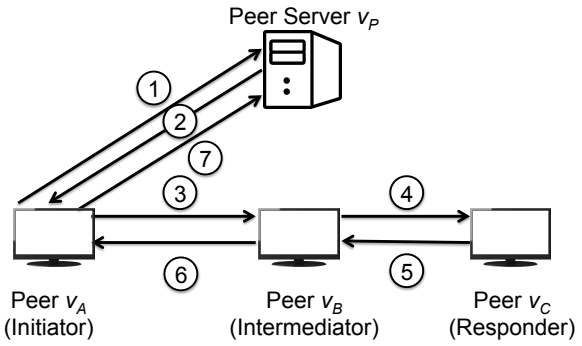


Figure 3.10: Overview of how a peer fetches content from another peer via three nodes in APAC.

igator.geolocation [19]. After receiving the request, the server recognizes the user's browser as a new peer, and responds the peer with an identity and a peer key (symmetric key). Meanwhile, the server records the peer with its properties, i.e., identity, IP address, peer key and locality (derived from IP addresses or coordinates). As for the peer key, after a period of time, each peer will communicate with the peer server to update a new key. After the initiation, the new peer can take advantage of the service provided by APAC to fetch/deliver resources from/to other peers.

Fetching content from the content server. When the total number of peers and the number of peers having the requested resource do not meet the requirement for anonymity (the input parameters in the circuit selection algorithm), the peer server will instruct peers to retrieve resources directly from the content server. In this case, the peer server responds the peer with the requested resource's original URL on the content server. Then the peer issues another request to the content server to retrieve the resource. After receiving the resource, the peer stores it in indexedDB, appends it to the page and updates the caching status to the peer server.

Fetching content from other peers. Content delivery via peers can start to operate when the system is sufficient to preserve the required anonymity as discussed in Section 3.3.4. We illustrate how peer v_A fetches a resource R from v_C

via a 3-node circuit (as shown in Figure 3.10) step by step[8]

- ①: Peer v_A issues a request for a resource R to the peer server v_P;
- ②: After receiving the request, v_P first searches for online peers having R. Then v_P sets up a 3-node circuit based on the configuration, constructs the packet below with layered encryption, and responds v_A with the packet.

$$\begin{aligned} &\{ID_{v_B}, K_R, \{ID_{v_C}, Nonce_{v_B}, \\ &\{R_{name}, K_R, Nonce_{v_C}\}_{K_{v_C}}\}_{K_{v_B}}\}_{K_{v_A}} \end{aligned} \tag{3.8}$$

  K_{v_A}, K_{v_B} and K_{v_C} are peer keys for v_A, v_B and v_C respectively. ID_{v_B} and ID_{v_C} are the identities for v_B and v_C. K_R is the key generated by v_P for v_A and v_C to encrypt/decrypt R. R_{name} is the requested resource's name. $Nonce_{v_B}$ and $Nonce_{v_C}$ contain the timestamp and padding, which make any intermediate node difficult to determine its relative position on the circuit. With Advanced Encryption Standard (AES) [113], the data (i.e., R_{name}, K_R and $Nonce_{v_C}$) is encrypted by K_{v_C} (denoted by $\{...\}_{K_{v_C}}$), K_{v_B} and K_{v_A} layer by layer.
- ③: Peer v_A receives the packet, decrypts it with K_{v_A}, and obtains ID_{v_B} (i.e., the next peer to contact), K_R and the remaining cipher text. Then v_A forwards the remaining data to v_B.
- ④: While receiving the packet from v_A, v_B obtains ID_{v_C} and $Nonce_{v_B}$ by decrypting it with K_{v_B}. After verifying the timestamp in $Nonce_{v_B}$, v_B forwards the remaining packet to v_C.
- ⑤: Peer v_C receives the packet from v_B, strips off the layer with K_{v_C}, and obtains R_{name}, K_R and $Nonce_{v_{res}}$. v_C verifies the timestamp in $Nonce_{v_{res}}$ and searches R_n in its indexedDB. If the timestamp is not out of date and R exists in the indexedDB, v_C responds to v_B with $\{R, Nonce_R\}_{K_R}$. Otherwise, v_C

[8] Alternatively, the peer server can also construct a circuit and instruct the responder to deliver the requested resource to the initiator via the circuit. However, in this way every two nodes also have to set up a connection first and then perform data transmission, which is the same as the original design and does not save extra time for content delivery.

responds to v_B with an empty packet.

- ⑥: After receiving the response from v_C, v_B directly forwards the packet to v_A.
- ⑦: When receiving $\{R, Nonce_R\}_{K_R}$ from v_B, v_A decrypts it with K_R and obtains R. v_A verifies the integrity of R. If the obtained packet is empty or R is bogus, v_A issues another request for R to v_P. Otherwise, v_A stores R in its indexedDB and issues a request to v_P to update the status that v_A has R.

Fetching content from indexedDB. Once a peer successfully fetches a resource from the content server or another peer (with verifying the integrity) and stores it in the indexedDB, the peer will directly load the resource from its indexedDB instead of issuing a new request afterwards. APAC is compatible with the current browser cache, i.e., the default cache and HTML5 application cache. The APAC's client-side code typically appends resources using data URI scheme. Since these cache mechanisms support to cache resources with data URI scheme, they can help to speed up the loading time for these resources.

3.5 Performance Evaluation

To demonstrate the effectiveness of APAC, we evaluate the *network throughput* of the deployed site's server, and the *network latency* of fetching resources in APAC. We investigate the following research questions in this section.

- What is the bandwidth saving of APAC compared with non-peer-to-peer client/server system?
- What is the latency reduction of APAC with various configurations (e.g., circuit's length and distribution factors)?

3.5.1 Measurement Setup

We have deployed APAC on a website that provides images of different sizes in a range of 1 KB - 2 MB. APAC supports all standard text and media for-

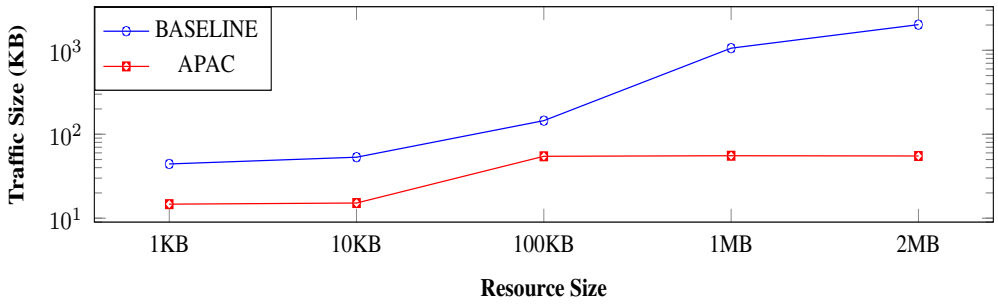


Figure 3.11: Total outgoing network traffic size of a server in response to a request in APAC setting ($APAC$) and in the client-server setting ($BASELINE$), measured in KB.

mats, e.g., html, JavaScript, jpg, ogg and mp4. Since the majority of requests in popular websites are for images [231], we use images as the representatives of distributed resources in our experiment. Both the site's content server (i.e., edge server) and the peer server are located in City A[9]. Considering a typical scenario that peers and the content/peer server are in different cities, we place peers in City B different from City A. We launch over 100 different browser instances with located across City B. When browsing the same site deployed with APAC, these browsers join the system as peers.

To measure the network throughput, we deploy a performance measurement tool called `iftop` on the content server as well as the systems hosting browser instances. To measure the client-side network latency in various situations, we adjust the settings in the circuit selection algorithm. We vary the number of nodes on the circuit from 2 (no intermediate node) to 6 by changing L_{max}, and diversify the locations of intermediate nodes (tuning distribution factors α_{init} and α_{res}) in City B. We demonstrate how these settings for circuit construction affect the performance of APAC.

Comparison to other CDNs. CDN operators can increase the geo-density of edge servers up to a point (e.g., placing edge servers in major cities of various countries), but fetching resources for users far away from an edge server has

[9] In this chapter, City A represents New York City, and B represents Singapore. In our evaluation, we deployed one peer server, which is adequate for our experiments of 100 peers. For a large scale of users, e.g., 1 million, multiple peer servers can be deployed to handle requests in parallel.

major latency. Considering this typical scenario, we place peers and the peer/-content server (or CDN's edge server) in different cities in our experiment setting. In this setting, compared to regular CDN services that clients directly fetch resources from the CDN's edge server, APAC can reduce network bandwidth and network latency. We show the details of bandwidth savings in Figure 3.11 and 3.12, and network latency reduction in Figure 3.13. For example, when every circuit has up to 4 nodes, APAC can reduce 97.3% bandwidth for the CDN's edge server and reduce 27.4% to 44.1 network latency of fetching 2 MB resources for peers. Naturally, a single-hop peer-assisted CDN, such as Swarmify, can achieve higher network latency reduction (e.g., 69.4% for Swarmify case) than APAC does. However, it does not preserve the anonymity guarantee which APAC aims to provide. We provide the detailed comparison on latency reduction in Figure 3.13.

3.5.2 Bandwidth Saving

First, we measure how much bandwidth can be saved by deploying APAC on the server side. We record the total size of network packets that go out from the content/peer server after the circuit ($L_{max} = 2$ for this experiment) has been established and the initiator starts fetching a specific resource (labeled as $APAC$). As a baseline, we also measure the size of outgoing network traffic from the content server in the typical client-server environment (labeled as $BASELINE$). We define the bandwidth saving (labeled as BS) as the percentage reduction from $BASELINE$.

Figure 3.11 shows the total size of outgoing network traffic from the server in client-server and APAC settings. The total size of outgoing packets from the server in APAC ($APAC$) is much smaller than the size of network traffic in the normal client-server setting ($BASELINE$), and this applies to all resources with different sizes. The reason is that peers in APAC can fetch a large fraction of resources from other peers instead of the server. When the initiator

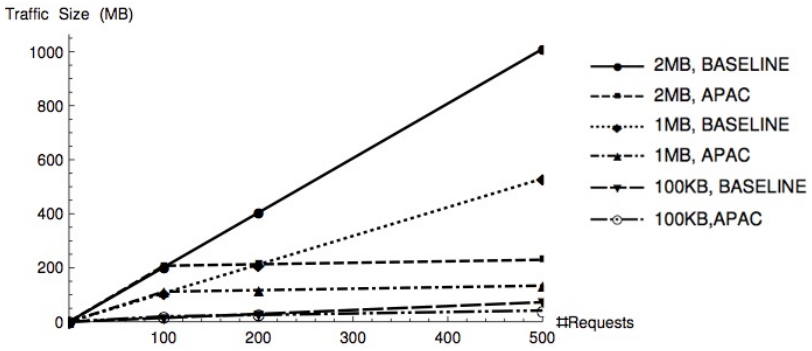


Figure 3.12: When the number of requests increases, the total outgoing network traffic for $BASELINE$ significantly increases, but the traffic for $APAC$ only slowly increases.

fetches a resource of 2 MB[1], APAC can save 97.3% of the server's bandwidth[10]. When the total number of requests increases, APAC can significantly save the bandwidth as shown in Figure 3.12[11].

3.5.3 Network Latency

Next, we measure the network latency when fetching content in APAC with different peer placements. According to the statistics until 15th February 2016 [50], the averaged total size of transferred data when loading a site is 2268 KB. Thus in our experiment, we use the loading time of 2 MB resources (i.e., images) as the representative. With fixing the size of the resource to 2 MB, we evaluate APAC under the settings where the initiator, responder and intermediate nodes are located in: 1) the same LAN, 2) the same WLAN and 3) different LANs but in the same city (i.e., City B, labeled as "WAN-City"). We then vary the length of circuit as well as distribution factors and compare those results with a baseline: the network latency for a browser in City B to directly fetch the resource from the content server without APAC[12]. Analogous to BS, we define the client-side network latency reduction NLR as the percentage reduction from $BASELINE$. The higher the percentage of NLR is, the better is the

[10] $BASELINE$ = 2020 KB, $APAC$ = 55.1 KB, and BS = 97.3%.

[11] We set the requirement for the total number of peers as 100, thus the content delivery via peers starts to operate when 100 peer join the system.

[12] During the initial investigation, we observed the baseline latency ($BASELINE$) to be 9420 ms.

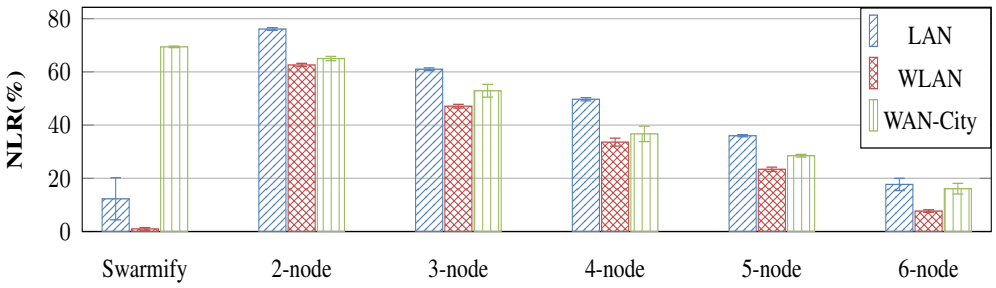


Figure 3.13: Network latency reduction (NLR) of an initiator peer in Swarmify and APAC under three network configurations, as compared to the baseline. All data are based on a resource with size of 2 MB and averaged over 30 runs with 95% confidence intervals for each of them.

performance (0% = $BASELINE$). All our results are the average of 30 runs with 95% confidence intervals for each of them.

Varying the circuit's length. In Figure 3.13, "Swarmify" represents the Swarmify system, and "2-node" represents the non-anonymous peer-to-peer setting in APAC, i.e., 2 nodes without any intermediate nodes. Since they do not have any intermediate nodes to route data, they have the largest network latency reduction, i.e., 69.4% for Swarmify in WAN and 76.1% for APAC in LAN respectively. Due to the implementation issue, Swarmify does not properly support P2P data transmission in LAN and WLAN, so the NLR for Swarmify in LAN and WLAN are quite small in Figure 3.13. We have reported this issue to their developer team. Since APAC involves more intermediate nodes than a single-hop non-anonymous system like Swarmify, APAC naturally shows a less latency improvement than Swarmify. This is illustrated in Figure 3.13, for a 4-node circuit where APAC provides a latency reduction (49.7%) lower than the performance obtained for Swarmify (69.4%) and non-anonymous setting (76.1%). Notably, APAC still outperforms the baseline. In general, with the increase of the circuit's length, the network latency reduction decreases, as more hops are required to route the transmitted data. In the three configurations, the latency in LAN is the smallest, as all the peers in the circuit are nearby. Since the bandwidth in WLAN is limited, the latency in WLAN is larger than the other two settings.

We also evaluate the setup latency of a circuit for various number of interme-

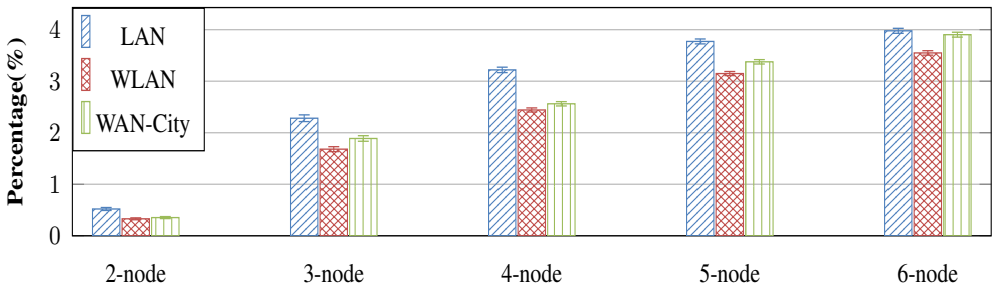


Figure 3.14: The percentage of the network latency for setup overhead of a circuit in three configurations.

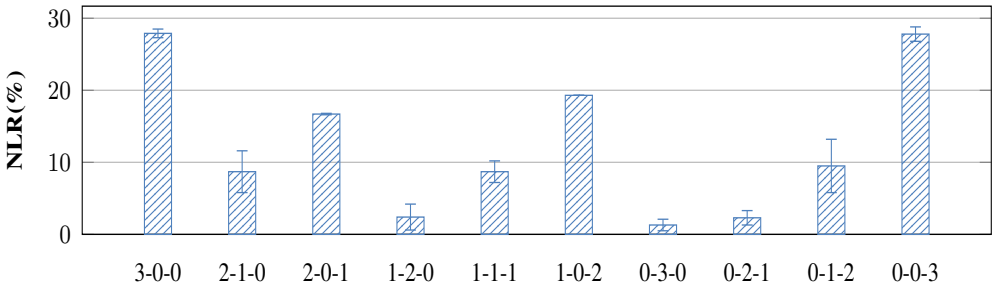


Figure 3.15: Network latency reduction (NLR) varies when intermediate nodes are in different regions. "a-b-c" means the number of nodes in the initiator's region ($a = \lfloor \alpha_{init} l \rfloor$), nodes randomly chosen ($b = l - \lfloor \alpha_{init} l \rfloor - \lfloor \alpha_{res} l \rfloor$) and nodes in the responder's region ($c = \lfloor \alpha_{res} l \rfloor$).

diate nodes. As Figure 3.14 shows, the setup latency of a circuit is only a small fraction of the network latency. For example, the setup latency of a circuit for 2 to 6 nodes is 11.6 ms, 83.8 ms, 152.7 ms, 227.3 ms, and 308.5 ms respectively in the LAN setting. The percentage of the network latency for each of them is 0.52%, 2.28%, 3.22%, 3.77%, and 3.98% respectively.

Varying the distribution factors α_{init} and α_{res}. In addition to adjusting the circuit's length, we can also tune the distribution factors α_{init} and α_{res} to reduce the latency in APAC. For a 5-node circuit, we place the initiator and responder in two different regions (different LANs across City B), and we diversify the locations of intermediate peers to change α_{init} and α_{res}. As Figure 3.15 illustrates, setting more intermediate nodes around the initiator/responder (increasing α_{init}/α_{res}) for a circuit, the system further reduces the network latency. Thus for the circuit construction, the system can select larger α_{init} and α_{res} to reduce performance overhead. For the best case in our experiment, the system

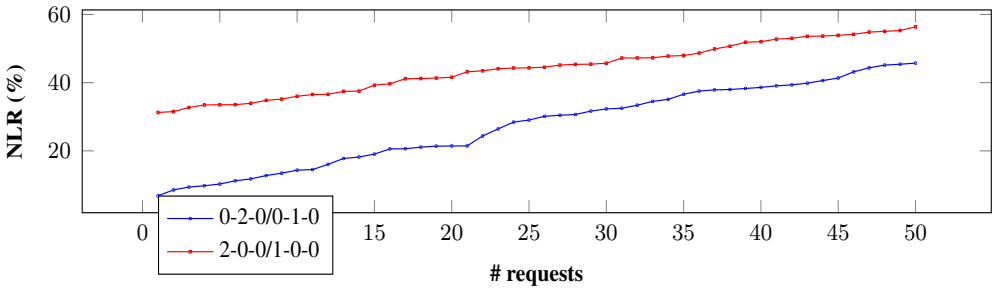


Figure 3.16: The network latency reduction (NLR) for 50 requests(sorted in ascending order) when 100 joined peers are in APAC.

can reduce 27.8% latency only by adjusting the distribution factors. The system can be set as not "locality-aware" and randomly chooses all intermediate nodes (i.e., 0-3-0 setting) for the circuit. This configuration causes the worst performance, but it enables the system to preserve the best level of anonymity comparing to other settings.

Overall performance. We evaluate the overall performance in APAC for the deployed site, when 100 peers (browsers) have joined APAC and are available for content delivery. We set the maximum number of intermediate nodes on a circuit as 2 ($L_{max} = 2$), and record each 50 different resource requests in APAC when all intermediate nodes are randomly chosen (0-2-0/0-1-0 for $\lfloor \alpha_{init} l \rfloor = \lfloor \alpha_{res} l \rfloor = 0$) and all relays are nearby the initiator (2-0-0/1-0-0 for $\lfloor \alpha_{init} l \rfloor = l$) respectively. The average NLR for the 0-2-0/0-1-0 and 2-0-0/1-0-0 settings are 27.4% and 44.1% respectively. The 0-2-0/0-1-0 setting has the considerable network latency reduction, though it holds the worst performance comparing to other settings analogous to Figure 3.15. The 0-2-0/0-1-0 setting provides the higher level of anonymity than the 2-0-0/1-0-0 one, which makes the adversary that places peers near the victim gain limited advantage over the honest-but-curious adversary as discussed in Section 3.3.4.1. As Figure 3.16 demonstrates, the latency reduction in the 2-0-0/1-0-0 setting is much larger than the other one, which reflects that tuning distribution factors α_{init} and α_{res} can assist APAC to provide reasonable performance.

Key findings. As we discussed in Section 3.3.4, when the circuit has up to

two relays ($L_{max} = 2$), and adversarial peers uniformly exist in the network ($f = f_{init} = f_{res} = f_R$), APAC can preserve the adequate degree of initiator/responder anonymity (i.e., 0.8). As we show in this section, with intermediate peers up to 2 ($L_{max} = 2$) and proper setting for distribution factors (α_{init}/α_{res}), APAC can save 97.3% bandwidth for the site's server and reduce 27.4%/44.1% network latency of fetching 2 MB resources for peers within one city. Therefore, APAC can preserve initiator/responder anonymity with considerable bandwidth saving and latency reduction.

3.5.4 Performance under Churn

By default, APAC is deployed without any browser extensions, so once a user closes the tab for the retrofitted site, her browser (or peer) will leave APAC's network. If the peer happens to be a node in a circuit, its departure will break down the circuit, and the initiator has to issue another request. Therefore, as a part of the dynamics of peer participation or *churn* [216], the departure of peers in the duration of a request/circuit (i.e., from the creation of the circuit to its teardown) influences the success of data transmission through the circuit. To quantify the success rate of circuits under Churn, we mathematically analyze this issue based on a previous study conducted by Liu *et al.* [169]. It shows that the time users spend on a web page follows Weibull distribution [60]. We assume that the stay time of users on the deployed site's pages also follows Weibull distribution, and we detail our analysis in Section 3.6.3.

As an in-advance conclusion, we show that the success rate of a circuit decreases when its length l or its duration t' increases in Figure 3.17. We can see that if data transmission via a circuit is finished quickly (i.e., the circuit has short duration), e.g., a small-size resource transferred in a high-speed network, the success rate of the circuit is quite high, e.g., over 90% when the median stay time is 39.8 s and $t' = 2$. This indicates that over 90% requested resources can be successfully transmitted via circuits for the first time without issuing ad-

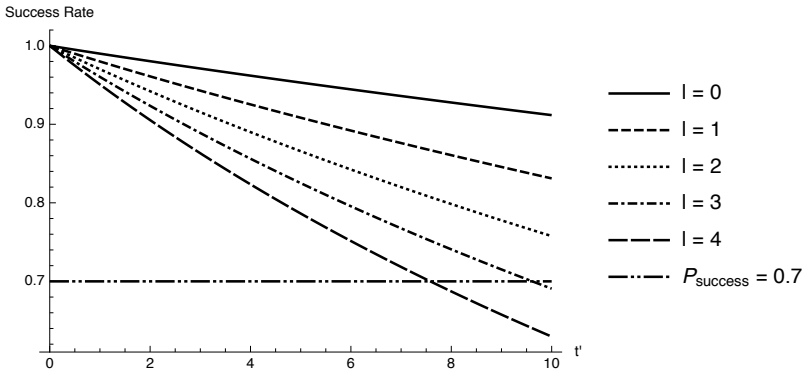


Figure 3.17: The success rate decreases when the length of the circuit increases.

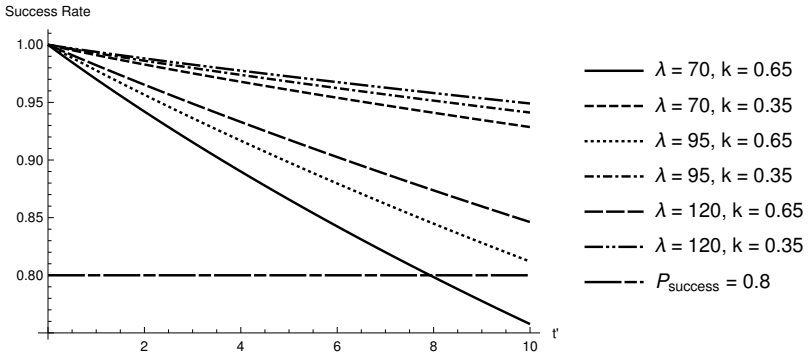


Figure 3.18: The success rate increases when the stay time follows the Weibull distribution with larger λ and smaller k. (The circuit has 2 intermediate nodes.)

ditional requests. In some cases, e.g., fetching 2 MB resources via a 4-node circuit, the duration of the circuit is similar to the fetching time directly from server (9.42 s), the success rate is still around 80%. If the site can incentivize users to stay longer on the page, the success rate can increase a lot, e.g., over 95% with 42.1 s median stay time (as plotted in Figure 3.18). Alternatively, the site operator can create several backup circuits for one request, and the complete data transmission through any circuit is counted as the success for the request. This can drastically increase the success rate, e.g., over 2 backup circuits can make the success rate over 99% (details in Section 3.6.3).

3.5.5 Load on Peers

We consider CPU and bandwidth for APAC to evaluate the load on peers. The size of the client side code of APAC is 9.5 KB, and 68KB when including all additional libraries. This is significantly smaller than the average-transfer data

(2268 KB [50]) when loading websites. APAC would not affect the client-side performance much. Notably, most of the JavaScript files are static and cached.

We instrumented the htop and iftop tools [21, 23] to measure the average CPU usage and bandwidth for peers in a 3-node circuit for 10 times. Each client runs on a machine with a typical PC configuration Intel i7-2600 CPU of 3.4 GHz and 8 GB memory. We find that APAC only incurs a reasonable 0.88% of CPU overhead and 4.21MB bandwidth per circuit for the clients. Further, the site operator can also limit the maximum number (e.g., 10) of joining circuits for one peer in one cycle of completing data transmission in a circuit, to avoid exhausting the peer's available bandwidth.

3.6 Security Analysis

3.6.1 Degree of Initiator/Responder Anonymity in APAC

Initiator Anonymity. In our analysis, we consider the worst case for the system that the adversary can learn the distance between any two controlled peers by passively logging and timing attacks. We assume that peers are compromised independently. Let O_I be the observation that the adversary can determine the initiator for a given circuit with probability $P(O_I)$. For a given circuit in APAC, the probability of identifying the initiator of the circuit equals to the probability that the adversary determines the correct circuit's length as well as controls the first relay and the responder. Due to the distribution factors α_{init} and α_{res}, the first relay may be in $\mathbb{S}_{init}$, $\mathbb{S}_{im}$ or $\mathbb{S}_{res}$. Based on Algorithm 1, for a circuit in APAC, the responder is randomly chosen in $\mathbb{S}_R$. Following the notations in Table 3.1, for a given circuit with l intermediate nodes, the probability of

controlling the first relay and the responder can be computed as:

$$P(Q_I|L=l) = \begin{cases} f_{init}f_R & \text{if } l_1 \geq 1 \\ f_{im}f_R & \text{if } l_1 = 0, l_2 \geq 1 \\ f_{res}f_R & \text{if } l_1 = l_2 = 0, l_3 \geq 1 \end{cases} \tag{3.9}$$

where $l_1 = \lfloor \alpha_{init} l \rfloor$, $l_2 = l - \lfloor \alpha_{init} l \rfloor - \lfloor \alpha_{res} l \rfloor$, $l_3 = \lfloor \alpha_{res} l \rfloor$, and $f_{im} = \frac{fN - f_{init}N_{init} - f_{res}N_{res}}{N - N_{init} - N_{res}}$.

In APAC, the length of the circuit is variable, and number of intermediate nodes is randomly chosen from 1 to L_{max}. Thus the probability that the number of intermediate nodes is l can be computed as $P(L = l) = \frac{1}{L_{max}}$. For a specific circuit with l relays, when the adversary controls the first relay and the responder, he still cannot precisely determine the predecessor of the first relay as the initiator, as he does not know the exact length of the circuit. Therefore, l, $l+1$,..., L_{max} can be considered as the number of intermediate nodes, and the probability that the adversary guesses the correct length is $P(L' = l|(Q_I|L=l)) = \frac{1}{L_{max}-l+1}$. The probability that O_I occurs when $L = l$ can be represented as $P(O_I|L = L) = P(L' \cap Q_I|L = l) = P(Q_I|L = l)P(L' = l|(Q_I|L = l)) = P(Q_I|L = l)\frac{1}{L_{max}-l+1}$. Generally, for a specific circuit, the probability that the adversary can identify the initiator is:

$$\begin{aligned} P(O_I) &= \sum_{l=1}^{L_{max}} P(L=l)P(O_I|L=l) \\ &= \frac{1}{L_{max}} \sum_{l=1}^{L_{max}} \frac{P(Q_I|L=l)}{L_{max}-l+1} \end{aligned} \tag{3.10}$$

Under this condition, the adversary can precisely to determine which peer is the initiator. Thus the entropy for identifying the initiator is $H(I|O_I) = 0$.

When the adversary does not have the observation O_I as above, it is impossible for the adversary to directly identify the circuit's initiator. From the adversary's perspective, the initiator anonymity set is $(1-f)N$. Therefore the entropy under this situation is: $H(I|\neg O_I) = -\sum_{i=1}^{(1-f)N} \frac{1}{(1-f)N} \log_2 \frac{1}{(1-f)N} =$

$\log_2((1-f)N)$. From the adversary's observation O (i.e., O_I and $\neg O_I$), the overall entropy for the system can be represented as $H(I|O) = (1-P(O_I))\log_2((1-f)N)$. For the ideal case, every peer has the equal probability of $\frac{1}{N}$ to be identified as the initiator. The maximum entropy $H_M = \log_2 N$. Therefore, the degree of initiator anonymity for the system can be computed as:

$$D(I|O) = \frac{H(I|O)}{H_M} = (1 - P(O_I))\frac{\log_2((1-f)N)}{\log_2 N} \tag{3.11}$$

Responder Anonymity. Analogous to derivation for degree of initiator anonymity, we can derive Equation 3.3.4.2 for degree of responder anonymity. We briefly illustrate the derivation for degree of responder anonymity. Let O_R be the observation that the adversary can determine the responder for a given circuit with probability $P(O_R)$ to occur. For a given circuit in APAC, the probability of identifying the responder of the circuit equals to the probability that the adversary determines the correct circuit's length and controls the last relay and the initiator. Following the notations in Table 3.1, we show the difference between $P(Q_R|L = l)$ and $P(Q_I|L = l)$ below. For a given circuit with l intermediate nodes, the probability of controlling the last relay and the initiator can be computed as:

$$P(Q_R|L = l) = \begin{cases} f_{res}f & \text{if } l_3 \geq 1 \\ f_{im}f & \text{if } l_3 = 0, l_2 \geq 1 \\ f_{init}f & \text{if } l_3 = l_2 = 0, l_1 \geq 1 \end{cases} \tag{3.12}$$

where L_1, l_2, l_3 and f_{im} are defined same as Equation 3.6.1, and we assume that every peer is equally to be the initiator for a request.

In APAC, the length of the circuit is variable, and number of intermediate nodes is randomly chosen from 1 to L_{max}. Analogous to Equation 3.6.1, for a specific circuit, the probability that the adversary can identify the responder can be represented as $P(O_R) = \frac{1}{L_{max}}\sum_{l=1}^{L_{max}}\frac{1}{L_{max}-l+1}P(Q_R|L = l)$. From the

adversary's observation O (i.e., O_R and $\neg O_R$), the overall entropy for the system can be computed as $H(R|O) = (1 - P(O_R)) \log_2 ((1 - f) N)$. Therefore, the degree of responder anonymity for the system can be computed as:

$$D(R|O) = \frac{H(R|O)}{H_M} = (1 - P(O_R)) \frac{\log_2 ((1 - f)N)}{\log_2 N} \tag{3.13}$$

The adversary's advantage on placement of peers. In the normal analysis, for a 8-node circuit ($l = 6$) as shown in Figure 3.4, the initiator, intermediate peers and responder are v_{init}, v_1, v_2, v_3, v_4, v_5, v_6, and v_{res} respectively. By controlling v_1 (nearby the initiator), v_3 (globally), v_5 (nearby the responder) and v_{res}, the adversary can definitely determine that v_{init} is the initiator. Referring to the notations in Table 3.1, we assume that the adversary controls a fraction f of N peers in APAC, a fraction f_{init}/f_{res} of N_{init}/N_{res} peers nearby the initiator/responder are adversarial, and f_R is the fraction of adversarial peers having the requested resource. If the adversary randomly places adversarial peers in APAC, then $f = f_{init} = f_{res}$, and the adversary has the probability $P_{rand} = f^3 f_R$ to infer the initiator. If the adversary places more peers around the initiator/responder to increase f_{init}/f_{res}, she has the probability $P_{near} = f_{init} f_{res} \frac{fN - f_{init}N_{init} - f_{res}N_{res}}{N - N_{init} - N_{res}} f_R$ to determine the initiator. Let $f = f_R = 0.35$, $N = 1000$, $N_{init} = N_{res} = 300$, we find that the adversary's advantage of using the second strategy over the first one is quite limited: $\Delta_{adv} = max\{P_{near}\} - P_{rand} = 0.0154 - 0.0150 = 0.004$, where $P_{near} = max\{P_{near}\}$ when $f_{init} = f_{res} = 0.39$.

3.6.2 Degree of Anonymity in Current Peer-assisted CDNs

For a given request in peer-assisted CDNs, the initiator can be identified when the responder is adversarial. We assume that O_I is the observation when the responder of a particular request is under the adversary's control with the probability $P(O_I)$ that O_I occurs. Analogous to Section 3.6.1, we can derive the

same Equation 3.6.1, but the meaning of $P(O_I)$ is different. The entropy for the system with different observations (O_I and $\neg O_I$) can be computed as:

$$\begin{aligned} H(I|O) &= P(O_I)H(I|O_I) + (1 - P(O_I))\, H(I|\neg O_I) \\ &= (1 - P(O_I)) \log_2 ((1 - f)\, N) \end{aligned} \tag{3.14}$$

where N is the number of all peers. When O_I occurs, the adversary can definitely identify the initiator, thus $H(I|O_I) = 0$; otherwise, from the adversary's perspective, the initiator anonymity set is $(1 - f)N$. The probability that a benign peer in the system is the initiator is $\frac{1}{(1-f)N}$. Thus the entropy for the system when $\neg O_I$ occurs is $H(I|\neg O_I) = -\sum_{i=1}^{(1-f)N} \frac{1}{(1-f)N} \log_2 \frac{1}{(1-f)N} = \log_2 ((1 - f)N)$. For the ideal case, every peer has the equal probability of $\frac{1}{N}$ to be identified as the initiator. The maximum entropy H_M can be calculated as $H_M = -\sum_{i=1}^{N} \frac{1}{N} \log_2 \frac{1}{N} = \log_2 N$. Therefore, the degree of responder anonymity can be computed as:

$$D(I|O) = \frac{H(I|O)}{H_M} = (1 - P(O_I)) \frac{\log_2 ((1 - f)\, N)}{\log_2 N} \tag{3.15}$$

When the adversarial peers are uniformly scattered in the system, then an adversarial peer having the resource has the probability f to be chosen as the responder, i.e., the probability that the responder is adversarial $P(O_I) = f_R = f$. As the first plot ($L_{max} = 0$) in Figure 3.6 shows, the degree of initiator anonymity is less than 0.8 when $f \geq 0.2$. Analogous to the initiator anonymity, the current peer-assisted CDNs cannot preserve an adequate degree of responder anonymity when over 20% peers are adversarial.

3.6.3 Analysis of Churn in APAC

A user joins APAC when visiting the deployed site from her web browser, and leaves APAC when closing the tab. As an inherent property of peer-to-peer systems, the dynamics of peer participation, or *churn* [216], especially the stay

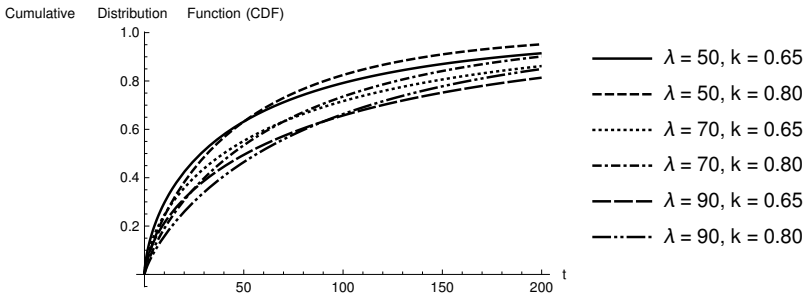


Figure 3.19: λ becomes larger and k turns smaller, when more users stay longer on the page.

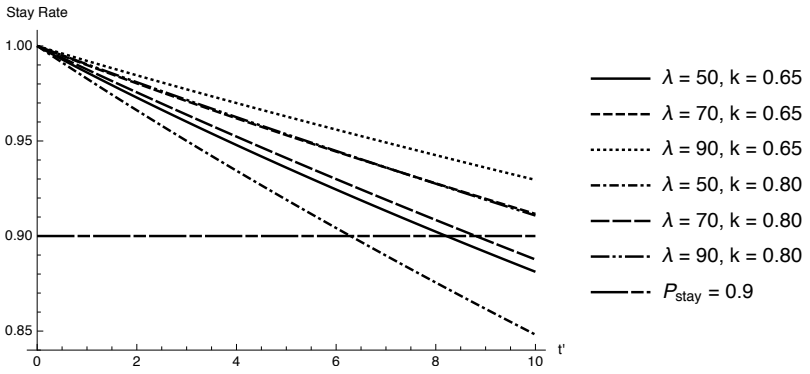


Figure 3.20: The stay rate decreases when the duration of a circuit and k increase, as well as λ decreases.

time of peers, affect the success rates of data transmission for circuits. To provide a mathematical understanding of users' page-leaving behaviors, Liu *et al.* analyzed page-visit stay time for 205,873 different pages for which they had captured upwards of 10,000 visits, and showed that the time users spend on a web page follows Weibull distribution [169]. Based on their finding, we can assume that the stay time[13] of users on our deployed site also follows Weibull distribution. We further calculate the stay rate of users in the duration of a circuit, and then compute the success rate of the circuit, i.e., all intermediate nodes and the responder stay in the network till data transmission via the circuit is completed.

The cumulative distribution function for the Weibull distribution is

$$\omega(t|k,\lambda) = 1 - \exp\left(-\left(\frac{t}{\lambda}\right)^k\right) t \geq 0 \tag{3.16}$$

[13]In this chapter, we treat Dwell time studied in Liu *et al.*'s work as the stay time of users on a page.

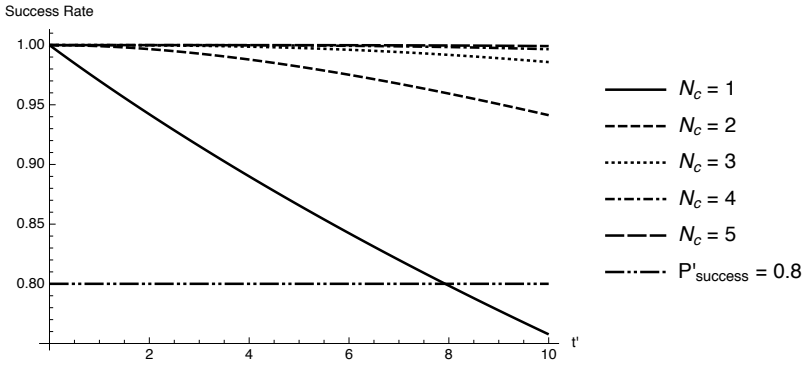


Figure 3.21: The success rate increases when the number of created circuits increases.

with t, λ and k are the stay time (dwell time), the scale and shape parameters, respectively. As shown in Figure 3.19, the longer of the average stay time is, the larger λ is and the smaller k is. For a $(l+2)$-node circuit created at time T, we assume that the duration of the circuit is t' and the average number of page visits is N_v. It takes t' seconds for the circuit to complete data transmission, and thus the minimal duration requirement for all nodes in this circuit is t'. For users who visit the site t seconds ahead the creation of the circuit, $(1-\omega(t+t'|k,\lambda))\,N_v$ of them are remaining after the teardown of the circuit, and $(1-\omega(t|k,\lambda))\,N_v$ users stay on the page at least till T. Since the peer server selects nodes for the circuit at T, we can compute the stay rate for users existing at T and lasting for t' below:

$$P_{stay}(t'|k,\lambda) = \frac{\mathsf{INIT}_0^{+\infty}\,(1-\omega(t+t'|k,\lambda))\,N_v\,dt}{\mathsf{INIT}_0^{+\infty}\,(1-\omega(t|k,\lambda))\,N_v\,dt} \tag{3.17}$$

Furthermore, the probability that all intermediate nodes and the responder stay for t' or the success rate of the circuit is[14]:

$$P_{success}(t', l|k,\lambda) = P_{stay}(t'|k,\lambda)^{l+1} \tag{3.18}$$

Based on the study of Liu *et al.* [169], λ for 80% of stay-time distributions is no more than 70, and the median value of k vary from 0.65 to 0.80. Figure 3.20 shows that when the duration t' and k are larger, or λ is smaller, then the stay

[14]We assume that the number of peers for the creation of a circuit is large enough to ignore the probability that the same peer is selected as the intermediate node for multiple circuits.

rate is lower, more users existing at T may leave before the teardown of the circuit. For $\lambda = 70$ and $k = 0.65, 0.80$ (representing that the median stay time is 39.8 s or 44.3 s[15]), the stay rate is always over 90% when the duration of the circuit is 9.42 s (i.e., the fetching time of a 2 MB resource from the remote server). Figure 3.17 presents that with $\lambda = 70$ and $k = 0.65$ (the median stay time is 39.8 s), the longer length of the circuit may cause the success rate of data transmission lower. For a 4-node circuit, the success rate is always over 75% even if $t' = 10$. If the duration t' is short enough (e.g., 2 s), the success rate is always over 90%, which means over 90% circuits can be successfully completed. For the retrofitted site of APAC, the site can incentivize users to stay longer on the page, then the stay-time distribution can have larger λ and smaller k, further the success rate becomes higher, e.g., over 95% when $\lambda = 120$ and $k = 0.35$ (the median stay time is 42.1 s) in Figure 3.18.

Once the site operator observes that the distribution of stay time does not provide a proper success rate for a circuit (e.g., 80%), the site operator can suspend the circuit-based data transmission and reuse the client-server mode. Alternatively, as a relaxation of anonymity, the site operator can create several backup circuits for one request, and the complete data transmission through any circuit is counted as the success for the request. In this case, the success rate can be represented as:

$$P'_{success}(N_c, t', l|k, \lambda) = 1 - (1 - P_{success}(t', l|k, \lambda))^{N_c} \tag{3.20}$$

with the number N_c of circuits for one request. Figure 3.21 shows that with $\lambda = 70$, $k = 0.65$ and $l = 2$, 1 backup circuit[16](in total 2 circuits) for one request can drastically increase the success rate (e.g., over 90%). For over 2

[15] The median stay time for Weibull distribution is:

$$T_{median}(k, \lambda) = \lambda (\ln 2)^{\frac{1}{k}} \tag{3.19}$$

[16] In Figure 3.21, we use the same-length circuits as backup circuits for one request. In fact, the peer server can set up backup circuits of any length (not larger than L_{max}).

backup circuits, the success rate of one request is always over 99% even the duration is 10 s. In the worst case, as a relaxation of compatibility, the site can employ APAC in an extension and ask users to install it. The extension checks whether the client is in any circuit whenever the client is attempting to leave APAC. If so, the extension will take over and complete data transmission for the pending circuits.

3.7 Related Work

In this section, we discuss recent work related to security & privacy in peer-assisted CDNs and anonymous communication systems.

3.7.1 Security & Privacy in Peer-assisted CDNs

Numerous peer-assisted CDNs have been proposed in recent years [2, 37, 45, 62, 130, 134, 135, 149, 218, 222, 228, 231]. In contrast to traditional infrastructure-based CDNs, peer-assisted CDNs offload content delivery tasks on clients (peers) to save the bandwidth of servers [147, 156], and reduce the latency of fetching content at the client side. For instance, NetSession can offload $70 - 80\%$ of the traffic to the peers [232]. For the thorough evaluation on Etsy, Maygh is able to reduce the 95th-percentile bandwidth due to image content at the operator by over 75% [231].

Meanwhile, researchers also propose solutions to preserve the integrity and authenticity of content [62, 218, 231]. For example, FireCoral introduces *signing service* and *tracker* components to authenticate and verify content [218]. All content in Maygh is self-certifying [231]. Aditya et al. proposed RCA for NetSession to detect and quarantine malicious clients [62]. On the other hand, no systematic studies of inference attacks on peer-assisted CDNs have been conducted yet and few defenses against such attacks are deployed on peer-assisted CDNs. In this work, we systematically analyze inference attacks on

System's Name	No Installation	Initiator Anonymity.	Responder Anonymity.	Locality-aware
Onion Routing/Tor-based Systems [64, 123, 204, 223]	✗	✓	✗	Partially
Crowds [205], & Morphmix [207], etc. [136, 178–181, 187]	✗	✓	✗	✗
Hidden Service [48], I2P [22] & Freenet [18], etc. [53, 71, 148, 209]	✗	✓	✓	✗
APAC	✓	✓	✓	✓

Table 3.2: Comparison with low-latency anonymous systems.

peer-assisted CDNs, and have mounted attacks on three popular systems, i.e., Swarmify, BemTV and P2PSP. Furthermore, we propose APAC to mitigate this class of attacks with minor performance overhead. We raise the bar significantly beyond what the current peer-assisted CDNs have.

3.7.2 Anonymous Communication Systems

As we have discussed in Chapter 2, researchers have proposed numerous anonymous communication systems to provide anonymity for users as shown in Table 3.2. Nevertheless, the primary goal for these approaches is to preserve high level of anonymity. They typically require users to install client-side software and are not locality-aware by default[17]. In Tor-based approaches, typically the initiator creates/constructs the circuit and selects the intermediate nodes for each request. Therefore these approaches preserve initiator anonymity, but cannot hide the responder's identity. Furthermore, together with the peer-to-peer anonymous communication systems, these systems introduce non-negligible circuit setup latency when the initiator indirectly communicates with intermediate nodes to set up the circuit. For example, the circuit setup latency for a 4-hop circuit in ShadowWalker is 1820 ms [179]. Therefore, we cannot directly apply previous mechanisms (as shown in Table 2.1) to achieve our predefined

[17]The relay selection algorithm in Tor can be adjusted to be locality-aware [64].

three goals. In APAC, we utilize the peer server (instead of peers) to construct the circuit for each request, which avoids the non-negligible overhead and preserves responder anonymity. Further, based on the locality information of all peers maintained by the server, we optimize the onion routing's circuit selection algorithm in APAC by introducing distribution factors, which can be tuned to locate intermediate peers nearby the initiator/responder to reduce network latency. Comparing to previous low-latency approaches, APAC can achieve all the primitives listed in Table 3.2.

3.8 Summary

In this chapter, we systematically study inference attacks on peer-assisted CDNs. Further, we developer an anonymous peer-assisted CDN called APAC, which preserves a high degree of anonymity that is significantly beyond what current peer-assisted CDNs have. Our performance evaluation shows that the locality-aware APAC can bring desired network latency reduction for content fetching and bandwidth savings for deployed sites.

Chapter 4

OBLIVP2P: An Oblivious Peer-to-Peer Content Sharing System

4.1 Introduction

In this chapter, we cope with such global adversary in the generic content sharing P2P systems, in which web overlays are included. In the last chapter, we consider a partial adversary, but a global adversary capable of long-term traffic analysis is feasible in real-world P2P systems. Content sharing P2P systems, especially P2P file-sharing applications such as BitTorrent [7], Storj [44] and Freenet [18] are popular among users for sharing files on the Internet. More recently, peer-assisted CDNs such as Akamai Netsession [1] and Squirrel [149] are gaining wide adoption to offload web CDN traffic to clients. The convenient access to various resources attract millions of users to join P2P networks, e.g., BitTorrent has over 150 million active users per month [56] and its file-sharing service contributes 3.35% of all worldwide bandwidth [36]. However, the majority of such P2P applications are susceptible to long-term traffic analysis through global monitoring; especially, analyzing the *pattern* of commu-

nication between a sender and a receiver to infer information about the users. For example, many copyright enforcement organizations such as IFPI, RIAA, MPAA, government agencies like NSA and ISP's are reported to globally monitor BitTorrent traffic to identify illegal actors. Monitoring of BitTorrent traffic has shown to reveal the data requested and sent by the peers in the network [164, 200, 212]. Unfortunately, while detecting copyright infringements is useful, the same global monitoring is applicable to *any* user of the P2P network, and can therefore collect benign users' data. Thus, users of such P2P systems are at a risk of leaking private information such as the resources they upload or download.

To hide their online traces, users today employ anonymous networks as a solution to conceal their digital identities or data access habits. Currently, anonymous networks include Mix networks [100, 117, 183], and Onion routing/Tor-based systems [64, 123, 204, 223], as well as other P2P anonymity systems [136, 178, 179, 187, 205, 207]. Such systems allow the user to be *anonymous*, so that the user is unidentifiable within a set of users [199].

Although above solutions provide an anonymity guarantee, they are vulnerable to long-term traffic pattern analysis attacks, which is an important threat for P2P systems like BitTorrent [63, 127, 128, 159, 224, 225]. Researchers have demonstrated attacks targeting BitTorrent users on top of Tor that reveal information related to the resources uploaded or downloaded [8, 80]. Such attacks raise the question - *is anonymizing users the right defense against traffic pattern analysis in P2P content sharing systems?*

In this chapter, we investigate a new approach to solve the problem of persistent analysis of data communication patterns. We advocate that data / resource access pattern hiding is an important and necessary step to thwart leakage of users data in P2P systems. To this end, we present a first candidate solution, OBLIVP2P— an oblivious protocol for peer-to-peer content sharing systems. Hiding data access patterns or making them *oblivious* unlinks user's identity

from her online traces, thereby defending against long-term traffic monitoring.

4.1.1 Approach

For hiding data access patterns between a trusted CPU and an untrusted memory, Goldreich and Ostrovsky proposed the concept of an Oblivious RAM (ORAM) [142]. We envision providing similar obliviousness guarantees in P2P systems, and therefore select ORAM as a starting point for our solution. To the best of our knowledge, OBLIVP2P is the first work that adapts ORAM to accesses in a P2P setting. However, directly employing ORAM to hide access patterns in a P2P system is challenging. We outline two key challenges in designing an oblivious and a scalable P2P protocol using ORAM.

Obliviousness. The first challenge arises due to the difference in the setting of a standard ORAM as compared to a P2P content sharing system. Classical ORAM solutions consists of a single client which securely accesses an untrusted storage (server), wherein the client is eventually the owner and the only user of the data in the memory. In contrast, P2P systems consist of a set of trusted trackers managing the network, and multiple data owners (peers) in the network. Each peer acts both as a client as well as a server in the network i.e., a peer can either request for a data or respond to other peer's request with the data stored on its machine. Hence, adversarial peers present in the network can see the plaintext and learn the data requested by other peers, a threat that does not exists in the traditional ORAM model where only encrypted data is seen by the servers.

Scalability. The second challenge lies in seeking an oblivious P2P system that 1) the throughput scales linearly with the number of peers in the network, 2) has no centralized bottleneck and 3) can be parallelized with an overall acceptable throughput. In standard ORAM solutions, the (possibly distributed) server is responsible for serving all the data access requests from a client one-by-one. In contrast, P2P systems operate on a large-scale with multiple peers (clients) requesting resources from each other simultaneously without overloading a par-

ticular entity. To retain scalability of P2P systems, it is necessary to ensure that requests can be served by distributing the communication and computation overhead.

Solution Overview. We start with a toy construction (OBLIVP2P-0) which directly adapts ORAM to a P2P setting, and then present our main contribution which is a more efficient solution (OBLIVP2P-1).

*Centralized Protocol (*OBLIVP2P-*0):* Our centralized protocol or OBLIVP2P-0, is a direct adaptation of ORAM in a P2P system. The peers in the network behave both like distributed storage servers as well as clients. They request a centralized, trusted tracker to access a particular resource. The tracker performs all the ORAM operations to fetch the resource from the network and returns it to the requesting peer. However, this variant of OBLIVP2P protocol has limited scalability as it assigns heavy computation to the tracker, making it a bottleneck.

*Distributed Protocol (*OBLIVP2P-*1):* As our main contribution, we present OBLIVP2P-1 which provides both obliviousness and scalability properties in a tracker-based P2P system. To attain scalability, the key idea is to avoid any single entity (say the tracker) as a bottleneck. This requires distributing all the ORAM operations for fetching and sharing of resources among the peers in the network, while still maintaining obliviousness guarantees. To realize such a distributed protocol, our main building block, which we call *Oblivious Selection* (OblivSel), is a novel combination of private information retrieval with recent advances in ORAM. Oblivious Selection gives us a scalable way to securely distribute the load of the tracker. Our construction is proven secure in the honest-but-curious adversary model. Constructions and proofs for arbitrarily malicious fraction of peers is slated for future work.

4.1.2 System and Results

We provide a prototype implementation of both OBLIVP2P-0 and OBLIVP2P-1 protocols in Python. Our source code is available online [32]. We exper-

imentally evaluate our implementation on DeterLab testbed with 15 servers simulating up to 2^{14} peers in the network. Our experiments demonstrate that OBLIVP2P-0 is limited in scalability with the tracker as the main bottleneck. The throughput for OBLIVP2P-1, in contrast, scales linearly with increase in the number of peers in the network. It attains an overall throughput of 3.19 MBps for a network of 2^{14} peers that corresponds to 7 requests per second for a block size of 512 KB. By design, OBLIVP2P-1 is highly parallelizable over the computational capacity available in a real P2P network. Further, our protocol exhibits no bottleneck on a single entity in experiment, thereby confirming that the network and the computational overhead can be completely offloaded to the P2P network.

Contributions. We summarize our contributions below:

- **Problem Formulation.** We formulate the problem of making data access pattern oblivious in P2P systems. This is a necessary and important step in building defenses against long-term traffic analysis.
- **New Protocols.** We propose OBLIVP2P— a first candidate for an oblivious peer-to-peer protocol in content sharing systems. Our main building block is a primitive which we refer to as oblivious selection that makes a novel use of recent advances in Oblivious RAM combined with private information retrieval techniques.
- **System Implementation & Evaluation.** Our prototype implementation is available online [32]. We experimentally evaluate our protocol to measure the overall throughput of our system, latency for accessing resources and the impact of optimizations on the system throughput.

4.2 Problem

Many P2P applications are not designed with security in mind, making them vulnerable to traffic pattern analysis. We consider BitTorrent as our primary

case study. However, the problem we discuss is broadly applicable to other P2P file sharing systems like Gnutella [20], Freenet [18] and Storj [44] or peer-assisted CDNs such as Akamai Netsession [1], Squirrel [149] and APAC [150].

4.2.1 BitTorrent: A P2P Protocol

The BitTorrent protocol allows sharing of large files between users by dividing it into blocks and distributing it among the peers. It has a dynamic network, made up of a number of nodes that join the network and volunteer themselves as peers. Each peer holds data blocks in its local storage and acts both as a client / requester and server / sender simultaneously. There exists a tracker that tracks which peers are downloading / uploading which file and saves the state of the network. It keeps information regarding the position or the IP addresses of peers holding each resource but does not store any real data blocks. A peer requests the tracker for a particular resource and the tracker responds with a set of IP addresses of peers holding the resource. The requester then communicates with these IP addresses to download the blocks of the desired resource. The peers interact with each other using a P2P protocol[1]. The requester concatenates all the blocks received to construct the entire resource.

4.2.2 Threat Model

In our threat model, we consider the tracker as a trusted party and peers as passive honest-but-curious adversaries i.e., the peers are expected to correctly follow the protocol without deviating from it to learn any extra information. In P2P systems including CDNs (content delivery networks) and BitTorrent, passive monitoring is already a significant threat on its own. We consider the following two types of adversaries:

[1]We want to emphasize that there are other models of P2P networks without tracker based on DHT that we are not addressing in this work.

Global Passive Adversary. Since BitTorrent traffic is public, there exist tools like Global BitTorrent Monitor [42] or BitStalker [72] that support accurate and efficient monitoring of BitTorrent. Previous research has shown that any BitTorrent user can be logged within a span of 3 hours, revealing his digital identity and the content downloaded [104]. Further, the adversary can log the communication history of the network traffic to perform offline analysis at a later stage. Hence, we consider it rational to assume the presence of a global adversary with the capability to observe long term traffic in the network.

Passive Colluding Peers. Some of the peers in the P2P network can be controlled by the global adversary. They can further collude to exchange data with other adversarial peers in the system. While colluding these "sybil" peers can share information such as observed / served requests and the contents stored at their local storage. Their goal is to collectively glean information about other peers in the network. A formal definition of passive colluding peers is as follows:

Definition 4.2.1. *(Passive Colluding peers) We say that a peer* P_i passively *colludes with peer* P_j *if both peers share their views without any modification, where a view consists of: a transcript of the sequence of all accesses made by* P_i*, a partial or total copy of peer's private storage, and a transcript of the access pattern induced by the sequence of accesses. We denote by* $\mathcal{C}(P_i)$ *the set of colluding peers with* P_i*.*

Note that from the above definition, we have a symmetric relation such that if $P_i \in \mathcal{C}(P_j)$ for $i \neq j$, then $P_j \in \mathcal{C}(P_i)$ and further $\mathcal{C}(P_i) = \mathcal{C}(P_j)$. It follows that if $P_i \notin \mathcal{C}(P_j)$, then $\mathcal{C}(P_i)$ and $\mathcal{C}(P_j)$ are disjoint.

Our protocol tolerates a fraction of c adversarial peers in the network such that $c \in O(N^{\epsilon})$, where N is the total number of peers in the network and $\epsilon < 1$. Although the P2P network undergoes churn, we assume the fraction of adversarial peers c remains within the asymptotic bounds of $O(N^{\epsilon})$. Our choice of the upper bound for c ensures an exponentially small advantage to the attacker; for

an application that can tolerate higher attacker's advantage, a larger malicious fraction can be allowed.

4.2.3 Insufficiency of Existing Approaches

Existing techniques propose anonymizing users to prevent traffic pattern analysis attacks. However, these solutions are not sufficient to protect against a global adversary with long term access to communication patterns.

Unlinkability Techniques (e.g. Mixnet). Existing anonymity approaches "unlink" the sender from the receiver (see survey [115]). Chaum proposed the first anonymous network called mix network [100], which shuffles messages from multiple senders using a chain of proxy servers and sends them to the receiver. Another recent system called Riposte guarantees traffic analysis resistance by unlinking a sender from its message [109]. However, all these systems are prone to attack if an adversary can observe multiple request rounds in the network.

For example, consider that $Alice$ continuously communicates with Bob using a mixnet service. A global adversary observes this communication for a couple of rounds, and records the recipient set in each round. Let the senders' set consists of $S_1 = \{Alice, a, b, c\}$ and $S_2 = \{a', b', Alice, c'\}$, and the recipients' set consists of $R_1 = \{x, y, z, Bob\}$ and $R_2 = \{x', y', Bob, z'\}$ for rounds 1 and 2 respectively. The attacker can then infer the link between sender and receiver by intersecting $S_1 \cap S_2 = \{Alice\}$ and $R_1 \cap R_2 = \{Bob\}$. The attacker learns that Alice is communicating with Bob, and thus breaks the unlinkability. This attack is called the *intersection, hitting set* or *statistical disclosure attack* [63, 159]. Overall, one time unlinkability is not a sufficient level of defense when the adversary can observe traffic for arbitrary rounds.

Path Non-Correlation (e.g. Onion routing). Another approach for guaranteeing anonymity is to route the message from a path such that the sender and the receiver cannot be correlated by a subset of passive adversarial nodes. Onion-routing based systems like Tor enable anonymous communication by us-

ing a sequence of relays as intermediate nodes (called circuit) to forward traffic [69, 123]. However, Tor cannot provide sender anonymity when the attacker can see both the ends of the communication, or if a global adversary observes the entire network. Hence, if an attacker controls the entry and the exit peer then the adversarial peers can determine the recipient identity with which the initiator peer is communicating [127, 128, 224, 225]. This is a well-known attack called the *end-to-end correlation attack* or *traffic confirmation attack* [49, 51].

4.2.4 Problem Statement

Our goal is to design a P2P protocol that prevents linking a user to a requested resource using traffic pattern analysis. Section 4.2.3 shows how previous anonymity based solutions are susceptible to attacks in our threat model. In this work, we address this problem from a new viewpoint, by making the communication pattern oblivious in the network. We advocate that hiding data / resource access pattern is a necessary and important step in designing traffic pattern analysis resistant P2P systems.

In a P2P system such as BitTorrent, a user accesses a particular resource by either downloading (Fetch) or uploading (Upload) it to the network. We propose to build an oblivious P2P content sharing protocol (OBLIVP2P) that hides the data access patterns of users in the network. We formally define an Oblivious P2P protocol as follows:

Definition 4.2.2. *(Oblivious P2P): Let* $(P_1, \cdots, P_n)$ *and* $\mathcal{T}$ *be respectively a set of* n *peers and a tracker in a* P2P *system. We denote by* $\overrightarrow{x_i} = (x_{i,1}, \cdots, x_{i,M})$ *a sequence of* M *accesses made by peer* P_i *such that* $x_{i,j} = (\mathsf{op_{i,j}}, \mathsf{fid_{i,j}}, \mathsf{file_{i,j}})$ *where* $\mathsf{op_{i,j}} = \{\mathsf{Upload}, \mathsf{Fetch}\}$*,* $\mathsf{fid_{i,j}}$ *is the filename being accessed, and* $\mathsf{file_{i,j}}$ *is the set of blocks being written in the network if* $\mathsf{op_{i,j}} = \mathsf{Upload}$*.*

We denote by $\mathcal{A}(\overrightarrow{x_i})$ *the access pattern induced by the access sequence* $\overrightarrow{x_i}$ *of peer* P_i*. The access pattern is composed of the memory arrays of all peers accessed while running the sequence* $\overrightarrow{x_i}$*. We say that a* P2P *is* oblivious *if for*

any two equal-length access sequences $\overrightarrow{x_i}$ *and* $\overrightarrow{x_j}$ *by two peers* P_i *and* P_j *such that*

- $P_j \notin \mathcal{C}(P_i)$
- $\forall k \in [M] : x_{i,k} = \mathsf{Fetch} \Leftrightarrow x_{j,k} = \mathsf{Fetch} \wedge x_{i,k} = \mathsf{Upload} \Leftrightarrow x_{j,k} = \mathsf{Upload}$
- $\forall k \in [M]$, $|\mathsf{file_{i,k}}| = |\mathsf{file_{j,k}}|$

are indistinguishable for all probabilistic poly-time adversaries except for $\mathcal{C}(P_i)$, $\mathcal{C}(P_j)$, *and tracker* $\mathcal{T}$.

Scope. OBLIVP2P guarantees resistance against persistent communication traffic analysis i.e., observing the path of communication and thereby linking a sender to a particular resource. OBLIVP2P does *not* prevent against:

a) *Active Tampering*: An adversarial peer can tamper, alter and deviate from the protocol to learn extra information. Admittedly, this can have an impact on obliviousness, correctness and availability of the network.
b) *Side Channels*: An adversary can monitor any peer in the system to infer its usage's habits via side channels: the number of requests, time of activity, and total number of uploads. In addition, an adversary can always infer the total file size that any peer is downloading or uploading to the P2P network. Literature shows that some attacks such as website fingerprinting can be based on the length of file requested by peers [191].
c) *Orthogonal Attacks*: Other attacks in P2P file sharing systems consist of threats such as poisoning of files by uploading corrupted, fake or misleading content [161] or denial of service attacks [129]. However, these attacks do not focus on learning private information about the peers and hence are orthogonal to our problem.

Admittedly, our assumption about honest-but-curious is less than ideal and simplifies analysis. We hope that our construction spurs future work on tackling the active or arbitrary malicious adversaries. Emerging trusted computing

primitives (e.g., Intel SGX [25]) or cryptographic measures [192] are promising directions to investigate. Lastly, OBLIVP2P should not be confused with traditional anonymous systems where a user is anonymous among a set of users. OBLIVP2P does not guarantee sender or receiver anonymity, but hides data access patterns of the users.

4.3 Our Approach

As a defense against traffic pattern analysis, we guarantee oblivious access patterns in P2P systems. We consider Oblivious RAM as a starting point.

4.3.1 Background: Tree-Based ORAM

Oblivious RAM, introduced by Goldreich and Ostrovsky [142], is a cryptographic primitive that prevents an adversary from inferring any information via the memory access pattern. Tree-based ORAM introduced by Shi et al. [210] offers a poly-logarithmic overhead which is further reduced due to improvements suggested in the follow up works [87, 112, 121, 182, 206, 213]. In particular, we use Ring ORAM, [206], one of the latest improvements for tree-based ORAM in our protocol. In Ring ORAM, to store N data blocks, the memory is organized in a (roughly) $\log N$-height full binary tree, where each node contains z real blocks and s dummy blocks. Whenever a block is accessed in the tree, it is associated to a new randomly selected leaf identifier called, tag. The client stores this association in a position map PosMap along with a private storage (stash). To read and write to the untrusted memory, the client performs an Access followed by an Evict operation described *at a high level* as follows:

- Access(adr): Given address adr, the client fetches the leaf identifier tag from PosMap. Given tag, the client downloads one block per every node in the path $\mathcal{P}(\mathsf{tag})$ that starts from the root and ends with the leaf tag. The

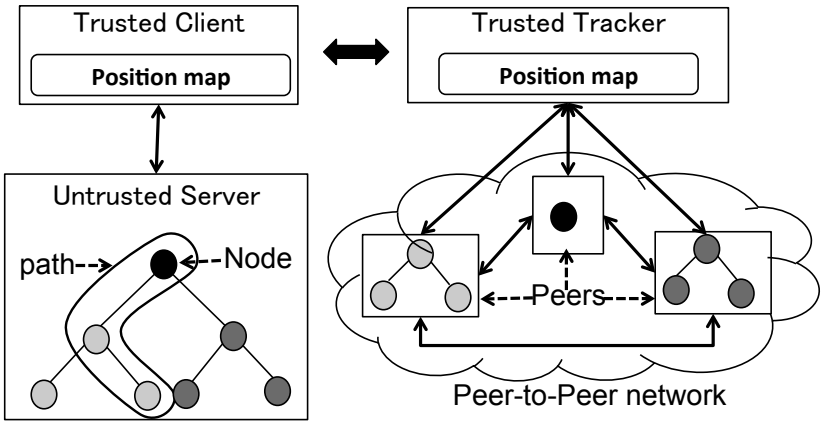


Figure 4.1: Mapping of a client / server ORAM model to a P2P system

client decrypts the retrieved blocks, and retrieves the desired block. This block is appended to the stash.

- Evict(A, ν): After A accesses, the client selects a path $\mathcal{P}(\nu)$ based on a deterministic reverse lexicographic order, downloads the path, decrypts it and appends it to the stash. The client runs the least common ancestor algorithm to sort the blocks as in [213]. Finally, the client freshly encrypts the blocks and writes them back to the nodes in the path.

The stash is upper bounded by $O(\log N)$. The overall bandwidth may reach $\simeq 2.5 \log N$, for N blocks stored. In Ring ORAM, eviction happens periodically after a controllable parameter $A = 2z$ accesses where z is the number of blocks in each bucket [206].

4.3.2 Mapping an ORAM to a P2P setting

We start from a traditional ORAM in a client / server model where the client is trusted and the server is not, and simulate it on a tracker / peers setting. In particular, we consider that the server's memory is organized in a tree structure, and we delegate every node in the tree to a peer. That is, a full binary tree of N leaves is now distributed among $N_p = 2N - 1$ peers (refer to Figure 4.1). In practice, many nodes can be delegated to many peers based on the storage capacity of each peer.

Contrary to the client / server setting where the client is the only one who can fetch, modify or add a block, in P2P, the peers can also request and add new blocks. In addition, the peers are volatile, i.e, many peers can join or leave the network. Moreover, from a security perspective, the network peers do not trust each other, and an adversarial peer can always be interested in finding out the block being retrieved by other peers. To avoid this, the tracker instructs the peers in a P2P system to save encrypted blocks in their local memory (different from the conventional BitTorrent model). Our construction ensures that the peer neither has the keys necessary to decrypt its storage nor can it collude with other adversarial peers to recover it. In this setting, we first present a strawman approach that guarantees our security goal but is restricted in terms of scalability.

4.3.3 OBLIVP2P-0 : Centralized Protocol

Almost all ORAM constructions are in a client / server setting and not designed for a P2P setting. A simple approach is to map the role of the trusted client in an ORAM setting (refer to Figure 4.1) to the trusted tracker in a P2P system. The client in ORAM is simulated by the trusted tracker (storing the position map, private keys and the stash) and the server by the untrusted peers (storing the encrypted blocks). With such a mapping from an ORAM model to a P2P setting, a peer (initiator) can request for a resource to the tracker. To access a particular resource, the tracker fetches the blocks from a path in the tree and decrypts them to get the desired block. It then returns the requested resource to the initiator peer. This simple *plug-&-play* construction satisfies all our P2P security requirements.

In OBLIVP2P-0, the trusted tracker behaves as the client in traditional ORAM model. Whenever a peer requests a block, the tracker performs all the ORAM access work, and then sends the plaintext block to the initiator. The tracker downloads the path composed of a logarithmic number of nodes, writes back the path with a fresh re-encryption before routing the block to the initiator. As

long as the tracker is trusted, this ensures the obliviousness property of peers' accesses, as stated by definition 4.2.2.

Upload algorithm. To upload a file, the peer divides it into data blocks and sends the blocks to the tracker. The tracker appends the block to the stash stored locally while generating new random tags. The tracker updates accordingly TagMap, and FileMap (refer to Table 4.1).

Fetch algorithm. To fetch a file, the peer sends the file identifier, as an instance a filename, to the tracker. The tracker fetches from the FileMap and TagMap the corresponding blocks and sends requests to the corresponding peers to retrieve the blocks, following the Ring ORAM Access protocol. For every retrieved block, the tracker sends the plaintext block to the requesting peer.

Sync algorithm. The synchronization happens after every $A \simeq 2z$ accesses [206] (e.g., nearly 8 accesses) at which point the tracker evicts the stash.

Tracker as Bottleneck. In OBLIVP2P-0, the tracker has to transmit / encrypt a logarithmic number of blocks on every access. The tracker requires a bandwidth of $O(\log N \cdot B)$ where B is the block size and the computation cost of $O(\log N \cdot E)$ where E is time for encrypting / decrypting a block. Moreover, our evaluation in Section 4.5 shows that the eviction step is network-intensive. In a P2P setting with large number of accesses per second, the tracker creates a bottleneck in the network.

4.3.4 OBLIVP2P-0 Analysis

Our analysis follow from Ring ORAM construction. To access a block the tracker has to transmit $\sim 2.5 \log N \cdot B$ bits per access. During a block access or eviction, any peer at any time transmits $O(B)$ bits. The tracker's main computational time consists of decrypting and encrypting the stash. Since the stash has a size of $O(\log N)$ blocks, the tracker does $O(\log N)$ blocks encryption/decryption. In terms of storage, every peer has $(z + s)$ blocks to store, where z is

number of real blocks and s is a parameter for dummy blocks. From a security perspective, it is clear that if there are two sequences verifying the constraints of Definition 4.2.2, a malicious peer monitoring their access pattern cannot infer the retrieved blocks, since after every access the block is assigned to a random path in the simulated ORAM.

4.4 OBLIVP2P-1: Distributed Protocol

In this section, we describe our main contribution, OBLIVP2P-1 protocol that provides both security and scalability properties. In designing such a protocol, our main goal is to avoid any bottleneck on the tracker i.e., none of the real blocks should route through the tracker for performing an access or evict operations of ORAM. We outline the challenges in achieving this property while still retaining the obliviousness in the network.

4.4.1 Challenges

First Attempt. A first attempt to reduce tracker's overhead is to modify OBLIVP2P-0 such that the heavy computation of fetching the path of a tree and decrypting the correct block is offloaded to the initiator peer. On getting a resource request from a peer, the tracker simply sends information to the peer that includes the path of the tree to fetch, the exact position of the requested block and the key to decrypt it. However, unlike standard ORAM, the peer in our model is not trusted. Giving away the exact position of the block to the initiator peer leaks additional information about the requested resource in our model, as we explain next.

Recall that in a tree-based ORAM, blocks are distributed in the tree such that the recently accessed blocks remain in the top of the tree. In fact, after every eviction the blocks in the path are pushed down as far as possible from the root of the tree. As an instance, after N deterministic evictions, all blocks that were

never accessed are (very likely) in the leaves. Conversely, consider that an adversarial peer makes two back-to-back accesses. In the first access, it retrieves a block from the top of the tree while in the second access it retrieves a block from a leaf. The adversarial peer (initiator) learns that the first block is a popular resource and is requested before by other peers while the second resource is a less frequently requested resource. This is a well known issue in tree-based ORAM, and is recently formulated as the *block history* problem [208]. Disclosing the block position, while hiding the scheme obliviousness requires to address the block history challenge in ORAM. Unfortunately, an ORAM hides the block history only if the communication spent to access a block dominates the number of blocks stored in the entire ORAM. This would be asymptotically equivalent to downloading the entire ORAM tree from all the peers. We refer readers to [208] for more details.

Second Attempt. Our second attempt is a protocol that selects a block while *hiding* the block position from the adversary i.e., to hide which node on the path holds the requested block. Note that in a tree-based ORAM, disclosing the path does not break obliviousness, but leaking which node on the path holds the requested block is a source of leakage. One trick is to introduce a circuit, a set of peers from the P2P network, that will simulate the operations of a mixnet. That is, the peers holding the path of the tree send their content to the first peer in the circuit, who then applies a random permutation, adds a new encryption layer, and sends the permuted path to the second peer and so on. The tracker, who knows all the permutations, can send the final block position (unlinked from original position) to the initiator, along with the keys to decrypt the block. The mixing guarantees that the initiator does not learn the actual position of the block. We note that mixing used here is for only one accessed "path", which is already randomized by ORAM. Hence, it is not susceptible to intersection attack discussed in Section 4.2.3. Finally, the initiator then peels off all layers of the desired block to output the plaintext block.

Structure	Mapping	Purpose
FileMap	file id fid to block addresses $\{\mathsf{adr}_i\}_{i\in[\frac{f}{B}]}$	Blocks identification
TagMap	block address adr to tag $\xleftarrow{\$} [N_B]$	Path identification
NetMap	peer id pid to network info $(\mathsf{IP}, \mathsf{port}) \in \{0,1\}^{128+16}$	Network representation
PosMap	block address adr to path and bucket position $\mathsf{pos} \in [N_p] \times [L \cdot z + \lvert\mathsf{stash}\rvert]$	Block exact localization
KeyMap	block address adr to key value $\mathsf{k} \xleftarrow{\$} \mathbb{Z}_q$	Input of key block generation
StashList	peers' identifiers $\{\mathsf{pid}_i\}_{i\in[\lvert\mathsf{stash}\rvert]}$	Stash localization

Table 4.1: Various meta-information contained in the state s, for OBLIVP2P-0 and OBLIVP2P-1. B is the block size in bits, N_P the number of peers, N_B number of blocks, L the path length, and z the bucket size.

However, there is an important caveat remaining in using this method. Note that the initiator has the keys to peel off all the layers of encryption and hence it has access to the same encrypted block fetched from the path in the tree. Thus, it can determine which peer's encrypted block was finally selected as the output of the mixnet. Hence, delegating the keys to the initiator boils down to giving her the block position. One might think of eliminating this issue by routing the block through the tracker to peel off all layers, but this will just make the tracker again a bottleneck.

So far, our attempts have shown limitations, but pointed out that there is a need to formally define the desired property. Considering a tracker, the initiator, and the peers holding the path, we seek a primitive that given a set of encrypted blocks, the initiator can get the desired plaintext block, while no entity can infer the block position but the tracker. We refer to this primitive as *Oblivious Selection* (OblivSel) and describe it next.

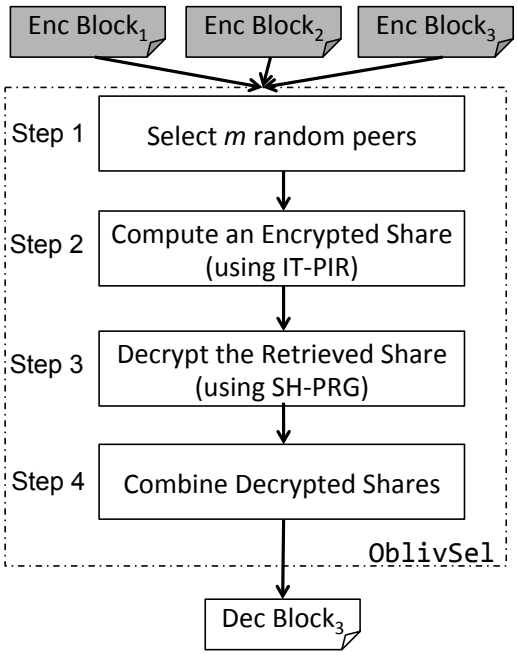


Figure 4.2: Oblivious Selection protocol using IT-PIR and Seed Homomorphic PRG as base primitives

4.4.2 Oblivious Selection

4.4.2.1 Definitions

We define OblivSel and its properties as follows:

Definition 4.4.1. *(Oblivious Selection).* OblivSel *is a tuple of two probabilistic algorithms* $(\mathsf{Gen}, \mathsf{Select})$ *such that:*

- $(\vec{\sigma}, \vec{r}) \leftarrow \mathsf{Gen}(\mathsf{k}, \mathsf{pos})$*: a probabilistic algorithm run by the tracker, takes as input a key* k *and the block position* pos*, picks uniformly at random* m *peers* $(P_1, \cdots, P_m)$*, and outputs* $(\vec{\sigma}, \vec{r})$ *where* $\vec{\sigma} = \{\sigma_1, \cdots, \sigma_m\}$ *and* $\vec{r} = \{r_1, \cdots, r_m\}$ *such that* (σ_i, r_i) *is given to the* i*th peer* P_i*.*

- $\Delta \leftarrow \mathsf{Select}(\vec{\sigma}, \vec{r}, \mathsf{Enc}(\mathsf{k}_1, \mathsf{block}_1), \cdots, \mathsf{Enc}(\mathsf{k}_\mathsf{L}, \mathsf{block}_\mathsf{L}))$*: a probabilistic algorithm run by* m *peers, takes as input* $\vec{\sigma}$*,* $\vec{r}$*, and a set of encrypted blocks* $\mathsf{Enc}(\mathsf{k_i}, \mathsf{block_i})$*, for* $i \in [\mathsf{L}]$*, and outputs the value* Δ*.*

Definition 4.4.2. OblivSel*, is* ***correct****, if*

$$\Pr[\forall\ \mathsf{pos} \in [\mathsf{L}], \mathsf{k} \in \{0,1\}^{\lambda}, (\vec{\sigma}, \vec{r}) \leftarrow \mathsf{Gen}(\mathsf{k}, \mathsf{pos});$$
$$\Delta \leftarrow \mathsf{Select}(\vec{\sigma}, \vec{r}, \mathsf{Enc}(\mathsf{k}_1, \mathsf{block}_1), \cdots, \mathsf{Enc}(\mathsf{k}_{\mathsf{L}}, \mathsf{block}_{\mathsf{L}}));$$
$$\Delta = \mathsf{Dec}(\mathsf{k}, \mathsf{Enc}(\mathsf{k}_{\mathsf{pos}}, \mathsf{block}_{\mathsf{pos}}))] = 1$$

For instance, if $(\mathsf{Enc}, \mathsf{Dec})$ is a private key encryption, OblivSel returns a decrypted block when the key given as input to the Gen function is the same as the private key of the block i.e., $\Delta = \mathsf{block}_{\mathsf{pos}}$ if $\mathsf{k} = \mathsf{k}_{\mathsf{pos}}$.

Definition 4.4.3. *(Position Hiding.) We say that* OblivSel *is a* position hiding *protocol if for all probabilistic polynomial time global adversaries, including the initiator and the* m *peers, guess the position of the block* pos *with a negligible advantage in the implicit security parameter.*

4.4.2.2 OblivSel Overview

The intuition for constructing OblivSel stems from the fact that the tracker cannot give the position or private key of the desired block to the peers in the network.

To privately select a block from the path without leaking its position, we propose to use an existing cryptographic primitive, called information-theoretical private information retrieval (IT-PIR) [103]. IT-PIR requires a *linear* computation proportional to the data size that makes it expensive to use for real time settings. However, note that in our setting, we want to obliviously select a block from a logarithmic number of blocks (i.e., a path of the tree). Thus, applying IT-PIR over tree-based ORAM comes with significant computational improvement, hence making it practical to use in our protocol. The high level idea is to apply IT-PIR primitive only on one path since the obliviousness is already guaranteed by the underlined tree-based ORAM construction.

Figure 4.2 shows the steps involved in our OblivSel primitive. As a first step,

the tracker randomly samples m peers from the network. For a bounded number of colluding adversarial peers in the system, this sample will contain at least one honest peer with high probability. The blocks of the path are fetched by all of the m peers. Each of the m peers then locally computes an encrypted share of the desired block using IT-PIR from the set of input blocks. Note that the tracker must not download the shares or it will violate our scalability requirement. On the other hand, we require to decrypt the block without giving away the private key to the network's peers. For this purpose, we make use of a second cryptographic primitive — a seed homomorphic pseudo-random generator (SH-PRG) [81]. The tracker generates a valid key share for each of the m peers to be used as seeds to the PRG function. Each peer decrypts (or unblinds) its encrypted share using its own key share such that the combination of decrypted shares results in a valid decryption of the original encrypted block in the tree. This property is ensured by SH-PRG and explained in detail in Section 4.4.2.3. Finally, each peer submits its decrypted share to the initiator peer who combines them to get the desired plaintext block. The colluding peers cannot recover the private key or the encrypted block since there is at least one honest peer who does not disclose its private information. This solves the issues raised in our second attempt.

Remark. OblivSel primitive can be used as a black box in different settings such as distributed ORAMs to decrease the communication overhead. We further show in Section 4.4.2.4 that OblivSel is highly parallelizable and can leverage peers in the network such that the computation takes constant time.

4.4.2.3 Base Primitives

Information-theoretic PIR. Information-theoretic private information retrieval (IT-PIR) [103] is a cryptographic primitive that performs oblivious read operations while requiring multiple servers $m \geq 2$. In the following, we present the details of one of the first constructions of IT-PIR by Chor et al. [103] which

Algorithm 2: IT-PIR protocol by Chor et al. [103]

$(r_1, \cdots, r_m) \leftarrow \mathsf{Query}(\mathsf{q}, \mathsf{L}, \mathsf{pos})$

- randomly generate $m-1$ random vectors such that $r_i \xleftarrow{\$} Z_q^{\mathsf{L}}$
- compute r_m such that for all $j \in [\mathsf{L}] \setminus \{\mathsf{pos}\}$, set $r_{m,j} = -\sum_{k=1}^{m-1} r_{k,j}$, otherwise, $r_{m,\mathsf{pos}} = 1 - \sum_{k=1}^{m-1} r_{k,j}$

$R_i \leftarrow \mathsf{Compute}(\mathsf{r_i}, \mathsf{DB})$

- parse database such as $DB = (\mathsf{block}_1, \cdots, \mathsf{block}_{\mathsf{L}})$
- compute $R_i = \sum_{j=1}^{\mathsf{L}} r_{i,j} \mathsf{Block_j}$

$\mathsf{block}_{\mathsf{pos}} \leftarrow \mathsf{Recover}(\mathsf{R}_1, \cdots, \mathsf{R_m})$: compute $\mathsf{block}_{\mathsf{pos}} = \sum_{j=1}^{m} R_i$

is secure even when $m-1$ among m servers collude passively, i.e., the servers collude in order to recover the retrieved block while not altering the protocol. An IT-PIR is a tuple of possibly randomized algorithms $\mathsf{IT-PIR} = (\mathsf{Query}, \mathsf{Compute}, \mathsf{Recover})$. Query takes as an input the block position pos to be retrieved, and outputs an IT-PIR query for m servers. Compute runs independently by every server, takes as input the corresponding IT-PIR query and outputs a share. Recover takes as inputs all shares output by all m servers, and outputs the plaintext block. We give the construction in Algorithm 2.

Seed homomorphic PRG (SH-PRG). A seed homomorphic PRG, G, is a pseudorandom generator over algebraic group with the additional property that if given $G(s_1)$ and $G(s_2)$, then $G(s_1 \oplus s_2)$ can be computed efficiently. That is, if the seeds are in a group $(\mathbb{S}, \oplus)$, and outputs in $(\mathbb{G}, \otimes)$, then for any $s_1, s_2 \in \mathbb{S}$, $G(s_1 \oplus s_2) = G(s_1) \otimes G(s_2)$. We refer to [188] for more details.

Decryption / Re-encryption using SH-PRG. Leveraging the property of SH-PRG, we explain the encryption, decryption and re-encryption of a block in our protocol. Every block in the tree is encrypted as $\mathsf{Enc}(\mathsf{k}_1, \mathsf{block}) = \mathsf{block} + G(\mathsf{k}_1)$. The decryption of the block can be then represented as $\mathsf{block} = \mathsf{Dec}(\mathsf{k}_1, \mathsf{Enc}(\mathsf{k}_1, \mathsf{block})) = \mathsf{block} + \mathsf{G}(\mathsf{k}_1) - \mathsf{G}(\mathsf{k}_1)$ For re-encrypting the encrypted block with a different key k_2, the tracker decrypts the encrypted block with a new secret key of the form $\mathsf{k}_1 - \mathsf{k}_2$ such that, $\mathsf{Dec}(\mathsf{k}_1 - \mathsf{k}_2, \mathsf{Enc}(\mathsf{k}_1, \mathsf{block})) = \mathsf{block} + \mathsf{G}(\mathsf{k}_1) - \mathsf{G}(\mathsf{k}_1 - \mathsf{k}_2) = \mathsf{block} + \mathsf{G}(\mathsf{k}_2) = \mathsf{Enc}(\mathsf{k}_2, \mathsf{block})$.

Algorithm 3: OblivSel with seed-homomorphic PRG

1 $(\vec{\sigma}, \vec{r}) \leftarrow \mathsf{Gen}(\mathsf{k}, \mathsf{pos})$

- set $\vec{r} = (r_1, \cdots, r_m) \leftarrow \mathsf{IT-PIR.Query}(\mathsf{q}, \mathsf{L}, \mathsf{pos})$;
- set $\vec{\sigma} = (\sigma_1, \cdots, \sigma_m)$, s.t., $(\sigma_1, \cdots, \sigma_{m-1}) \xleftarrow{\$} \mathbb{S}^{m-1}$, and $\sigma_m = \mathsf{k} - \sum_{i=1}^{m-1} \sigma_i$;

$\mathsf{block} \leftarrow \mathsf{Select}(\vec{\sigma}, \vec{r}, \mathsf{DB})$ `// Every peer` P_i

- compute $Eshare_i \leftarrow \mathsf{IT-PIR.Compute}(\mathsf{r_i}, \mathsf{DB})$;
- set $Dshare_i = Eshare_i - G(\sigma_i)$;

  `// Initiator`
- compute $\Delta = \sum_{i=1}^{m} Dshare_i$;

4.4.2.4 OblivSel Instantiation

In the following, we present an instantiation of OblivSel. We consider a set of L encrypted blocks. Each block $\mathsf{block_i}$ is a vector of elements in a finite group $\mathbb{G}$ of order q. For every block, the key is generated at random from $\mathbb{Z}_q$. The tracker has to keep an association between the block key and its position. An algorithmic description is given in Algorithm 3.

The tracker runs the Gen algorithm, which takes as inputs the secret key k with which the block is encrypted and the block's position pos, and outputs a secret shared value of the key, $\vec{\sigma}$, as well as the IT-PIR queries, $\vec{r}$. Every peer P_i holds a copy of the L encrypted blocks and receives a share of the key, σ_i, as well as its corresponding query, $\mathsf{r_i}$. Next, every peer runs locally an $\mathsf{IT-PIR.Compute}$ on the encrypted blocks and outputs a share, $Eshare_i$. After getting the encrypted share $Eshare_i$, each peer subtracts the evaluation of the SH-PRG G on σ_i from $Eshare_i$ ($Eshare_i - G(\sigma_i)$) to get the decrypted share $Dshare$. Finally, initiator outputs the sum of all the $Dshare_i$'s received from the m peers to get the desired decrypted block. As long as there is one non-colluding peer among the m peers and G is a secure PRG, the scheme is *position hiding*.

Highly Parallelizable. Notice that, in Algorithm 3, each of the m peers performs scalar multiplications proportional to the number of encrypted input blocks. The encrypted blocks can be further distributed to different peers such that each

Scheme	Tracker bandwidth (bits)	Network bandwidth (# blocks)	Tracker # encryption	Network computational overhead	Network Storage overhead	Tracker storage # blocks
0	$O(\log N \cdot B)$	$O(1)$	$O(\log N \cdot E)$	–	$O(1)$	$O(\log N)$
1	$O\big(\log^3 N\big)$	$O(\frac{\log N}{N})$	–	$O(\frac{\log^4 N}{N} \cdot \mathcal{E})$	$O(\mathsf{burst})$	–

Table 4.2: Comparison of OBLIVP2P instantiation per access. B the block size, N the number of blocks in the network, E the overhead of a block encryption, $\mathcal{E}$ a multiplication in elliptic curve group, burst the number of versions.

peer performs constant number of scalar multiplications. Given the availability of enough peers in the network, OblivSel is extremely parallelizable and therefore provides a constant time computation.

OblivSel **as a building block.** OblivSel protocol can be used as a building block in our second and main OBLIVP2P-1. For fetching a block, an invocation of OblivSel is sufficient as it obliviously selects the requested block and returns it in plaintext to the initiator. Additional steps such as re-encrypting the block and adding it to stash are required to complete the fetch operation. The details of these steps are in Section 4.4.3.

However, the eviction operation in ORAM poses an additional challenge. Conceptually, an eviction consists of block sorting, where the tracker re-orders the blocks in the path (and the stash). Fortunately, our protocol can perform eviction by several invocation of OblivSel primitive. Given the new position for each block, the P2P network can be instructed to invoke OblivSel recursively to output the new *sorted* path. The encryption of blocks has to be refreshed, but this is handled within OblivSel protocol itself when refreshing the key, using seed homomorphic PRG. We defer the concrete details of performing oblivious eviction to Section 4.4.3.

4.4.3 OBLIVP2P-1: Complete Design

In a P2P protocol for a content sharing system the tracker is responsible for managing the sharing of resources among the peers in the network. To keep a consistent global view on the network, the tracker keeps some *state* information that we formally define below:

Definition 4.4.4. *P2P network's state consists of: (1) number of possible network connections per peer, and (2) a lookup associating a resource to a (set of) peer identifier.*

The tracker can store more information in the state depending on the P2P protocol instantiating the network. We start first by formalizing a P2P protocol.

Definition 4.4.5. *A P2P protocol is a tuple of four (possibly interactive) algorithms* $\mathsf{P2P} = (\mathsf{Setup}, \mathsf{Upload}, \mathsf{Fetch}, \mathsf{Sync})$ *involving a tracker,* $\mathcal{T}$*, and a set of peers,* $(P_1, \cdots, P_n)$*, such that:*

- $\mathsf{s}' \leftarrow \mathsf{Setup}(\mathsf{s}, \{\mathsf{pid}\})$*: run by the tracker* $\mathcal{T}$*, takes as inputs a state* s *and a (possibly empty) set of peers identifiers* $\{\mathsf{pid}\}$*, and outputs an updated state* s'*.*
- $\big(\mathsf{out}, (A'_1, \cdots, A'_m), \mathsf{s}'\big) \leftarrow \mathsf{Upload}\big((\mathsf{fid}, \mathsf{file}), (\mathsf{A}_1, \cdots, \mathsf{A}_\mathsf{m}), \mathsf{s}\big)$*: is an interactive protocol between an initiator peer, a (possibly randomly selected) set of* $m \geq 0$ *peers, and a tracker* $\mathcal{T}$*. The initiator peer has as input a file identifier* fid*, and the file* file*, the peers' input is memory array* A_i *each, while for the tracker its state* s*. The initiator's output is* $\mathsf{out} \in \{\perp, \mathsf{file}\}$*, the peers output each a modified local memory* A'_i*, while the tracker outputs an updated state* s'*.*
- $(\mathsf{file}, \perp, \mathsf{s}') \leftarrow \mathsf{Fetch}\big(\mathsf{fid}, (\mathsf{A}_1, \cdots, \mathsf{A}_\mathsf{m}), \mathsf{s}\big)$*: is an interactive protocol between an initiator peer, a (possibly randomly selected) set of* $m \geq 0$ *peers, and a tracker* $\mathcal{T}$*. The initiator peer has as input a file identifier* fid*, the peers' input is a memory array* A_i *each, while for the tracker its state*

s. *The initiator outputs the retrieved file* file*, each peer gets* $\perp$*, while the tracker outputs an updated state* s.

- $\big((A'_1, \cdots, A'_m), \mathsf{s}'\big) \leftarrow \mathsf{Sync}\big((\mathsf{A_1}, \cdots, \mathsf{A_m}), \mathsf{s}\big)$*: is an interactive protocol between the tracker and a (possibly randomly selected) set of* $m \geq 0$ *peers. The peers' input is a memory array* A_i *each, while for the tracker its state* s*. The peers output each a (possibly) modified memory array* A'_i*, while the tracker outputs an updated state* s'.

Note that a modification of a file already stored in the network is always considered as uploading a new file.

Setup Algorithm. In a P2P network, different peers have different storage capacities and hence we differentiate between the number of blocks, N_B, and the number of physical peers N_P. For this, we fragment the conceptual ORAM tree into smaller chunks where every peer physically handles a number of buckets depending on its local available storage. In addition, to keep a consistent global view on the network, the tracker keeps some *state* information. In OBLIVP2P-1, the state is composed of different meta-information that are independent of the block size: FileMap, PosMap, TagMap, NetMap, KeyMap, and StashMap. Table 4.1 gives more details about the metadata. The state also contains a counter recording the last eviction step, and $\sim \frac{B}{\log q}$ points sampled randomly from a q-order elliptic curve group $\mathbb{G}$ to be used for DDH seed homomorphic PRG, where B is the block size. The number of points in the generator needs to be equal to those in the data block. These points are publicly known to all peers in the network. The tracker randomly distributes the stash among the peers and records this information in the StashList.

Fetch Algorithm. The Fetch process is triggered when a peer requests a particular file. The tracker determines the block tag and position from its state for all the blocks composing the file. The m peers, the tracker, and the initiator runs OblivSel protocol such that the initiator retrieves the desired block. The OblivSel is invoked a second time to add a new layer to the retrieved block and send it

Algorithm 4: Fetch(fid, s): OBLIVP2P-1 fetch operation

Input: file id fid, and state s
Output: file {block}, and updated state s

```
// Initiator requests tracker for a file
{adr} ← FileMap(fid);
for adr in {adr} do
    (tag, pos) ← (TagMap(adr), PosMap(adr));
    k ← KeyMap(adr);
    compute (σ⃗, r⃗) := OblivSel.Gen(k, pos);
    set A = (stash, P(tag, 1), ··· , P(tag, L));
    // Initiator retrieves the block
    compute block := OblivSel.Select(σ⃗, r⃗, A);
    // Re-encryption with a new secret
    compute k ←$ Z_q;
    compute (σ⃗, r⃗) := OblivSel.Gen(k, pos);
    append Δ := OblivSel.Select(σ⃗, r⃗, A) to the stash, and update state s;
end
```

to the peer who will hold the stash. The tracker updates its state, in particular, update KeyMap with the new key, update the PosMap with the exact position of the block in the network (in the stash), and TagMap with the new uniformly sampled tag. We provide an algorithmic description of the Fetch process in Algorithm 4.

Sync Algorithm. The Sync in OBLIVP2P-1 consists of: (1) updating the state of the network, but also, (2) evicting the stash. The tracker determines the path to be evicted, $\mathsf{tag} = \nu \bmod 2^L$ and then fetches the position of all blocks in the stash and the path, $\mathcal{P}(\mathsf{tag})$. The tracker then generates, based on the least common ancestor algorithm (LCA), a permutation π that maps every block in $\mathsf{A} = \big(\mathsf{stash}, \mathcal{P}(\nu \bmod 2^L, 1), \cdots, \mathcal{P}(\nu \bmod 2^L, L)\big)$ to its new position in A', a new array that will replace the evicted path and the stash. The block $\mathsf{A}[\pi(i)]$ will be mapped *obliviously* to $\mathsf{A}'[i]$, for all $i \in [|\mathsf{stash}| + z \cdot L]$. The oblivious mapping between A and A' is performed by invoking OblivSel between the tracker, the peers in the path and m peers, $|\mathsf{stash}| + z \cdot L$ times. Note that (1) the blocks in A' are encrypted with a freshly-generated key, and (2) the mapping is not disclosed to any peers in the path as long as there is one non-colluding peer. Refer to Algorithm 5 for more detail about the Sync algorithm.

Upload Algorithm. A peer can request the tracker to add a file. For this, the

Algorithm 5: Sync(s): OBLIVP2P-1 sync operation

Input: tracker state s

// Fetch necessary parameters

$\nu \leftarrow \mathsf{s}$;

$\{\mathsf{adr}\} \leftarrow \mathsf{PosMap}^{-1}(\nu \bmod 2^L)$;

for adr **in** $\{\mathsf{adr}\}$ **do**

  set $\mathsf{T} = \mathsf{T} \cup \mathsf{tag} \leftarrow \mathsf{TagMap}(\mathsf{adr})$;

end

set $\mathsf{A} = \big(\mathsf{stash}, \mathcal{P}(\nu \bmod 2^L, 1), \cdots, \mathcal{P}(\nu \bmod 2^L, L)\big)$;

Initialize an array A', $\pi \leftarrow \mathsf{LCA}(\mathsf{T}, \nu)$;

// tracker generates key shares

for l **from** 1 **to** $z \cdot L + |\mathsf{stash}|$ **do**

  if $\exists \mathsf{adr}, l = \mathsf{PosMap}(\mathsf{adr})$ **then**

    set $\mathsf{k} \leftarrow \mathsf{KeyMap}(\mathsf{adr})$;

    set $\mathsf{k}'' = \mathsf{k}' - \mathsf{k}$, $\mathsf{k}' \xleftarrow{\$} Z_q$;

    compute $(\vec{\sigma}_l, \vec{r}_l) = \mathsf{OblivSel.Gen}(\mathsf{k}'', \pi(\mathsf{l}))$;

  else

    set $\mathsf{k}'' \xleftarrow{\$} Z_q$, compute $(\vec{\sigma}_l, \vec{r}_l) = \mathsf{OblivSel.Gen}(\mathsf{k}'', \pi(\mathsf{l}))$;

  end

end

// Peers generate the new array A'

for j **from** 1 **to** $z \cdot L + |\mathsf{stash}|$ **do**

  set $\mathsf{A}'[j] = \mathsf{OblivSel.Select}(\vec{\sigma}_j, \vec{r}_j, \mathsf{A})$;

end

for $j \in [m]$, send ${\mathsf{A}'}_j[1, \cdots, |\mathsf{stash}|]$ and ${\mathsf{A}'}_j[|\mathsf{stash}| + 1, \cdots, L]$ to peers in $\mathcal{P}(\nu)$ and the stash, and update state s;

tracker selects uniformly at random a set of m peers. The peer sends the file in a form of blocks. Every block is secret shared such that every peer in the m peers receives a share. The tracker generates a secret unique to the block, k. The tracker secret shares k to the m peers. The peers evaluate a seed-homomorphic PRG on the received shares and add it to the block share. Finally, the block is appended to a randomly selected peer in the network to hold a part of the stash.

4.4.4 Optimization: Handling Bursts

OBLIVP2P-1 has a functional limitation inherited by ORAMs. Any access cannot be started unless the previous one has concluded [2]. In our case, the tracker can handle fetching several blocks before starting the Sync operation. In our setting, we target increasing the P2P network throughput while leveraging the network storage and communication. In order to build a scalable system, we

[2] We do not consider a multi-processor architectures as those considered in OPRAM literature [87].

propose several optimizations.

O1: Replication. In Ring ORAM, $A = 3$ accesses can be performed before an eviction is required. To support A parallel accesses, we replicate every block A times in the tree. This absorbs the fetching access time and allows A simultaneous accesses, even for requests to the same resource. Additionally, we may replicate every block over A times on different peers, in case that the peer holding the block is offline due to churn, and cannot serve the block to the other peers. Lastly, the network operator can deploy multiple trackers to serve peers simultaneously, which leads to the throughput of OBLIVP2P-1 proportional to the number of trackers.

O2: Pipelining. While the eviction is highly parallelizable in OBLIVP2P-1, an eviction can take a considerable amount of time to terminate. If we denote by f the average number of fetch requests in the P2P network, and by t the time to perform an eviction, then the system can handle all the accesses if $t < \frac{1}{f}$. However, in practice $t > \frac{1}{f}$ and therefore the accesses will be queued and creates a bottleneck. To address this issue, we create multiple copies of the buckets that are run with different instances of OBLIVP2P-1 protocol which overlays on the same network. In the setup phase, every node creates l copies of its bucket space. Every bucket will be associated to different versions of OBLIVP2P-1 instantiations. For example, with replication we can handle A accesses in parallel on the (same) ith version of the buckets, but the upcoming accesses will be made on the $(i+1)th$ version. This will absorb the eviction time. To sum up, having different versions will increase the throughput of the system to $\frac{l}{f}$. In order to prevent pipeline stalls, we need to choose $l \geq t \cdot f$ in our implementation.

Another aspect (not considered for our implementation) for further optimizations in our versioning solution is to distribute the communication overhead of the peers in the network. In fact, the peers holding blocks at the higher level of the tree will be accessed more often compared to lower levels. In order

to distribute the communication load on the network peers, peers' location can be changed for different versions such that: the peer at the ith level of the tree in the jth version will be placed at the $(L - i + 1)th$ level of the tree in the $(j + 1)th$ version.

O3: Parallelizing Computation across m Peers. The scalar multiplication in the elliptic curve is expensive and can easily delay the fetch and sync time. For this, we consider every peer in the OblivSel as a set of peers. Whenever there is a need to perform scalar multiplication over a path, several peers participate in the computation and only the representative of the set will perform the aggregation. This optimization speeds up the OblivSel to be proportional to the number of peers' used to parallelize a single peer.

4.5 Implementation and Evaluation

Implementation. We implement a prototype of OBLIVP2P-0 and OBLIVP2P-1 in Python. The implementation contains 1712 lines of code (LOC) for OBLIVP2P-0 and 3226 for OBLIVP2P-1 accounting to a total of 4938 lines measured using CLOC tool [10]. Our prototype implementation is open source and available online [32]. As our building block primitives, we implement the Ring ORAM algorithm, IT-PIR construction and seed-homomorphic PRG. For Ring ORAM, we have followed the parameters reported by authors [206]. Each bucket contains $z = 4$ blocks and $s = 5$ dummy blocks. The eviction occurs after every 3 accesses. The blocks in OBLIVP2P-0 are encrypted using AES-CBC with 256 bit key from the pycrypto library [40]. For implementing IT-PIR and seed homomorphic PRG in OBLIVP2P-1, we use the ECC library available in Python [41]. We use the NIST P-256 elliptic curve as the underlying group.

Experimental Setup. We use the DeterLab network testbed for our experiments [16]. It consists of 15 servers running Ubuntu 14.04 with dual Intel(R) Xeon(R) hexa-core processors running at 2.2 Ghz with 15 MB cache (24 cores

each), Intel VT-x support and 24 GB of RAM. The tracker runs on a single server while each of the remaining servers runs approximately 2400 peers. Every peer process takes up to $4 - 60$ MB memory which limits the maximum network size to 2^{14} peers in our experimental set up. The tracker is connected to a 128 MBps link and the peers in each server share a bandwidth link of 128 MBps as well. We simulate the bandwidth link following the observed BitTorrent traffic rate distribution reported in [79]. In our experimental setting, multiple peers are simulated on a single machine hence our reported results here are conservative. In the real BitTorrent setting, every peer has its own separate CPU.

Evaluation Methodology. To evaluate the scalability and efficiency of our system, we perform measurements for a) the overall throughput of the system b) the latency for Fetch and Sync operations and c) the data transferred through the tracker for both OBLIVP2P-0 and OBLIVP2P-1. All our results are the average of 50 runs with 95% confidence intervals for each of them. Along with the experimental results, we plot the theoretical bounds computed based on Table 4.2. This helps us to check if our experiments match our theoretical expectations. In addition, we perform separate experiments to demonstrate the effect of our optimizations on the throughput of our OBLIVP2P-1 protocol. For our experiments in this section, we leverage the technical optimization introduced in Section 4.4.4.

We vary the number of peers in the system from 2^4 to 2^{14} peers (capacity of our testbed) and extrapolate them to 2^{21} peers. Note that, when increasing the number of peers, we implicitly increase the total data size in the entire network which is computed as the number of peers $\times$ the block size. That is, our P2P network handles a total data size that spans from 16 KB to 32 GB. For our evaluation, we consider each peer holds one ORAM bucket because of the limited available memory. In reality, every peer can hold more buckets. Note that, we linearly extrapolate our curves to show the expected results for larger number of peers starting from $2^{15} - 2^{21}$ (shown dotted in the Figures) , and therefore

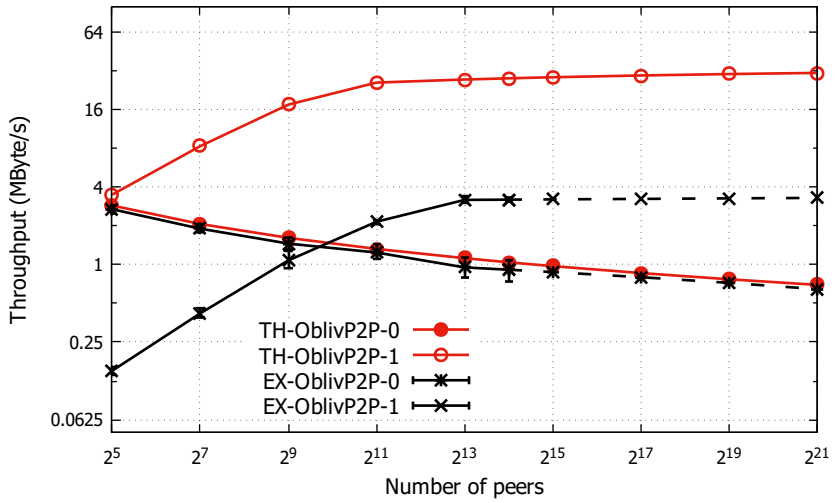


Figure 4.3: Theoretical (Th) and experimental (Ex) comparison of OBLIVP2P-0 and OBLIVP2P-1 parameters for block size of 512 KB. The throughput for OBLIVP2P-1 linearly scales with the increase in network size.

larger data size in the network. Aligned to the chunks in BitTorrent, we select our blocksize as 128 KB, 512 KB and 1 MB.

4.5.1 Linear Scalability with Peers

The throughput is an important parameter in designing a scalable P2P protocol. We define the throughput, as the number of bits that the system can serve per second.

From Figure 4.3, we observe that the throughput of OBLIVP2P-0 decreases with the increase in the total number of peers in the network. For a network size of 2^{14} peers, the experimental maximum throughput is 0.91 MBps. As we extrapolate to larger network size, the maximum throughput decreases, e.g., for 2^{21} peers, the throughput is 0.64 MBps. This shows that as the network size increases, the tracker starts queuing the requests that will eventually lead to a saturation. However, for OBLIVP2P-1, the maximum throughput for network size of 2^{14} is 3.19 MBps and is 3.29 MBps when extrapolated to 2^{21} peers. The throughput increases as there are more peers available in the network to distribute the computation costs. The throughput shows a similar behaviour for blocksize of 128 KB and 1 MB (as shown in Figure 4.7). Hence, we expect

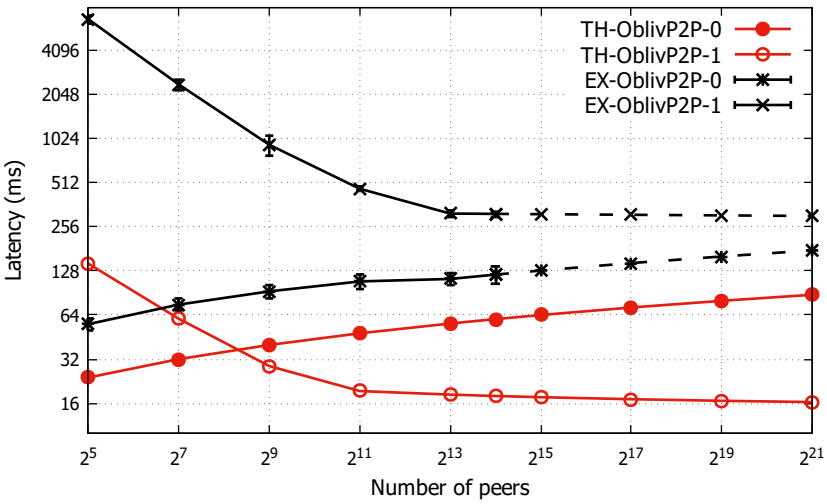


Figure 4.4: The latency for fetching a block for OBLIVP2P-1 reduces up to 2^{13} and then becomes constant.

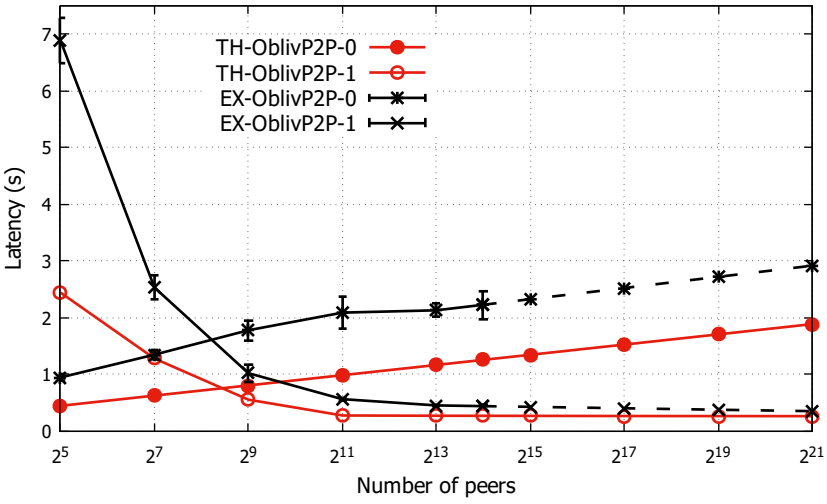


Figure 4.5: The latency for sync operation for OBLIVP2P-1 reduces up to 2^{13} and then becomes constant.

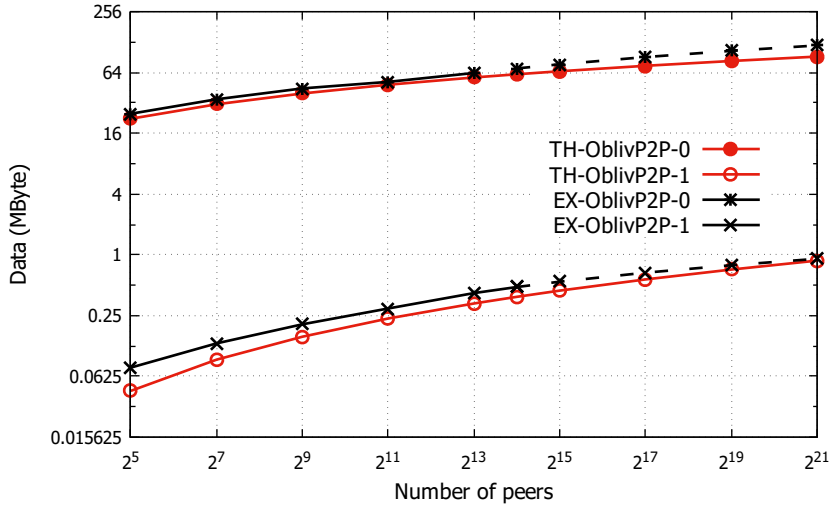


Figure 4.6: The data transferred through the tracker for OBLIVP2P-0 increases linearly with the number of peers

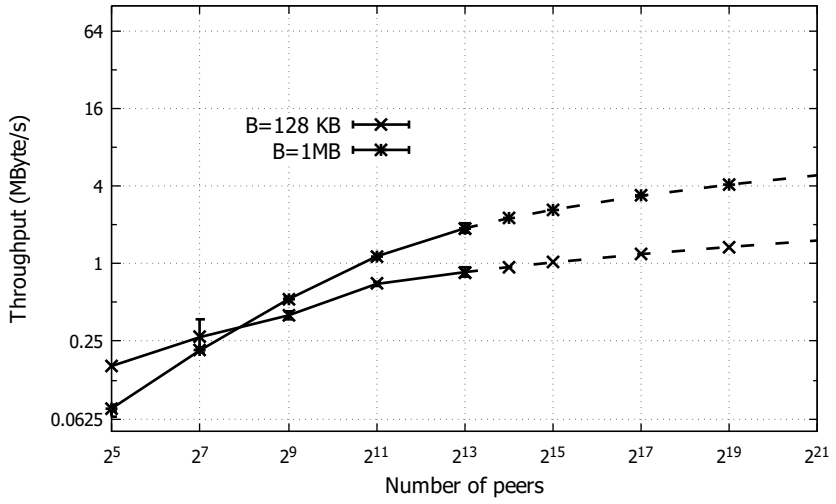


Figure 4.7: The throughput for blocksize 128 KB and 1 MB increases with increase in the network size.

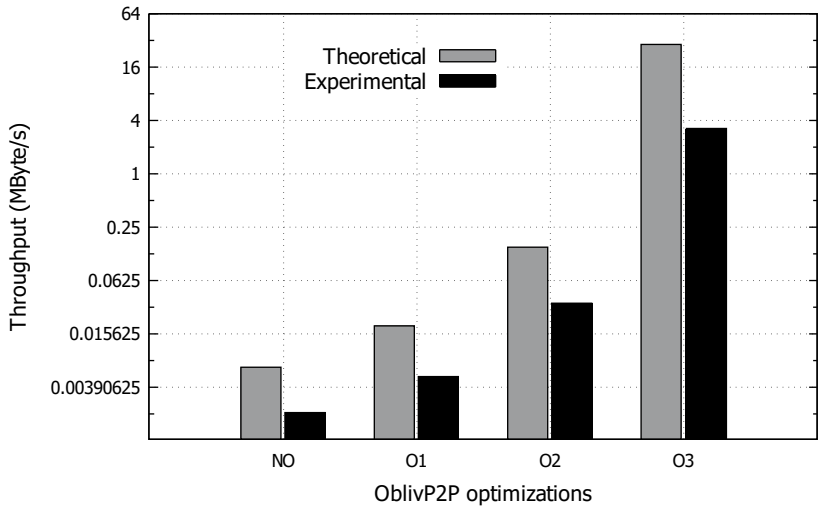


Figure 4.8: Impact of optimizations (O1-O3) on the throughput of OBLIVP2P-1 for 2^{14} peers and blocksize of 512 KB.

OBLIVP2P-1 to provide better throughput in a real setting where more computational and communication capacity for each peer can be provisioned. The throughput values for OBLIVP2P-1 are calculated after applying all the 3 optimizations discussed in Section 4.4.4. The behaviour of the theoretical throughput matches our experimental results. The theoretical throughput has higher values as it does not capture the network latency in our test environment.

Result 1. Our results show that the centralized protocol is limited in scalability and cannot serve a large network. Whereas, the throughput for OBLIVP2P-1 linearly scales ($0.15 - 3.39$ MBps) with increasing number of peers ($2^5 - 2^{21}$) in the network.

Result 2. For a block of size 512 KB and 2^{14} peers, OBLIVP2P-1 serves around 7 requests / second which can be enhanced with multiple copies of ORAM trees in the network.

Remark. The throughput may be acceptable to privacy-conscious users (e.g., whistleblowers), where privacy concerns outweigh download / upload latencies. As long as the number of request initiators is small, the perceived throughput remains competitive with a non-oblivious P2P system. Further, the network operator can deploy multiple trackers to serve peers simultaneously, which leads

to the throughput of OBLIVP2P-1 proportional to the number of trackers.

4.5.2 Latency Overhead and Breakdown

We define the latency as the time required to perform one ORAM operation in our P2P protocol. We measure the latency for the following operations:

Fetch. Figure 4.4 shows that the average time for fetching a block of 512 KB increases for OBLIVP2P-0 with increase in the size of the network. This is due to the increased computation and bandwidth overhead at the tracker. However, for OBLIVP2P-1, the latency initially reduces with the increasing number of peers (from 2^5 to 2^{11}) and then becomes constant after the network is large enough (around 2^{13}) to distribute the computation cost in the network[3]. OBLIVP2P-1 has a higher latency for fetch as compared to OBLIVP2P-0 due to the expensive computation required for performing scalar multiplication. The average time for fetching a block of size 512 KB is around 0.31 s for a network size of 2^{14} peers and remains steady with increase in the number of peers.

Sync. We measure the time for performing a sync operation for different network sizes. Figure 4.5 shows that the time for performing a sync operation increases in OBLIVP2P-0 with increase in the number of peers. Whereas for OBLIVP2P-1, the sync time reduces gradually at first and then becomes steady after the network size reaches 2^{13} peers which is as expected through our theoretical calculation. OBLIVP2P-1 uses the peers in the network to distribute the computation load and hence the sync time tends to be steady for large network sizes.

Data transferred through tracker. Figure 4.6 shows the amount of data that is transferred through the tracker per request. We perform this measurement to show that the centralized tracker becomes a bottleneck in OBLIVP2P-0. The

[3] Since a large number of nodes (e.g., over 1000 nodes) share one physical machine, its limited computation power drastically affects our result. Therefore, to be more realistic, we use the ideal computing time for each node solely in one physical machine as the computing time per node, and simulate our experiments for OBLIVP2P-1.

amount of data that the tracker has to process increases with increase in the number of peers. At 2^{21} peers, the amount of data is 118 MB (almost) reaching the bandwidth limit (128 MBps) of the tracker. Whereas, for OBLIVP2P-1 the amount of data transferred is around 1 MB for 2^{21} peers. This implies that the tracker could manage up to 128 copies of ORAM tree in parallel, which will increase the overall throughput by 128 times.

Result 3. OBLIVP2P has no centralized infrastructure as a bottleneck, ensuring that communication and computational overhead can be completely offloaded to the network.

4.5.3 Optimization Measurements

We perform incremental experiments to quantify the impact of each of the introduced optimizations on the overall throughput in Section 4.4.4, as shown in Figure 4.8. We fix the number of peers in the network to be equal to 2^{14} and the block size to 512 KB. We chose our optimization parameters based on our results in section 4.5.3. We fix the number of replicas to be equal to $A = 3$, i.e., the same data block is replicated three times. The burst parameter needs to be in $O(\frac{B}{\log q} \log N_p)$, where N_p is the number of peers, B the block size, and q the elliptic curve group order. Finally, we fix the number of parallel peers in the OblivSel.Select algorithm to be in $O(\frac{B}{\log q} \log N_p)$.

O1: Replication. Replication enables to perform $A = 3$ fetch operations in parallel. This implies that the throughput *theoretically* increases 3 times when compared to our baseline without any optimizations. Our experimental results show that we have 2.55 times improvement over the baseline, as expected theoretically.

O2: Pipelining. We evaluate the effect of our optimization (O2) that absorbs the eviction time by pipelining the fetch requests to different versions of the ORAM tree in the P2P network. We show that this optimization, when coupled to (O1), has theoretically increased the overall throughput by 23.05 times if compared

to the baseline. Our experiments are aligned to our theoretical results and show 17.2 times improvement over the baseline with a burst parameter of 17. Clearly, if the number of versions increases beyond 17, then OBLIVP2P-1 can handle parallel accesses, hence increasing the system throughput.

O3: Parallelizing m peers. We measure the effect of parallelizing the computation load of m peers by leveraging more peers in the network on the overall throughput of the system. We increase the number of peers to 116 peers that are used to compute the fetch and sync operations. Our theoretical result shows an improvement of 4398 times over the baseline, when coupled with (O1) and (O2). Our experiments support this result and demonstrates 1589 times improvement, the difference is due to the real network latency are not considered in our theoretical calculation.

Result 4. OBLIVP2P-1 is subject to several optimizations due to its highly parallelizable design.

4.6 OBLIVP2P-1 Analysis

In this section, we present the theoretical analysis on computation / communication overhead of tracker / peers and security analysis for OBLIVP2P-1.

4.6.1 Performance

We report OBLIVP2P-1 computation and communication overhead for the tracker and the network in Table 4.2. In particular, OBLIVP2P-1's tracker transmits a number of bits *independent* of the block size, the tracker does not perform any computation on the blocksize or store any block locally.

Tracker overhead. To fetch a block, the tracker invokes $\mathsf{OblivSel}$ twice. While for the eviction, the tracker performs $\mathsf{OblivSel}$ $\big(L \cdot z + |\mathsf{stash}|\big)$ times. That is, it is sufficient to first analyze $\mathsf{OblivSel}$ overhead and than just conclude for the overall tracker overhead.

Within one instance of $\mathsf{OblivSel}$, the tracker computes an $\mathsf{IT-PIR.Query}$ that outputs m vectors for m peers, each of size $L{\cdot}z{+}|\mathsf{stash}|$. Each $\mathsf{IT-PIR.Query}$ vector costs $\log q(L \cdot z + |\mathsf{stash}|)$ bits, where q is the group order. The tracker also needs to generate shares for the key, where the shares are in $\mathbb{Z}_q$. That is, one $\mathsf{OblivSel}$ costs the tracker $O\big(m \cdot \log q \cdot (L \cdot z + |\mathsf{stash}|)\big)$.

That is, the tracker has to transmit $O\big(m \cdot \log q \cdot (L \cdot z + |\mathsf{stash}|)^2\big)$ bits. Considering $L, |\mathsf{stash}| \in O(\log N)$, q the group order in $\mathsf{poly}(N)$, m the number of peers and z the bucket size as constants, then the tracker needs to send $O(\log^3 N)$ bits independently of the block size. That is, if $\mathsf{block} \in \Omega(\log^3 N)$, the tracker has a constant communication work per block. Moreover, the tracker is very lightweight as it does not perform any heavy computation like encryption / decryption of blocks, which permits the tracker to handle frequent accesses.

Peers overhead. Considering the communication between the peers, the main communication overhead comes from block transfer from the peers holding the path to the selected m peers. The m peers are selected uniformly at random. Each peer receives $(z \cdot L + |\mathsf{stash}|)$ blocks from the peers in the selected path and the stash. That is, in terms of communication overhead, the peers sends on average $\sum_{i=0}^{L} \frac{z}{2^i} + \frac{(z \cdot L + |\mathsf{stash}|)}{N} + \frac{z}{N}$ blocks per peers in the network. Considering z constant and $L, |\mathsf{stash}| \in O(\log N)$, implies that every peer is expected to transmit $O(\frac{\log N}{N})$ blocks per access.

In terms of computation, the main computational bottleneck consists of the scalar multiplication from the seed homomorphic PRG. For every $\mathsf{OblivSel}$, every peer needs to perform $(z \cdot L + |\mathsf{stash}|) \cdot \frac{B}{\log q}$ scalar multiplications per block. The second term, $\frac{B}{\log q}$, represents the number of points that a block contains. We also have $(z \cdot L + |\mathsf{stash}|)$ instances of $\mathsf{OblivSel}$ during the eviction. That is, the total number of scalar multiplication equals $O\big((z \cdot L + |\mathsf{stash}|)^2 \cdot \frac{B}{\log q}\big)$. Finally, the amortized computation over the total number of peers in the network equals $O(\frac{\log^4 N}{N})$ multiplications per eviction, considering $B \in \Omega(\log^3 N)$ and $q \in \mathsf{poly}(N)$.

4.6.2 Security Analysis

We show that OBLIVP2P-1 is an *oblivious* P2P as stated by Definition 4.2.2. For this, it is sufficient to show that an adversary cannot distinguish between a randomly generated string and the access pattern leaked by any peer's real access. This underlines the fact that the access pattern is independent of the address of the requested block. In our threat model, the adversary can have access to the content of buckets, monitors the communication between the peers, and has a total view of the internal state of dishonest peers. Buckets' content is assumed to be transmitted without any additional layer of encryption.

We present our *address-tag* experiment AT that captures our security definition. Let OBLIVP2P $= (\mathsf{Setup}, \mathsf{Upload}, \mathsf{Fetch}, \mathsf{Sync})$ represents an oblivious P2P protocol. Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an $\mathsf{IND\$ - CPA}$ encryption scheme. Let $\mathcal{G}$ be a secure pseudo-random generator. $\mathsf{AT}^{OblivP2P}_{\mathcal{A},\mathcal{E},\mathcal{G}}$ refers to the instantiation of the *address-tag* experiment by algorithms OBLIVP2P, $\mathcal{E}$, $\mathcal{G}$, and adversary $\mathcal{A}$. We denote by Col the event that m peers in the network collude and set $\Pr[\mathsf{Col}] = \delta_m$, by $\mathcal{B}_{\delta_m}$ the Bernoulli distribution, and λ the security parameter.

In the following, we fix the number of colluding peers $c \in O(N^{\epsilon})$, for $0 < \epsilon < 1$. We consider every peer in the network as a random variable distributed based on a Bernoulli distribution with probability equal to $\frac{c}{N} \in O(N^{\epsilon-1})$. Let us denote by $(X_1, \cdots, X_m)$ the random variables of the selected peers for every instantiation of $\mathsf{OblivSel}$. Note that $\Pr[\mathsf{Col}] = \Pr[X_1 = 1 \ \mathsf{AND} \cdots \ X_m = 1]$. Since all X_i's are independent, then, $\Pr[\mathsf{Col}] = \Pi_{i=1}^{m} \Pr[X_i = 1] = (\frac{c}{N})^m$. That is, $\delta_m = (\frac{c}{N})^m$ which implies under our assumptions that $\delta_m \in O(2^{\log N \cdot m \cdot (\epsilon-1)})$.

In the following experiment, we only consider the Fetch algorithm for obliviousness analysis. In our model, Upload sequences are indistinguishable by construction assuming that peers uploads blocks that are randomly distributed, and using random key for every block encryption. The experiment $\mathsf{AT}^{OblivP2P}_{\mathcal{A},\mathcal{E},\mathcal{G}}(\lambda, b)$ consists of:

- The adversary $\mathcal{A}$ picks one access operation $(\mathsf{Fetch}, \mathsf{adr}, \perp)$ and sends it

to the challenger $\mathcal{C}$

- If $b = 1$, pick $X \xleftarrow{\mathcal{B}_{\delta_m}} \{0,1\}$, if $X = 1$, then set $\mathsf{var} = \mathsf{adr}$, otherwise $\mathsf{var} = \perp$ and set

$$\begin{aligned}\pi_1 = \{&\big(\mathcal{P}(\mathsf{tag},1),\cdots,\mathcal{P}(\mathsf{tag},L)\big), \mathsf{tag} \leftarrow \mathsf{TagMap}[\mathsf{adr}],\\ &\big(\mathsf{Enc}(q_1),\cdots,\mathsf{Enc}(q_m)\big),\\ &(q_1,\cdots,q_m) \leftarrow \mathsf{IT}-\mathsf{PIR.Query}(\mathsf{pos}),\\ &\mathsf{pos} \leftarrow \mathsf{PosMap}[\mathsf{adr}], \mathsf{var}\}\end{aligned}$$

  If $b = 0$, set $\pi_0 = \{\big(P_1,\cdots,P_L\big) \xleftarrow{\$} \mathbb{G}^{z\times L},$
  $\big(q_1,\cdots,q_m\big) \xleftarrow{\$} \{0,1\}^{\lambda\times m}, \perp\}$

- Adversary $\mathcal{A}$ has access to an oracle $\mathcal{O}^{OblivP2P}$ that issues the access patterns for polynomial number of accesses (while paths are re-encrypted for every request)
- $\mathcal{A}$ outputs b'
- The output of the experiment is 1 if $b = b'$, otherwise 0. If $\mathsf{AT}^{OblivP2P}_{\mathcal{A},\mathcal{E},\mathcal{G}}(\lambda, b') = 1$, we say that $\mathcal{A}$ won the experiment.

The experiment differentiates between a realistic setting where the adversary can see the access pattern, and in which a possible colluding setting can happen with a pre-fixed probability, δ_m, and an ideal setting where the adversary receives a random string. We slightly re-formulate Definition 4.2.2 below.

Definition 4.6.1. *We say that a* P2P *is* oblivious *iff for all PPT adversaries $\mathcal{A}$, there exists a negligible function* negl *such that:*

$$\Pr[\mathsf{AT}^{OblivP2P}_{\mathcal{A},\mathcal{E},\mathcal{G}}(\lambda,1)=1] - \Pr[\mathsf{AT}^{OblivP2P}_{\mathcal{A},\mathcal{E},\mathcal{G}}(\lambda,0)=1] \leq \mathsf{negl}(\lambda)$$

Theorem 4.6.1. *If $\forall N > 1$, and $\forall \epsilon < 1$, $\exists m > 1$ s.t. $2^{\log N\cdot m\cdot(1-\epsilon)} \in \mathsf{negl}(\lambda)$, $\mathcal{G}$ is a secure pseudo-random generator, $\mathcal{E}$ is $\mathsf{IND\$-CPA}$ secure, then* OBLIVP2P-*1 is an oblivious P2P as in Definition 4.6.1.*

Proof. To prove our theorem, we proceed with a succession of games as follows:

- Game_0 is exactly the same as $\mathsf{AT}^{OblivP2P}_{\mathcal{A},\mathcal{E},\mathcal{G}}(\lambda, 1)$
- Game_1 is the same as Game_0 except that the blocks in the buckets $\mathcal{P}(\mathsf{tag}, i)$ are replaced with random points from $\mathbb{G}$
- Game_2 is the same as Game_1 except that the the encrypted $\mathsf{IT-PIR}$ queries are replaced with random strings

From games' description, we have

$$\Pr[\mathsf{Game}_0 = 1] = \Pr[\mathsf{AT}^{OblivP2P}_{\mathcal{A},\mathcal{E},\mathcal{G}}(\lambda, 1) = 1], \tag{4.1}$$

For Game_1, we can build a distinguisher B_1 that reduces security of $\mathcal{G}$ to PRG security such that:

$$\Pr[\mathsf{Game}_0 = 1] - \Pr[\mathsf{Game}_1 = 1] \leq \mathsf{Adv}^{PRG}_{B_1,\mathcal{G}}(\lambda), \tag{4.2}$$

Similarly for Game_1, we can build a distinguisher B_2 that reduces $\mathcal{E}$ to $\mathsf{IND\$-CPA}$ security such that:

$$\Pr[\mathsf{Game}_1 = 1] - \Pr[\mathsf{Game}_2 = 1] \leq \mathsf{Adv}^{\mathsf{IND\$-CPA}}_{B_2,\mathcal{E}}(\lambda), \tag{4.3}$$

We need now to compute $\Pr[\mathsf{Game}_2]$.

$$\begin{aligned}\Pr[\mathsf{Game}_2] =& \Pr[\mathsf{Col}] \cdot \Pr[\mathsf{Game}_2 = 1 |\ \mathsf{Col}] + \\ & \Pr[\overline{\mathsf{Col}}] \cdot Pr[\mathsf{Game}_2 = 1 |\ \overline{\mathsf{Col}}] \\ =& \delta_m + (1 - \delta_m)\frac{1}{N}\end{aligned}$$

On the other side $\Pr[\mathsf{AT}^{OblivP2P}_{\mathcal{A},\mathcal{E},\mathcal{G}}(\lambda, 0) = 1] = \frac{1}{N}$, since the tag is generated uniformly at random for every access.

$$\Pr[\mathsf{Game}_2] - \Pr[\mathsf{AT}^{OblivP2P}_{\mathcal{A},\mathcal{E},\mathcal{G}}(\lambda, 0) = 1] = \delta_m(1 - \frac{1}{N}) \tag{4.4}$$

From equations 4.6.2, 4.6.2, 4.6.2, and 4.6.2 we obtain:

$$\Pr[\mathsf{AT}^{OblivP2P}_{\mathcal{A},\mathcal{E},\mathcal{G}}(\lambda,1)] - \Pr[\mathsf{AT}^{OblivP2P}_{\mathcal{A},\mathcal{E},\mathcal{G}}(\lambda,0)=1] \leq$$
$$\delta_m(1-\frac{1}{N}) + \mathsf{Adv}^{\mathsf{IND\$-CPA}}_{B_2,\mathcal{E}} + \mathsf{Adv}^{PRG}_{B_1,\mathcal{G}}.$$

Since $\delta_m \in O(2^{\log N \cdot m \cdot (\epsilon-1)})$, this ends our proof.

□

Quantitatively, if the number of peers in the network equals 2^{20}, number of colluding peers in the network is $c = N^{\frac{1}{2}}$ and $m = 12$, then $\delta_{12} = 2^{-120}$. Given the number of colluding peers and total number of peer, the value of m can always be adjusted to handle the desired colluding probability δ_m. In case of churn, the fraction c can vary and therefore the length of the circuit m has to be adapted to the new value. Furthermore, we implicitly assumed so far that no peer among the m selected leaves in the middle of the OblivSel process. If that occurs, the entire process has to abort, re-calculates the number of required peers m, and perform the OblivSel from scratch.

4.7 Discussion

Existing approaches. A valid question to investigate is whether existing solutions such as unlinkability or path non-correlation techniques can be extended to handle global adversaries and therefore prevent traffic analysis at the cost of providing more resources. It is easy to see that unlinkability techniques (e.g., mixnet) can provide better security in a P2P network under some assumptions. As an instance, assuming the case where a large number of peers behave as senders and issue requests that will be mixed by *sufficient* network peers before being answered by corresponding receivers' peers. Also, assuming that there is at least one honest peer in the mixing network, this solution would provide slightly the same level of security as OBLIVP2P where a global adversary can-

not distinguish the senders' peers access pattern. However, this solution suffers from two downsides. First, there is a need to have *sufficient* number of senders' peers *on-line* in order to prevent intersection attacks. That is, in order to prevent traffic analysis, the number of senders represents a security parameter of the system that has to be maintained throughout the entire run of the system. Second, as the receivers' contents are theirs and are not encrypted, plus, all peers are considered honest-but-curious, a global adversary can easily find out what content is being accessed independently of the sender identity. This therefore does not achieve obliviousness as defined in our work but only a weaker version of it. On the other hand, path non-correlation techniques conceptually cannot prevent against global adversary as we have detailed in Section 4.2. To sum up, it is not clear if existing techniques, even if given enough resources, can provide similar security insurances as those in OBLIVP2P.

Does better network & computation help? As empirically demonstrated in our evaluation section, the throughput of OBLIVP2P is around 3.19 MBps while considering only *one* tracker in the network. In a plaintext version of P2P system such as BitTorrent, the network leverages multiple trackers in order to handle more queries, and therefore increase the overall throughput. In OBLIVP2P, if we consider multiple copies of the entire network, we can also handle multiple trackers, and the throughput is expected to increase linearly with the number of trackers. However, as we delegate computation to the peers in OBLIVP2P, increasing the number of trackers beyond a particular threshold might turn out to be useless as the computation would represent a bottleneck of the system. As future work, we plan to investigate the asymptotic and empirical implications of including multiple trackers in the system. Moreover, it would be interesting to find out the relation between the number of trackers, number of peers for an ideal throughput of OBLIVP2P.

4.8 Related Work

Previous work has shown possible attacks by leveraging side channels such as packet sizes, number of packets and timing. These side channels leak users' private information, e.g., illnesses/medications/surgeries, income and investment secrets [102]. An attacker can employ machine learning techniques (e.g., Support Vector Machines) on network traffic to identify the user's browsing websites [127, 191, 224, 225]. However, our focus in this paper is to only prevent long-term pattern traffic analysis. The aforementioned side-channels of traffic analysis are out of scope. Although high-latency anonymous systems (e.g., mixed network) provide unlinkability between users and messages, these systems are prone to intersection or disclosure attacks if an adversary can observe multiple request rounds in the network [63,114,116,119,159,173,175,197,219].

For P2P content sharing systems, e.g., BitTorrent, studies have shown that monitoring BitTorrent traffic reveals users' private information, e.g., the data requested and sent by them [164, 200, 212]. Researchers also show that BitTorrent users on top of Tor still leak information related to the resources uploaded or downloaded [8, 80, 174] to a long-term traffic analysis attacker. With all the facts, long-term traffic analysis is still a major security concern and challenging problem for P2P content sharing systems. In this paper, we propose a new approach to hiding data access patterns, making P2P systems oblivious, and further to protecting against persistent, global traffic analysis in P2P content-sharing systems.

Multi-servers and parallel ORAM. There have been works on how to optimize ORAM constructions while leveraging multiple servers [171, 214, 215], multiple CPUs [87, 101], computational servers [121, 182], or distributed under a weaker threat model [112] . However, none of these recent constructions fit to a P2P setting as is. This is *mainly* due to the inherent client / server setting that results on a single entity bottleneck. The client has to either perform non-trivial computation or/and transmit several amount of bits.

OblivStore [215], Lu and Ostrovsky [171], and Stefanov and Shi [214] demonstrate how to decrease the communication overhead while leveraging multiple ORAM nodes and servers. However, all these constructions are centralized and route the block through the tracker. This leads to a single entity bottleneck.

Recently, researchers have proposed oblivious parallel RAM (OPRAM) [87, 101]. This was motivated by current multi-cpu architectures that can access the same or multiple resources in parallel. However, OPRAM does not decrease the communication overhead making it as well a single-entity bottleneck. Dachman-Soled et al. introduced oblivious network RAM (ONRAM) [112]. ONRAM can reduce the communication overhead between the client and multiple banks of memory to be constant in the number of blocks. However, it assumes a weak threat model, and cannot achieve obliviousness in the case of a global adversary.

4.9 Summary

We advocate hiding data access patterns as a necessary step in defenses against long-term traffic pattern analysis in P2P content sharing systems. To this end, we propose OBLIVP2P— an oblivious peer-to-peer protocol. Our evaluation demonstrates that OBLIVP2P is parallelizable and linearly scalable with increase in number of peers, without bottleneck on a single entity.

Chapter 5

Robust Synchronous P2P Primitives Using SGX Enclaves

5.1 Introduction

The robustness of P2P protocols is the basis of the core utilities and security / privacy properties provided by these protocols. In APAC and OBLIVP2P, we consider that an adversary passively monitors the traffic in the network and follows the given protocol. However, in reality, malicious (byzantine) nodes can easily join these P2P systems, and the presence of such adversaries is a major security concern in P2P systems, as they can disrupt the protocol execution or robustness. In this chapter, we propose robust P2P primitives against byzantine adversaries in P2P systems. Peer-to-peer systems such as BitTorrent [7], Symform [46], CrashPlan [12], StorJ [44], Tor [47] and Bitcoin [6] are becoming popular among users due to ease of accessibility. In such P2P systems, online users can simply volunteer as peers (nodes) to join the network. However, this exact property allows adversarial or Sybil peers to be a part of the network and exhibit a *byzantine* behavior. Such byzantine adversaries introduce lots of serious security issues to P2P systems. For example, researchers have demonstrated that in Bitcoin byzantine nodes can collude to eclipse or partition the honest

nodes leading to double-spending and selfish mining attacks [146, 189]. Further, byzantine nodes in anonymous P2P systems can selectively deny service to weaken the anonymity guarantee of such systems as Tor [47, 84]. In addition, byzantine nodes in the network can selectively forge, divert, delay or drop messages to disrupt the protocol execution. Therefore, designing robust P2P protocols in a byzantine setting continues to be an important research problem.

Researchers have extensively worked in the byzantine model to design solutions for fundamental P2P problems such as reliable broadcast and agreement among the peers [61, 65, 76, 78, 138, 139, 162, 195]. There are well-known impossibility results in the standard model of byzantine setting, such as the inability to achieve reliable broadcast or agreement when over $\frac{1}{3}$ of the network is byzantine [162, 195]. In a quest for efficient protocols that tolerate a larger fraction of malicious nodes, several failure models have been proposed which limit the capabilities of the byzantine adversaries. For instance, one such model is the *general-omission* model where the byzantine node can only omit messages that are either sent or received by it during the execution of a protocol [193, 198]. In this weaker adversarial model, it is possible to tolerate $\frac{N}{2}$ adversarial nodes and design relatively simple and efficient protocols for reliable broadcast [99, 145, 193, 198]. However, many of these adversary models make strong assumptions, which are not always realistic and have not had a concrete basis for implementation.

Our approach. To this end, we study the possibility of using recent hardware-root-of-trust mechanisms for making previous adversarial models realizable in practical systems. We observe that emerging hardware, specifically Intel SGX, provides stronger trusted computing capabilities, which allow running hardware-attested user-level enclaves on commodity OSes [25, 26, 110]. Enclaves provide hardware-isolated execution environment which guarantees that an application executing in an enclave is tamper-resistant and can be attested remotely. Assuming that SGX-like capabilities become commodity and widescale in end hosts,

we ask if it is feasible to build robust P2P protocols. Our main observation is that by leveraging the capabilities of such a trusted hardware, one can restrict the behavior of byzantine adversaries to the *general-omission* model in synchronous networks [99, 145, 193, 198].

Specifically, we use four SGX features, i.e., enclave execution (F1), unbiased randomness (F2), remote attestation (F3) and trusted elapsed time (F4)[1]. Based on these hardware features, we enforce six security properties (P1 - P6). First, we enforce execution integrity (P1), message integrity & authenticity (P2) and blind-box computation (P3), thus the attacker cannot forge messages or deviate from the execution of the given protocol. Thus, the adversarial node can only delay, replay and omit messages. We further leverage lockstep execution (P5) and message freshness (P6) to reduce the adversarial model to the general-omission model, where byzantine nodes have no additional advantage than omitting to send / receive messages[2]. In such model, P3 disallows the adversary to selectively omit messages based on the content. Lastly, the halt-on-divergence (P4) allows us to detect and eliminate peers that selectively omit messages based on identities of senders / receivers, thus in turn reducing round complexity and "sanitizing" the network. Leveraging these properties we can further achieve improvement for the efficiency of protocols. We present efficient designs for reliably broadcasting messages called *Enclaved Reliable Broadcast (*ERB*)* protocol and an unbiased common random generator called *Enclaved Random Number Generator (*ERNG*)* protocol. Both ERB and ERNG primitives can be used as building blocks to solve a wide range of problems in distributed systems, such as random beacons [202], voting schemes [184], random walks [144], shared key generation [140, 141], cryptocurrency protocols [172] and load balancing protocols [111, 203] (details in Section 5.10.2).

Results. Our work targets synchronous network where every machine is run-

[1]These are available in Intel SGX and are commonly provided by other trusted hardware mechanisms [3, 55].

[2]A node's crash failure is equivalent to omitting all messages for the rest of the execution for the protocol.

ning an SGX-enabled CPU. As shown in Table 5.1 and Table 5.2, both of our protocols asymptotically reduce the round and communication complexity as compared to previous works in the byzantine model, and match with (or outperform) the results in general-omission model. For a network of size N, the round and communication complexity for ERB are $\min\{f+2, t+2\}$ and $O(N^2)$, where t and f ($f \leq t$) are the number of byzantine peers and peers actually behaving maliciously for one execution of ERB, respectively. The round and communication complexity for basic ERNG are $O(N)$ and $O(N^3)$, and for the optimized ERNG are $O(\log N)$ and $O(N \log N)$, respectively. We implement a prototype of our solution and the source code is available online [34]. We evaluate our implementation for both ERB and ERNG and our experimental results match our theoretical claims.

Contributions. The main contributions of this chapter are:

- *Realizable General-Omission Model* - We leverage SGX features to reduce the byzantine model to the general-omission model, where byzantine nodes have no extra advantage than omitting messages.
- *Better Synchronous P2P Protocols* - By enforcing our properties, we can improve the efficiency of P2P protocols. As the first attempt, we propose efficient protocols for reliable broadcast (ERB) and unbiased random number generation (ERNG) in synchronous settings.
- *Security Analysis & Evaluation* - We provide security analysis and proof for our protocol constructions. Our experimental evaluation confirms the theoretical expectations of our solutions.

[3] Some of these protocols are designed for byzantine agreement, but it is proved that they can be easily transformed to achieving reliable broadcast with only introducing additional message complexity of $O(N)$ [220].

[4] They assume that every byzantine node only sends a bounded number of messages per round, and honest nodes can use digital signatures to sign each message.

Protocol[3]	Attacker Model	Network Size	Round Complexity	Comm. Complexity
PT [198]	Omission	$t+1$	$\min\{f+2, t+1\}$	$O(N^3)$
PR [193]		$2t+1$	$\min\{f+2, t+1\}$	
CT [99]			$2t+1$	$O(N^2)$
PSL [195]	Byzantine	$3t+1$	$t+1$	O(exp (N))
BGP [78]			$\min\{f+2, t+1\}$	
BG [76]		$4t+1$	$t+1$	O(poly(N))
GM [138, 139]		$3t+1$	$\min\{f+5, t+1\}$	
AD15 [61]			$\min\{f+2, t+1\}$	
AD14 [65]	Byzantine[4]	$2t+1$	$3t+4$	$O(N^4)$
ERB	Byz. + SGX	$2t+1$	$\min\{f+2, t+2\}$	$O(N^2)$

Table 5.1: Round complexity and communication complexity for reliable broadcast in synchronous network.

Protocol	Network Size	Round Complexity	Comm. Complexity
AS [67]	$6t+1$	$O(N)$	$O(N^3)$
AD14 [65]	$2t+1$	$O(N)$	$O(N^4)$
Basic ERNG	$2t+1$	$O(N)$	$O(N^3)$
Optimized ERNG	$3t+1$	$O(\log N)$	$O(N \log N)$

Table 5.2: Round / communication complexity for random number generation protocols in synchronous distributed systems.

5.2 Problem

Designing efficient solutions for P2P protocols in the byzantine setting is a widely-recognized problem with limited solutions [61,65,76,138,139,195]. Our goal is to shed light on how SGX can aid to improve efficiency of synchronous P2P protocols. In this work, we take two fundamental problems as examples: 1) reliable broadcast and 2) common unbiased random number generator.

5.2.1 Problem Definition

In light of the previous works, we recall the standard definition of *reliable broadcast* [99, 193] and *common unbiased random number* [67] in the synchronous network:

Definition 5.2.1. (Reliable Broadcast). *A protocol for reliable broadcast in synchronous settings satisfies the following conditions:*

- *(Validity) If the sender is honest and broadcasts a message m, then all honest nodes eventually accept m.*
- *(Agreement) If an honest node accepts m, then all honest nodes eventually accept m.*
- *(Integrity) For any message m, every honest node accepts m at most once, if m was previously broadcast by the sender.*
- *(Termination) Every honest node eventually accepts a message (m or $\perp$).*

In order to define a *common unbiased random number* generator, we define the *bias* of any multi-variate function in a standard way [67].

Definition 5.2.2. (Unbiasedness). *Let $G : \{0,1\}^{k \times N} \rightarrow \{0,1\}^k$ be a deterministic multi-variate function that maps N elements in $\{0,1\}^k$ to one element in $\{0,1\}^k$. We define the* bias *of G, $\beta(G)$, as follows:*

$$\beta(G) = \max_{S \subseteq \{0,1\}^k} \left(\max \left(\frac{E[S]}{E_G[S]}, \frac{E_G[S]}{E[S]} \right) \right),$$

where $E_G[S]$ is the expected number of values in $G(x_1, \cdots, x_N) \in S$, and $E[S] = \frac{|S|}{2^k}$, which is the expected value when the output of G is distributed uniformly at random.

Definition 5.2.3. (Common Unbiased Random Number). *A protocol G generates a common unbiased random number r among N nodes if it satisfies the following conditions with high probability (w.h.p.)* [5]*:*

- *(Agreement) At the end of the protocol, all the honest nodes agree on the same value r.*
- *(Unbiasedness) The bias of $\beta(G) = 1$.*

For the analysis of protocols, we define the following complexities with respect to a single execution of the protocol.

- The *message / communication complexity* is defined as the total number of messages / bits transferred among all nodes in the worst case.
- The *round complexity* is defined as the number of executed rounds (or steps) in the worst-case.

5.2.2 Attacker Model

We consider a widely-studied standard synchronous model of P2P systems [61, 65, 76, 138, 139, 195]. In this model, our only new requirement is that every peer[6] in the network uses an SGX-enabled CPU to run the P2P protocols. In a network of N nodes, the number of byzantine nodes t is strictly bounded under a fraction of $\frac{N}{2}$. The number of peers that actually behave maliciously for a particular execution of the protocol is $f(\leq t)$. Thus, a P2P network $\mathcal{P}$ is composed of N peers $\mathcal{P} = \{p_1, \cdots, p_N\}$ such that $N = 2t + 1$. Every peer p_i in the P2P overlay has an identifier id_i and can communicate with other peers

[5] For the rest of the chapter, unless otherwise stated, if some probability p is negligible, it means that the event occurs with a probability of at most $O(e^{-\lambda})$ for some security parameter λ. Analogously, if some event occurs with high probability (w.h.p), the probability is at least $1 - O(e^{-\lambda})$.

[6] We use the word peer and node interchangeably in this work.

using their ids. The underlying TCP/IP substrate is assumed to provide reliable message delivery within a known bounded delay say Δ. Moreover, we consider a round-based *synchronous* model where each round is equal to the time an honest node requires to send a message and receive a response. Every peer is directly connected to all other peers in the network and knows the network size N. To summarize, we assume: the network size is N (S1); the protocol starts synchronously (S2); the round time is 2Δ (S3); the number of byzantine nodes is limited upto $\frac{N}{2}$ (S4); the peers are connected to each other (S5). This is a prominently used model in the previous literature of distributed P2P systems [61, 65, 67, 144, 193, 198]. We discuss the validity of these assumptions in Section 5.10.1.

Our Model using SGX. In our model, a byzantine peer has a compromised or malware-ridden operating system but executes protocols using SGX enclaves [25, 26, 110]. Enclaves guarantee untampered execution in presence of malicious underlying software or co-processes. The byzantine nodes can take arbitrary software actions as long as it does not violate SGX guarantees.

Scope. Our focus is showing how to leverage SGX features to improve the efficient of synchronous P2P protocols. Our model does not consider an adversary that can perform hardware attacks and break SGX security guarantees. We do not aim to prevent any information leakage through side-channels such as pagefaults, memory accesses or timing attacks to which SGX-enabled CPUs are known to be susceptible [165, 177, 227]. Indeed these problems are under investigation and recent research shows that defending against them is feasible. Existing solutions against these problems can directly apply to our work [190,211].

5.2.3 Strawman Solution & Attacks

Consider a strawman protocol for distributed random number generation using reliable broadcast, where the initiator broadcasts a random number m using an initialization message INIT to all the peers in a synchronous network (shown in

Algorithm 6). If m is generated randomly and unbiasedly as well as reaches every honest node without being tampered, then all honest nodes will agree on the common unbiased random number m and the goal of the protocol is achieved. In Algorithm 6, upon receiving the INIT message, each peer further multicasts an ECHO message to all other peers. After receiving the ECHO messages from the majority of nodes, each peer accepts m as the final message $\hat{m}$. Note that if the initiator is honest, all honest nodes receive the message INIT during the first round and multicast ECHO messages at the beginning of the second round. In the second round, every honest node receives at least $N - t$ ECHO messages from $N - t$ honest nodes and maybe some byzantine nodes. Thus, after two rounds, every honest node will output the same value m from the initiator, which satisfies all the conditions of reliable broadcast in Definition 5.2.1. However, we show how a byzantine initiator and other byzantine peers can attack this protocol to violate Definitions 5.2.1 and 5.2.3.

Attacks by Byzantine Adversary. Byzantine initiator and peers can tamper with the execution of protocol in Algorithm 6 and forge the values of INIT and ECHO messages to perpetrate the following attacks.

A1 (Execution Deviation): For this attack, an adversary deviates from the control flow of the running program for the given protocol. The adversary can disregard essential conditions to jump to the desired instructions and execute them directly. For example, the adversary can skip all the conditions like Line 7 & 13 to directly multicast its ECHO value to parts of honest nodes but not all of them, to introduce equivocation to their final decisions. Moreover, the adversary can also repeat particular instructions to obtain an output she wants. For instance, if m is generated from a random source without being tampered during the execution of the protocol, an unbiased common random number can be agreed among all the peers in the network. A byzantine peer, however, can repeat the step that generates m (Line 3) from the random source until it returns a favorable random number. Hence, the output is biased as per Definition 5.2.3.

Algorithm 6: Strawman distributed random number generation protocol using reliable broadcast.

Input: A P2P network $\mathcal{P}$ composed of N nodes, an initiator node $\mathsf{id}_{\mathsf{init}}$
Output: A message $\hat{m}$

```
1  Initialization: m̂ ←⊥; SS_m ← ∅; rnd ← 1
2  upon self_id is initiator:
3  get(m) // m is a random number
4  m̂ ← m
5  add self_id to SS_m
6  multicast INIT(m) to other peers
7  for rnd ≤ t + 1 do
8      upon receiving INIT(m):
9      m̂ ← m
10     add self_id and sender_id to SS_m
11     multicast ECHO(m) to other peers in round rnd + 1
12     upon receiving ECHO(m):
13     if m̂ =⊥ then
14         m̂ ← m
15         add self_id to SS_m
16         multicast ECHO(m) to other peers in round rnd + 1
17     end
18     if m = m̂ and sender_id ∉ SS_m then
19         add sender_id to SS_m
20         if |SS_m| = N − t then
21             accept m̂
22         end
23     end
24     rnd ← rnd + 1
25 end
26 if rnd > t + 1 then
27     accept ⊥
28 end
```

A2 (Message Forgery): Suppose that the adversary does not deviate from the execution of the given protocol, she can still alter the data flow (including input / output and intermediate states) of the program to forge messages. As per Definition 5.2.1, a reliable broadcast protocol requires that if one honest node accepts message m then all honest nodes accept m. The adversary can tamper with the INIT and ECHO messages to violate this agreement property of the protocol. A byzantine initiator colluding with other byzantine peers in the network can tamper with Line 6, 11 and 16 in the algorithm such that some honest nodes receive most ECHO messages with m' while others with m. This results in a fraction of honest nodes assigning $\hat{m}$ with m' and accepting m', while other honest nodes accept m as the final output of the protocol, thereby causing inconsistency in

the network.

A3 (Selective Omission): Assume that the adversary does not deviate from the control flow (i.e., the execution) of the given protocol or tamper with the data flow to forge messages, she can still omit, delay and replay messages in this restricted model. For an omission attack, it has two types: one is based on the content of the transmitted message and the other is dependent on the identity of the sender / receiver. For the first type, the adversary can observe its generated or received random number m and selectively decide to drop or forward it to other nodes based on its value, which introduces a bias in the final output for the honest nodes. For example, if the adversarial peers receive or initiate a message m, which is not the favorable one, they can omit to relay the message to the other nodes, thus all honest nodes may finally agree on $\perp$ instead of m. Further, to violate the agreement condition in Definition 5.2.1 and 5.2.3, the adversary can selectively decide to omit the message m depending on whether the destination peer is honest or malicious. It can broadcast m correctly to a few honest nodes and not send the message to the others for the last round. The honest nodes receiving m can multicast m to the others, but the others will not accept it as the execution ends. Thus, the honest nodes that do not receive a message will agree on $\perp$ while others will agree on m.

A4 (Message Delay): Alternatively, to generate an unbiased common random number, every peer can broadcast its random number to all other peers using Algorithm 6. All peers can then XOR the random numbers in the final set to generate the output. To bias this final output, a byzantine peer can intentionally hold its random number until it receives inputs from all other honest peers [67]. In this way, the adversary can "look ahead" in the protocol, calculate the final output and then decide whether to participate in the protocol by sending its random number. If the final random number already favors the adversary then it does not participate in the protocol, otherwise it sends its message to all the peers. Note that, for $t < \frac{N}{2}$, all the byzantine adversaries can collude to intro-

duce an exponential bias in the final value.

A5 (Message Replay): In the restricted model, the adversarial node can use a message m_{prev} from an instance of the protocol running in parallel, or which was run in the past to one (or more) honest node(s) and forward the correct message m to other honest nodes [168]. This results in an inconsistency where few honest nodes agree on m_{prev} and others agree on m, thereby violating the agreement condition of the protocol.

5.3 Solution Overview

In this work, our ultimate goals, with the help of SGX features, are twofold: 1) reducing the byzantine model to the general-omission model; 2) achieving improvement for the efficiency of synchronous P2P protocols (e.g., lower communication / round complexity). In this section, we put forward ideas using SGX features to enforce six security properties to restrict the capabilities (A1 - A5) of a byzantine adversary, as shown in Section 5.2.

5.3.1 SGX Features and Security Properties

We first start by recalling Intel SGX features which can also be provided by other trusted hardware.

F1: *Enclaved Execution* - SGX supports hardware-isolated memory region called enclaves such that a compromised underlying OS cannot tamper the execution of the code running inside this enclave.

F2: *Unbiased Randomness* - SGX provides a function `sgx_read_rand` that executes the `RDRAND` instruction to generate hardware-assisted unbiased random numbers.

F3: *Remote Attestation* - SGX allows a remote party to verify that an application is running in an enclave on an SGX-enabled CPU.

F4: *Trusted Elapsed Time* - SGX provides a function `sgx_get_trusted_time`

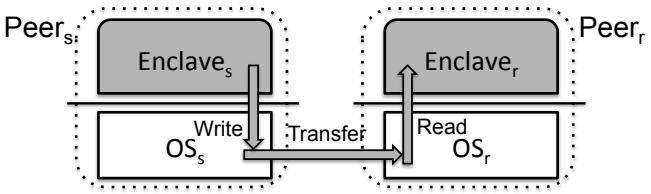


Figure 5.1: Each peer consists of two entities: an Enclave and an OS. The OS models the operating system and memory. The Enclave models the isolated memory and the secure execution of a program. The sender Enclave$_s$ can send a message via a secure channel to the receiver Enclave$_r$. The grey areas are secure against malicious OSes of byzantine nodes.

that returns a trusted elapsed time in seconds relative to a reference point.

Abstractly, a peer can be considered as the composition of two entities: an OS and an Enclave as shown in Figure 5.1. The OS models the untrusted entity including the operating system and memory. It has access to all the system resources such as file system and network. The OS can arbitrarily invoke an enclave program and start its execution. The Enclave models the isolated memory space that loads the program and executes it securely. Thus, Enclave corresponds to the trusted entity of a peer. We illustrate how to enforce P1 - P6 properties using SGX features to thwart A1 - A5 attacks.

P1 (Execution Integrity): With remote attestation (F3), an enclave in one peer can verify the correctness of the running program for the given protocol on the other nodes and whether it is executing on a valid SGX-enabled CPU or not. Moreover, F1 ensures that the execution in an enclave cannot be tampered with by the OS. F1 and F3 together enforce the execution integrity against A1. Hence, an adversary cannot deviate from the execution of the protocol in an enclave arbitrarily by skipping / repeating instructions to violate the control flow of the running program.

P2 (Message Integrity & Authenticity): In designing our protocols, we first perform a setup phase where each peer connects to every other in the network and then performs a series of steps. Analogous to P1, every enclave first uses F3 to verify the correctness of the protocol executing on other peers. Next, they generate public / private key pairs inside the enclaves and exchange the

public keys with each other. Then all the messages transmitted between any two enclaves can be signed to ensure the integrity and authenticity against A2. Moreover, the internal states of the program are also protected using F1. Therefore, the integrity of all messages including input / output / intermediate states is guaranteed. In this case, it is clear that an adversary cannot forge valid messages to bias the honest nodes to make inconsistent decisions.

P3 (Blind-box Computation): F1 ensures that all intermediate states of the protocol's computation are hidden from the OS. Leveraging F2, the provided randomness is also hidden from the OS. This guarantees that the input state is hidden along with the intermediate states of the protocol's execution We say in this case that the computation is a *blind-box computation*. As the adversarial node does not know the random number and given that the output of the computation is encrypted between the Enclave and the OS, she cannot selectively omit or drop messages based on their contents. Note that an important part of instantiating such a blind-box computation is the ability to instantiate a secure channel between two or more enclaves. In fact, enclaves can agree on a shared key to establish a secure channel using Diffie-Hellman key exchange. Nodes can then encrypt all the messages (including program's intermediate input / output) transmitted between each other to provide confidentiality against malicious OSes. Note that, establishing such a shared key in the enclaved setting is slightly weaker than the standard byzantine model, as the malicious operating system cannot access the shared secret keys and decrypt the exchanged messages due to F1. With P1 - P3, we can reduce the byzantine model to a restricted model, where an adversarial node can only replay, omit and delay messages.

P4 (Halt-on-Divergence): To mitigate selective omission based on nodes' identities (A3), we enforce a security mechanism called halt-on-divergence [7]. This property halts any malicious node deviating from the protocol under some given condition. As an instance, if an adversarial node sends a message, but

[7] Halt-on-divergence captures a general mechanism called early stopping that is often used in securing against general-omission model [193, 198].

does not receive adequate responses, it will be forced to leave the current protocol execution. Halt-on-divergence mechanism should be incorporated through a specific acknowledgment protocol instantiation in such a way that every malicious node will be forced to leave if the acknowledgment is not verified. In particular, we introduce an acknowledgment scheme where every receiver acknowledges the sender on receiving every valid message. A message sent over a secure channel is considered valid only if it contains the expected sequence and round number. Naturally, an acknowledgment is not sent for a replayed, omitted or delayed message. Since all honest receivers will reply with acknowledgment (ACK) messages on receiving valid messages, an honest sender should at least receive $t + 1$ ACK messages. Any node receiving less than $t + 1$ ACK messages will halt its execution and leave the network.

The key idea here is to penalize any deviating adversary by churning the node out of the network. This effectively "sanitizes" the network. Thus, to remain a part of the network, every peer should send valid messages to the majority of the network. This property also aids honest nodes in the protocol to decide the final output early and finish the execution immediately.

P5 (Lockstep Execution): F4 allows us to realize a synchronized network across all rounds of a protocol. Each peer uses F4 to decide the correct value of the ongoing round and inserts this round number in all the sent messages. To detect *delay* attacks (A4), a peer simply matches the round number present in an incoming message with the current round number. This defense is hard in the byzantine model with public-key infrastructure even if it supports F1, since the OS can tamper with the relative time to either increase or decrease the rounds of a node. Therefore, having access to a trusted elapsed time functionality allows to perform lockstep execution and detect delay attacks in the restricted model.

P6 (Message Freshness): Similar to [168], we use sequence numbers to ensure message freshness and therefore defend against replay attacks (A5). The main challenge lies in ensuring secure exchange of the initial sequence numbers

for each peer and ensuring that the sequence number remains untampered with during the entire intermediate states of the protocol execution. Using the secure channel, the peers securely exchange a nonce or a *sequence number*, which is incremented sequentially by the peer. The nonce is generated using F2 supported by SGX. This prevents the malicious adversary from tampering the initial nonce value to its own advantage. Note that the keys and initial sequence numbers exchange occur only once during the setup phase. If an adversarial node restarts or relaunches its enclave, all the data in the enclave will be removed. Since the enclave does not have the valid sequence number and round number, it cannot re-join the same or any on-going execution, which is equivalent to be considered as a new node for the protocol.

5.3.2 Overview of Our Results

In this work, we achieve the results below.

R1: *By enforcing (P1 - P6), we reduce the byzantine model to the general-omission model, where the byzantine adversary does not have any additional advantage than omitting messages.*

By enforcing P1 - P3, we first reduce byzantine model to a restricted model, in which byzantine nodes can only delay / omit / replay messages. Due to space constraints, we defer the formalization and proof to Section 5.7. We believe that the formalization, while based on traditional cryptographic primitives, provides a new conceptual framing of SGX-enabled CPUs security features, and may be of independent interest.

By applying P5 and P6, we further confine the adversarial nodes into the general-omission model in the synchronous setting. Previously, numerous protocols such as reliable broadcast and byzantine agreement were proposed for general-omission model from a theoretical standpoint, yet considered unrealistic at that time [99, 145, 193, 198]. By enforcing our properties, we want to emphasize that this entire class of protocol is realizable in the current SGX-enabled machines.

R2: *We propose an efficient reliable broadcast protocol (*ERB*) with early stopping, which improves communication complexity from $O(N^3)$ to $O(N^2)$ (refer to Section 5.4).*

For this result, we leverage four properties. First, P1 - P3 ensure that the adversarial nodes cannot forge messages and deviate from the execution of the protocol. Second, we leverage P4 to show that ERB can broadcast a message to the entire network in $\min\{f + 2, t + 2\}$ rounds with better performance as shown in Table 5.1. We further illustrate that our properties are generic and can improve the efficiency of traditional protocols of reliable broadcast. Due to space limitations, we detail our findings in Section 5.8.

R3: *We propose a new unbiased random number generation protocol (*ERNG*) with communication complexity $O(N^3)$ for the basic version, or $O(N \log N)$ for the optimized one (refer to Section 5.5).*

P3 ensures that the byzantine nodes do not know the random number and cannot selectively drop the unfavorable one. P5 disallows the adversary to look ahead and compute the final result before the last round. With P3 and P5, our unoptimized ERNG solution directly runs our ERB protocol as a sub-routine on the entire network to agree on a random number generated using F2. It has round and communication complexity of $O(N)$ and $O(N^3)$, respectively. We present an optimized version of ERNG by reducing the byzantine fraction from $\frac{N}{2}$ to $\frac{N}{3}$, and forming a cluster of peers within the network. Leveraging the trusted randomness F2 and P3, we can sample a small set of nodes forming a representative cluster. The ERB protocol is executed within this small cluster to generate the final unbiased random number. The round and communication complexity of this optimized ERNG is further reduced to $O(\log N)$ and $O(N \log N)$. Note that the optimized version of ERNG only applies when the size of the network is large enough.

5.4 Enclaved Reliable Broadcast Protocol

We propose an *enclaved reliable broadcast* (ERB) in the synchronous model using SGX features.

5.4.1 Preliminaries

The transmitted message, val, between any two peers has the following format:

$$\mathsf{val} := \langle \mathsf{type}, \mathsf{id}, \mathsf{seq}, m, \mathsf{rnd} \rangle,$$

where $\mathsf{type} \in \{\mathsf{INIT}, \mathsf{ECHO}, \mathsf{ACK}\}$ and rnd represents the current round of the ERB protocol. If $\mathsf{type} = \mathsf{INIT}$, then the initiator peer $\mathsf{id}_{\mathsf{init}}$ is initiating the broadcast by sending the message m with sequence number $\mathsf{seq}_{\mathsf{init}}$ at round rnd. If $\mathsf{type} = \mathsf{ECHO}$, it means that its sender knows that $\mathsf{id}_{\mathsf{init}}$ has sent m, as it has already received either a value with INIT or ECHO for the first time. Finally, if $\mathsf{type} = \mathsf{ACK}$, it means that the peer acknowledges that it has already received either INIT or ECHO values from the sender.

We introduce three functions Halt, Multicast and Wait defined as follows:

- $\mathsf{Halt}(\mathsf{st})$: is a function that sets the state st to $\perp$.[8]
- $\mathsf{Multicast}\big(\mathsf{id}_i, \mathsf{val}\big)$: is a functionality that multicasts the value val from the sender p_i to the receiver p_j, for all $j \in [N] \setminus \{i\}$.
- $\mathsf{Wait}(\tau)$: is a function that has as an input the current elapsed time τ in the ongoing round, and suspends the protocol for $(2\Delta - \tau)$ seconds.

Note that Halt function enforces the *halt-on-divergence* property (P4) that we have introduced in Section 5.3. For the sake of exposition, we write $\mathsf{Wait}(\mathsf{rnd})$ in the code description, we say in this case that the protocol waits until the end of the round rnd.

[8]Note that when the state of the node is set to $\perp$ the node halts on-divergence and is ejected from the P2P network $\mathcal{P}$.

5.4.2 ERB details

Prior to running the very first instance of the ERB protocol, there is a setup phase. The setup is performed whenever the program (ERB) needs to be updated or changed. We detail the setup phase followed by the explanation of our algorithm.

Setup Phase: Every pair of sender and receiver peer use remote attestation (F3) along with enclaved execution (F1) to verify the correctness of the execution, and therefore enforcing P1 - P3. Then they establish a secure channel using Diffie-Hellman key exchange. This setup enforces P1 - P3, which restricts the byzantine nodes to only omit, replay and delay messages. Next, each peer picks at random a sequence number such that $\mathsf{seq}_s, \mathsf{seq}_r \xleftarrow{\$} \{0,1\}^k$ and send it to each other. That is, every node has to store the sequence numbers of all other nodes in $\mathcal{P}$. Finally, every node sets the variable rnd to the value 1. The overhead of the setup is in $O(N^2)$ while the storage overhead per node is in $O(N)$.

Initialization Phase: An initiator node first multicasts the value $\mathsf{val} = \langle \mathsf{INIT}, \mathsf{id}_{\mathsf{init}}, \mathsf{seq}_{\mathsf{init}}, m, \mathsf{rnd}) \rangle$, where $\mathsf{seq}_{\mathsf{init}}$ is the sequence number of the initiator node, and rnd is the round number. The round rnd is first initialized to 1, the enclave will now increment the rnd after every 2Δ seconds—we take advantage of the elapsed time feature of SGX to tie a round to an interval of 2Δ seconds.

Echo Phase: Until round $t + 2$, if a node receives an INIT or ECHO message for the first time, it performs the following actions: (1) start the local clock and initialize the round rnd to 1, the round will increment every 2Δ seconds, (2) if both rnd and seq are consistent with the expected values, it will store the message m, else it just ignores it and treats it as an omitted message. If there is no delay or replay detected, then it multicasts an ECHO message to all nodes at the end of the current round. If the node has already received a valid ECHO message from a distinct node, it will only add the sender's identifier into the set SS_{echo}. Recall that at the end of the setup phase, all honest nodes have the same copy of the sequence number of all honest nodes. After every valid instance of

the protocol, nodes will increase all sequence numbers by 1.

Decision Phase: If the node has received at least $t + 1$ correct ECHO messages from distinct nodes, i.e., $|SS_{\mathsf{echo}}| = t + 1$, then the node accepts $\hat{m}$. After $t + 2$ rounds, if the node has not received adequate distinct ECHO messages, it accepts $\hat{m} := \perp$. Every multicast requires the node to receive at least $t + 1$ ACK messages, else the node churns out itself using the Halt function.

5.4.3 Analysis

Here we give an intuition about the security guarantees of ERB and how it prevents attacks in the restricted model (ensured by P1 - P3), where byzantine nodes can only delay, replay and omit messages.

Byzantine nodes can arbitrarily delay a message. With the round number in every INIT or ECHO message, the receiver can easily determine the round in which a particular message has been sent. A node in ERB will only accept messages that have been sent in the beginning of the same round. This way, if a message has been purposely delayed for over than a round, the message will not be accepted by any node. Recall that the SGX has access to a relative time (F4) untampered by the OS, and then it can tie any round to a particular interval for enforcing lockstep execution (P5). The ERB protocol will not send an ACK on encountering such a delayed message. Moreover, byzantine nodes can also replay messages from previous ERB instances in the current ERB to introduce inconsistent views among honest nodes, as in Attack A3. However, ERB protocol enforces that every message contains a unique sequence number seq for message freshness (P6). Whenever an enclave instantiates the ERB protocol it increments the sequence number. For every instance, the receiver $\mathsf{Enclave}_r$ validates the freshness of a received message. It checks whether the sequence number is incremented as compared to previous instance of the protocol. This step is bound to execute due to the enclaved execution feature (F1).

Further, if a byzantine sender decides to omit a message, then according

Algorithm 7: ERB: Enclaved reliable broadcast protocol (for a node id_i with the initiator $\mathsf{id}_{\mathsf{init}}$ sending a message m and a sequence number $\mathsf{seq}_{\mathsf{init}}$).

Input: A P2P network $\mathcal{P}$ composed N nodes, a message m and a sequence number $\mathsf{seq}_{\mathsf{init}}$ for the initiator $\mathsf{id}_{\mathsf{init}}$

Output: A message $\hat{m}$

- initialization: $\hat{m} \leftarrow \perp$; $SS_{\mathsf{echo}} \leftarrow \emptyset$; $\mathsf{rnd} \leftarrow 1$
- upon $\mathsf{id}_i = \mathsf{id}_{\mathsf{init}}$ and $\mathsf{st}_i \neq \perp$:
  $\hat{m} \leftarrow m$;
  $SS_{\mathsf{echo}} \leftarrow SS_{\mathsf{echo}} \cup \{\mathsf{id}_{\mathsf{init}}\}$;
  $\mathsf{Multicast}\big(\mathsf{id}_{\mathsf{init}}, \langle \mathsf{INIT}, \mathsf{id}_{\mathsf{init}}, \mathsf{seq}_{\mathsf{init}}, m, \mathsf{rnd}\rangle\big)$;
- **for** $\mathsf{rnd} \leq t + 2$ **do**
  - upon receiving $\langle \mathsf{INIT}, \mathsf{id}_{\mathsf{init}}, \mathsf{seq}, m, \mathsf{rnd}'\rangle$ from $\mathsf{id}_{\mathsf{init}}$:
    if $\mathsf{rnd}' = \mathsf{rnd}$ and $\mathsf{seq} = \mathsf{seq}_{\mathsf{init}}$ **then**
      send $\langle \mathsf{ACK}, \mathsf{id}_{\mathsf{init}}, \mathsf{seq}, H(m), \mathsf{rnd}\rangle$ to $\mathsf{id}_{\mathsf{init}}$;
      $\hat{m} \leftarrow m$;
      $SS_{\mathsf{echo}} \leftarrow SS_{\mathsf{echo}} \cup \{\mathsf{id}_{\mathsf{init}}\} \cup \{\mathsf{id}_i\}$;
      $\mathsf{Wait}\big(\mathsf{rnd}\big)$ then $\mathsf{Multicast}\big(\mathsf{id}_i, \langle \mathsf{ECHO}, \mathsf{id}_{\mathsf{init}}, \mathsf{seq}, m, \mathsf{rnd}+1\rangle\big)$;
    end
  - upon receiving $\langle \mathsf{ECHO}, \mathsf{id}_{\mathsf{init}}, \mathsf{seq}, m, \mathsf{rnd}'\rangle$ from peer id_j:
    if $\mathsf{rnd}' = \mathsf{rnd}$ and $\mathsf{seq} = \mathsf{seq}_{\mathsf{init}}$ **then**
      send $\langle \mathsf{ACK}, \mathsf{id}_{\mathsf{init}}, \mathsf{seq}, H(\mathsf{val}), \mathsf{rnd}\rangle$, where $\mathsf{val} = \langle \mathsf{ECHO}, \mathsf{id}_{\mathsf{init}}, \mathsf{seq}, m, \mathsf{rnd}\rangle$ to peer id_j;
      if $\hat{m} = \perp$ **then**
        $\hat{m} \leftarrow m$;
        $SS_{\mathsf{echo}} \leftarrow SS_{\mathsf{echo}} \cup \{\mathsf{id}_i\}$;
        $\mathsf{Wait}\big(\mathsf{rnd}\big)$ then $\mathsf{Multicast}\big(\mathsf{id}_i, \langle \mathsf{ECHO}, \mathsf{id}_{\mathsf{init}}, \mathsf{seq}, m, \mathsf{rnd}+1\rangle\big)$;
      end
      if $\mathsf{id}_j \notin SS_{\mathsf{echo}}$ **then**
        $SS_{\mathsf{echo}} \leftarrow SS_{\mathsf{echo}} \cup \{\mathsf{id}_j\}$
        if $|SS_{\mathsf{echo}}| = N - t$ **then**
          accept $\hat{m}$;
        end
      end
    end
  - upon $\mathsf{Multicast}(\mathsf{id}_i, \mathsf{val})$:
    send val to id_k, for all $k \in [N] \setminus \{i\}$;
    receive N_{ack} acknowledgements $\langle \mathsf{ACK}, \mathsf{id}_{\mathsf{init}}, \mathsf{seq}, H(\mathsf{val}), \mathsf{rnd}'\rangle$, where $\mathsf{rnd}' = \mathsf{rnd}$ and $\mathsf{seq} = \mathsf{seq}_{\mathsf{init}}$;
    if $N_{\mathsf{ack}} < t$ **then**
      $\mathsf{Halt}(\mathsf{st}_i)$;
    end
  - $\mathsf{rnd} \leftarrow \mathsf{rnd} + 1$;
- **end**
- **if** $|SS_{\mathsf{echo}}| < N - t$ **then**
    $\hat{m} \leftarrow \perp$;
    accept $\hat{m}$;
- **end**
- $\mathsf{seq}_{\mathsf{init}} \leftarrow \mathsf{seq} + 1$;

to Algorithm 7, it will not receive a corresponding ACK message as the sent messages never reach the receiver peer. The sender $\mathsf{Enclave}_s$ detects that the underlying OS_s is byzantine if it does not receive at least $t + 1$ ACK messages. On failing to receive majority ACK messages, $\mathsf{Enclave}_s$ executes the Halt function

as per our algorithm and churns itself out of the network based on our halt-on-divergence property (P4). We state our main theorem below and defer the detailed proof to Section 5.9.1.

Theorem 5.4.1. *If* $N \geq 2t + 1$, ERB *is a* reliable broadcast protocol *as defined in Definition 5.2.1.*

Proof Outline. We prove that ERB meets all the requirements of reliable broadcast, according to Lemma 5.9.2 - 5.9.7 in Section 5.9.1. □

ERB **Performance Analysis.** Algorithm 7 has a worst-case round complexity equal to $t + 2$ with message complexity in $O(N^2)$ and $t < N/2$ byzantine nodes. This only occurs if the byzantine peers delay the instance for t rounds before sending the message to at least one honest node. However, in this case, the round complexity is equal to $f + 2$ rather than $t + 2$ as the delay is only in function of the number of byzantine nodes f. On the other hand, byzantine nodes can also decide to not send the message to any honest node, and then the round complexity is $t + 2$ with message complexity equal to $O(t)$. Additionally, we study how to use our sanitization technique to further optimize the round complexity in Section 5.9.2.

5.5 Enclaved Random Number Generation

We present our algorithm that generates an unbiased common random number called *enclaved random number generation (*ERNG*)*.

5.5.1 Unoptimized ERNG

We detail our unoptimized ERNG in Algorithm 8. At a higher level, every node generates a random number from the enclave, and then performs ERB protocol to broadcast to every node. According to Theorem 5.9.1, all honest nodes in this case will receive the random numbers from all honest nodes after

Algorithm 8: Unoptimized-ERNG: Unoptimized enclaved unbiased random number generation protocol executed by peer p_i.

Input: A P2P network $\mathcal{P}$ composed of N nodes
Output: A unbiased random number r

- initialization: $SS_{\mathsf{final}} \leftarrow \emptyset$; $\mathsf{rnd} \leftarrow 1$
  for $\mathsf{rnd} \leq t + 2$ **do**
  - **if** $\mathsf{rnd} = 1$ **then**
    initiate ERB with inputs $m_i \xleftarrow{\$} \{0,1\}^k$ and seq_i;
    end
    if $2 \leq \mathsf{rnd} \leq t + 2$ **then**
    execute ERB instances and wait for the output ($\mathsf{M}_i = \{\hat{m_1}, \cdots, \hat{m_{l_i}}\}$);
    end
    $\mathsf{rnd} \leftarrow \mathsf{rnd} + 1$;

  end
- $SS_{\mathsf{final}} \leftarrow \mathsf{M}_i$;
  $\mathsf{seq}_j \leftarrow \mathsf{seq}_j + 1$, for all $j \in [N]$
  accept $r = \bigoplus_{v \in SS_{\mathsf{final}}} v$.

$t + 2$ rounds, and may eventually receive several random numbers from other byzantine nodes. According to Lemma 5.9.2, for each ERB instance, every honest node will accept a random number from its initiator or $\perp$ so that all honest nodes have the same final set SS_{final} of random numbers. By performing exclusive disjunction (or XOR) of all received random numbers, every honest node obtains an *unbiased common* random number eventually.

Unbiasedness and Randomness Analysis. We describe the main intuition behind the common unbiasedness and randomness of our ERNG's output and defer formal details to Section 5.9.3. To bias the random value, the adversary may perform several attacks. It can first try to directly forge the random number, however, this is restricted as per execution integrity (P1) and message integrity (P2) enforced by F1 and F3. An adversary can force the program to generate a local random number of its choice. However, each enclave generates an unbiased random number from SGX-enabled CPU instruction `RDRAND` using F2. It is not possible to bias the source of randomness based on the hardware guarantees of SGX.

Our blind-box computation (P3) together with the secure channel guarantee that an adversary cannot selectively omit its random number based on its value with the goal to bias the output. Therefore, the adversary cannot infer the ran-

dom numbers submitted by other honest peers during the execution. Note that, the defense against replay attacks is already provided by the ERB protocol.

One adversarial strategy is to learn the final output and then decide whether to participate or not in the protocol, as in Attack A4. From Algorithm 8, all honest nodes output the final value after round $t+2$. In order to bias the final value, the adversary should perform the following steps within round number $t+2$: (1) learn the XOR of random numbers from honest nodes, (2) decide whether to participate or not based on the final value, (3) and multicast its number to honest nodes. In Algorithm 8, the final XOR operation executes only when $\mathsf{rnd} > t+2$. The execution integrity (P1) ensures sequential execution of our protocol. This property restricts the adversary from directly jumping to the step that computes the XOR operation and learn the result before other honest nodes generate the final output. Next, the lockstep execution (P5) enforced by the elapsed time feature (F4) allows us to bound the time for each round, even on a byzantine peer. Therefore, the adversary cannot look ahead and compute the final output before the last round. If the adversary decides to delay its own random number based on the computed final value, the adversarial random number will be neglected by all honest peers as it will reach after $t+2$ round. Combining P1, P5 and P3, it is not possible for the byzantine adversary to achieve steps (1) and (3) simultaneously.

For clarity and without any loss of generality, we model Algorithm 8 as a multi-variate function $G : \{0,1\}^{k \times N} \rightarrow \{0,1\}^k$ that maps N elements in $\{0,1\}^k$ to one element in $\{0,1\}^k$ such that $G(x_1, \cdots, x_N) = \bigoplus_{i=1}^{N} x_i$.

Theorem 5.5.1. *The bias of* G $\beta(G) = 1$.

We defer the proof to Section 5.9.3.

5.5.2 Optimized ERNG

Next, we illustrate the main steps for ERNG with optimizations. In this section, we consider that at most $t \leq \frac{N}{3}$ nodes of the network can be byzantine. ERNG

Algorithm 9: ERNG: Enclaved unbiased random number generation protocol executed by peer p_i.

Input: A P2P network $\mathcal{P}$ composed of N nodes
Output: A unbiased random number r

- initialization: $SS_{\mathsf{M}} \leftarrow \emptyset$; $SS_{\mathsf{final}} \leftarrow \emptyset$; $SS_{\mathsf{chosen}} \leftarrow \emptyset$; $\mathsf{rnd} \leftarrow 1$

for $\mathsf{rnd} \leq \gamma + 4$ **do**

if $\mathsf{rnd} = 1$ **then**

- every peer p_i compute $r_i \xleftarrow{\$} \{0, \cdots, \frac{N}{2\gamma} - 1\}$;

if $r_i = 0$ **then**

$\mathsf{Multicast}(\mathsf{id}_i, \mathsf{val})$, where $\mathsf{val} = \langle \mathsf{CHOSEN}, \mathsf{id}_i, \mathsf{seq}_i, \perp, 1 \rangle$;

$SS_{\mathsf{chosen}} \leftarrow \{\mathsf{id}_i\}$;

end

- upon receiving $\mathsf{val} = \langle \mathsf{CHOSEN}, \mathsf{id}_j, \mathsf{seq}_j, m_j, \mathsf{rnd}_j \rangle$

if $\mathsf{type} = \mathsf{CHOSEN}$ **and** $\mathsf{rnd}_j = 1$ **and** $\mathsf{seq}_j = \mathsf{seq}_i$ **then**

$SS_{\mathsf{chosen}} \leftarrow SS_{\mathsf{chosen}} \cup \{\mathsf{id}_j\}$;

end

end

- **if** $r_i = 0$ and $\mathsf{rnd} = 2$ **then**

compute $r'_i \xleftarrow{\$} \{0, \cdots, \gamma' - 1\}$;

if $r'_i = 0$ **then**

initiate ERB with inputs $m_i \xleftarrow{\$} \{0,1\}^k$, seq_i and peers in SS_{chosen};

end

$\mathsf{seq}'_j \leftarrow \mathsf{seq}_j$, for all $\mathsf{id}_j \in SS_{\mathsf{chosen}}$;

end

if $r_i = 0$ and $3 \leq \mathsf{rnd} \leq \gamma + 2$ **then**

execute ERB instances and wait for the output;

end

if $r_i = 0$ and $\mathsf{rnd} = \gamma + 3$ **then**

$\mathsf{Wait}\big(\mathsf{rnd}\big)$ then obtain $\mathsf{M}_i = \{\hat{m}_1, \cdots, \hat{m}_{l_i}\}$;

$\mathsf{seq}_j \leftarrow \mathsf{seq}'_j$, for all $\mathsf{id}_j \in SS_{\mathsf{chosen}}$;

end

if $\mathsf{rnd} = \gamma + 4$ **then**

- **if** $r_i = 0$ **then**

$SS_{\mathsf{M}} \leftarrow SS_{\mathsf{M}} \cup \{\mathsf{M}_i\}$;

$\mathsf{Multicast}\big(\mathsf{id}_i, \langle \mathsf{FINAL}, \mathsf{id}_i, \mathsf{M}_i, \mathsf{seq}_i, \gamma + 4 \rangle\big)$;

end

- upon receiving $\mathsf{val} = \langle \mathsf{FINAL}, \mathsf{M}_j, \mathsf{seq}'_j, \mathsf{rnd}_j \rangle$:

if $\mathsf{rnd}_j = \gamma + 4$ **and** $\mathsf{seq}'_j = \mathsf{seq}_j$ **then**

$SS_{\mathsf{M}} \leftarrow SS_{\mathsf{M}} \cup \{\mathsf{M}_j\}$;

if *# of* $\mathsf{M}_\kappa \geq \gamma + 1$ *where* $\mathsf{M}_\kappa \in SS_{\mathsf{M}}$ **then**

$SS_{\mathsf{final}} \leftarrow \mathsf{M}_\kappa$;

accept $r = \bigoplus_{v \in SS_{\mathsf{final}}} v$.

end

end

end

$\mathsf{rnd} \leftarrow \mathsf{rnd} + 1$;

end

- $\mathsf{seq}_j \leftarrow \mathsf{seq}_j + 1$, for all $j \in [N]$;

terminates after $\gamma + 4$ rounds, where γ is a statistical parameter. The intuition behind our optimization can be formulated as follows: we notice that if we select uniformly at random nodes to constitute a representative cluster in $\mathcal{P}$,

we note that we can still guarantee w.h.p. an honest majority within this smaller representative cluster. By leveraging F2 to generate a random number and blind-box computation (P3), we can sample a set of peers forming the representative cluster. The main remaining question, therefore, is how big this cluster should be. As a starting point, note that if the cluster size is equal to $\frac{2N}{3}$, the probability of having an honest majority is equal to one. Therefore, this remark already suggests that the cluster size can be smaller. Conceptually, the optimized ERNG can be decomposed into three main steps:

Cluster Selection: The purpose of this step is to construct a representative cluster of the entire P2P network. The cluster will consist of nodes selected uniformly at random from $\mathcal{P}$. At round 1, every node picks uniformly at random a number from $\{0, \cdots, \frac{N}{2\gamma} - 1\}$[9] using SGX (F2). This operation is protected leveraging property P3 in such a way that the computation is hidden from the OS. If the random number equals 0, then the node is *chosen* to be part of the cluster, and then it multicasts a CHOSEN message to all nodes in $\mathcal{P}$. Upon receiving the CHOSEN message, every chosen node adds the identifier of the sender to its own set SS_{chosen}. The size of the set SS_{chosen} represents the size of the cluster.

ERB **Instances:** We first detail a pseudo-solution and then detail our main construction in Algorithm 9. In round 2, the nodes constituting the cluster will each generate a random number and broadcast it *only* to the nodes constituting the cluster (i.e., peers' identifiers in SS_{chosen}). That is, every node in the cluster will run an independent ERB instance. The intuition behind these multiple instances is the following: for the broadcast to be effective, at least one broadcast instance has to succeed in that the accepted message is different from $\perp$. However, the complexity of such solution is cubic in $O(|SS_{\mathsf{chosen}}|^3)$ which can be a handicap in term of efficiency. As a solution, we incorporate a two-phases clustering. The idea behind this choice is the following: in order to generate a random number

[9] N is much larger than γ, and we assume $\frac{N}{2\gamma} - 1$ is $\lfloor \frac{N}{2\gamma} - 1 \} \rfloor$.

we only require one honest node to output a random number r (otherwise the ERNG protocol may output $\perp$). We can then proceed to select just a few number of nodes to perform the ERB protocol. As long as at least one of these nodes is honest, the correctness of our ERNG holds. Concretely, to generate the second representative cluster, we perform the following: from nodes in SS_{chosen}, we uniformly pick at random a value from $\{0, \cdots, \gamma' - 1\}$, where γ' is a parameter in function of γ that verifies $\gamma' \leq \gamma$. The peers that output a random number equal to zero will be the only peers able to initiate the ERB protocol. We will show that this strategy will greatly decrease the communication complexity and defer its analysis to Section 5.9.4. Note that this phase lasts for $\gamma + 2$ rounds when all ERB instances terminate.

Selection Decision: At the end of the broadcast phase, the node of the clusters will have each a set containing eventually several random numbers. Note that, as ERB is a reliable broadcast primitive, we know that all honest peers in the cluster will have the same set of random numbers. Once a node in $\mathcal{P}$ receives at least $\gamma + 1$ sets of random numbers, M_κ, originating from the nodes in the cluster, it will output the set M_κ as SS_{final}. All honest nodes will output the same set under the assumption that there is a majority of honest nodes in the cluster. Finally, the random number equals the XOR value of all random numbers in SS_{final}.

5.5.3 Analysis

We present the proofs for the Lemma and Theorems below in Section 5.9.4.

Lemma 5.5.1. *If up to $t = \frac{N}{3}$ nodes are byzantine, then with at least $1 - \mathsf{negl}(\gamma)$ probability, the representative cluster has more than γ honest nodes, and less than γ byzantine nodes.*

Theorem 5.5.2. Agreement: *All honest nodes eventually agree on the same common set SS_{final} in* ERNG.

Theorem 5.5.3. Unbiasedness: *The output of the* ERNG *protocol is an unbiased random number.*

ERNG **Performance Analysis.** Note that in ERNG, $O(\gamma)$ nodes will be chosen to form the first representative cluster and therefore run $O(\gamma)$ Multicast functions. The communication complexity of this first step is $O(\gamma^2)$. Then, among this first representative cluster, a second cluster will be composed such that all nodes of this cluster will run each an ERB instance. If the size of the second representative cluster is $O(\sqrt{\gamma})$ (as shown in Corollary 5.9.1 in Section 5.9.4), then the communication complexity of this step is $O(\gamma^2 \cdot \sqrt{\gamma})$. Finally, the member of the first representative cluster will multicast the output of the ERB instances to all peers in $\mathcal{P}$. The communication complexity of this final step is $O(N \cdot \gamma)$. That is, overall, the communication complexity of ERNG equals $O(N \cdot \gamma + \gamma^{\frac{5}{2}})$. Based on Lemmas 5.9.1 and 5.9.2, if N is large such that it verifies $\gamma \in o(N)$, then we can set $\gamma \in O(\log N)$. In this case, the communication complexity and round complexity of ERNG are equal to $O(N \log N)$ and $O(\log N)$.

5.6 Evaluation

Implementation. We have implemented a prototype of ERB, unoptimized-ERNG and ERNG in C/C++ using Intel SGX's Linux SDK [26]. The implementation contains 4030 lines of code (LOC) measured using CLOC tool [10]. Our prototype implementation is open source and available online [34]. We reuse the ported OpenSSL library including cryptographic utilities (`libcrypto` available with Intel SDK), to perform Diffie-Hellman key exchange and AES encryption/decryption. We use `boost` [9] library to implement the communications between any two nodes and use Google `protobuf` libraries [39] and `rapidjson` [17] to serialize transferred data.

Experimental Setup. We use the DeterLab network testbed for our experiments [16]. It consists of 40 servers running Ubuntu 14.04 with dual Intel(R)

Xeon(R) hexacore processors running at 2.2 GHZ with 24 cores and 24 GB of RAM. All machines are connected and share the same link with the bandwidth of 128MBps. Every node in our protocol takes up to 1 - 800 MB memory which limits the maximum number of nodes to 2^{10} in our experiments. Since the trusted elapsed time (F4) is not supported by all platforms yet and we need to run multiple nodes on each machine, we use SGX simulation mode[10] for our program and use a simulated Intel attestation service (IAS).

Evaluation Methodology. To evaluate the correctness of our protocols, we measure the round complexity (time to terminate) and communication complexity (network traffic) for ERB, unoptimized-ERNG and ERNG, by varying the number of nodes from 2^2 to 2^{10}. We have highly optimized our system to handle dynamic ports allocations to handle a larger number of nodes within one machine (order of 25 nodes per machine). Part of our results reported in this section are for the *optimistic* case where all nodes behave honestly. We evaluate the round complexity of ERB while varying the number of byzantine nodes in the network up to $\frac{1}{4}$ of the entire network composed of 512 nodes. We also compare our experiment results for the traffic size with theoretical ones to verify if they match our asymptotic analysis.

5.6.1 ERB Evaluation

Honest Termination: Constant Scalability. Determining the termination of ERB is essential to validate our reliable broadcast primitive. Fig. (5.2) shows that the termination time, in the case of an honest initiator, is nearly equal to twice the value of one round. This validates our theoretical results where we show that ERB finishes in 2 rounds when the initiator is honest. The small increase at 2^8 is purely due to the bandwidth bottleneck of our testbed, as the nodes share the same link.

Traffic Size: Quadratic Scalability. Fig. (5.3) demonstrates that the commu-

[10] F4 is supported by the simulation mode in seconds.

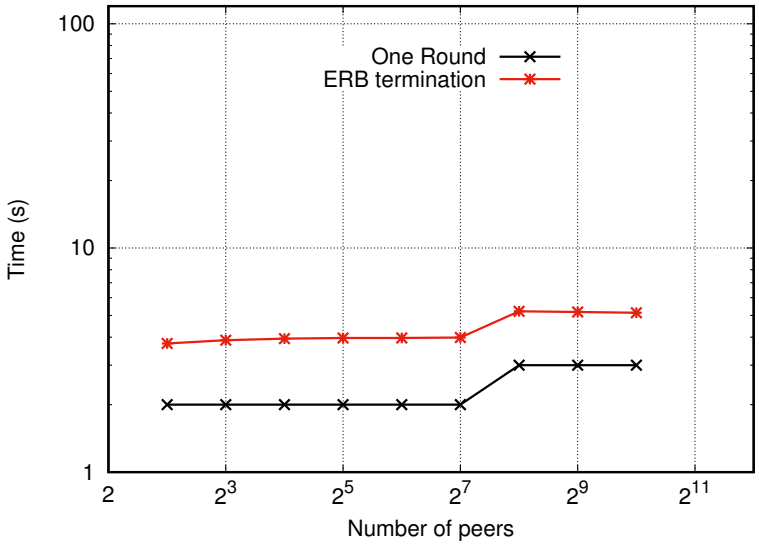


Figure 5.2: Termination time in seconds for ERB slightly increase with the number of peers.

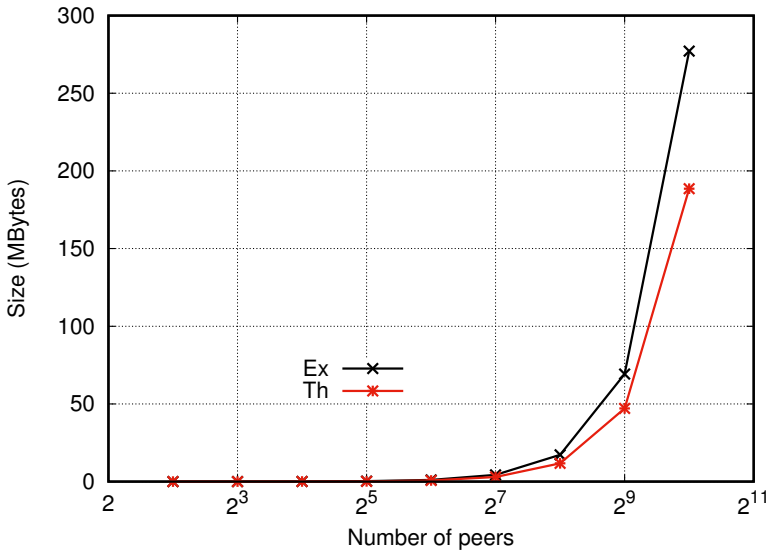


Figure 5.3: (Th) theoretical and (Ex) experimental comparisons of network overall communication bandwidth in MB for ERB in function of the number of nodes in $\mathcal{P}$.

nication complexity quadratically increases in function of the number of peers in $\mathcal{P}$ (note that the x-axis is logarithmic). The message size of INIT and ACK is around 100 Bytes and 80 Bytes, respectively. For 1024 nodes in $\mathcal{P}$, the traffic size equals 277 MB. We show that this result matches our theoretical expectation.

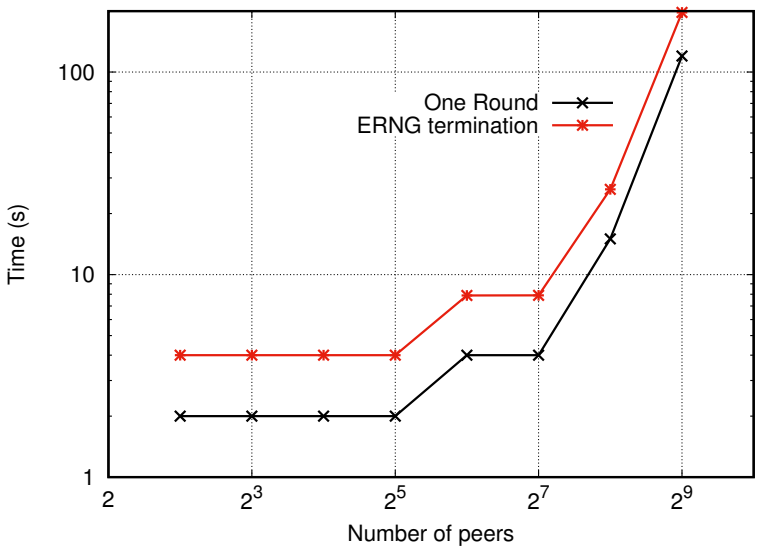


Figure 5.4: Termination time of ERNG in function of the number of nodes in $\mathcal{P}$.

5.6.2 ERNG Evaluation

Honest Termination: Limited Scalability. We show in Fig (5.4) that ERNG termination remains slightly constant from 2^2 to 2^7 and then increases afterwards. Unfortunately, this does not reflect our theoretical findings and this is mainly due to the limitation of our testbed, namely, the upper bound on the communication link of 128MBps that all nodes have to share[11]. For small values of peers N, the communication complexity of the unoptimized ERNG is cubic in N, while the optimized version is also (nearly) cubic for smaller values of N. Given a fixed bandwidth, this explains why the termination increases for larger values of N to reach 103 s for one instance of ERNG.

Traffic Size: Cubic Scalability. Fig. (5.5) demonstrates that the communication complexity cubically increases in function of the number of peers in $\mathcal{P}$ for the unoptimized ERNG. Our theoretical results back up our experimental result. For ERNG as the bandwidth links get overflowed much faster, we limited our experiments to 512 nodes. In this case, the traffic size was equal to ~ 30 GB. For the optimized ERNG, small values of the number of peers in the network

[11]Note that we had to increase the Δ as a message takes in this case more time to reach its destination.

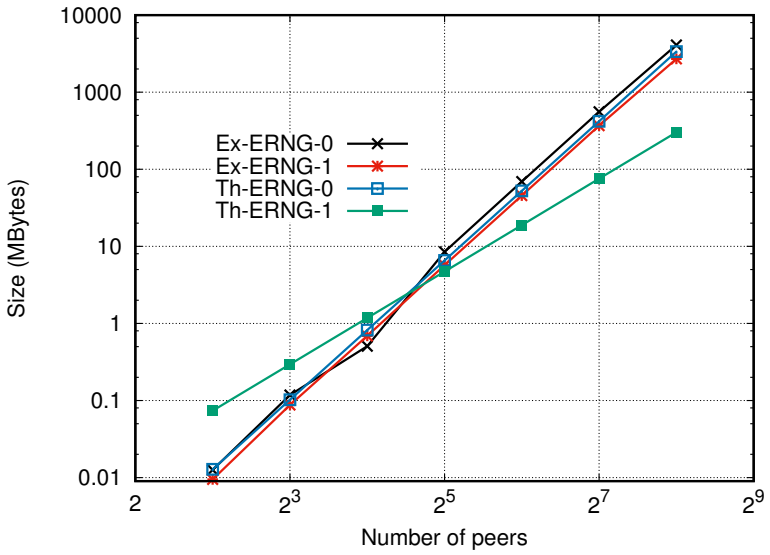


Figure 5.5: Communication overhead of ERNG in function of the number of nodes in $\mathcal{P}$.

did not allow us to optimally select a cluster size that can guarantee w.h.p. the agreement. In this case, we fix the cluster to be $\frac{2}{3}$ of the network and we show that the traffic size decreases to be equal to 12 GB for 512 nodes, a 60% improvement over the unoptimized one. Note that this result can get much better for a larger number of peers in realistic settings. Here, we draw our theoretical curve for the ideal evaluation which can be guaranteed only for larger N.

5.6.3 Byzantine case

In Fig (5.6), we show that the termination time of ERB linearly increases with the number of byzantine nodes behaving maliciously in the current instance. We gradually increase the fraction of byzantine nodes from $\frac{1}{512}$ to $\frac{1}{4}$. As a strategy of byzantine nodes, we have taken into consideration the worst-case where byzantine nodes create a chain (a byzantine sends its message to only one byzantine node each round and then gets eliminated) in order to delay the termination as much as possible. In the case of $\frac{1}{4}$ byzantine fraction, the ERB termination takes 389 seconds while it only takes 4 seconds in the honest case. For traffic size, if the number of byzantine nodes increases in the network, the communication complexity of ERB decreases as shown in Fig. (5.7). This is mainly due

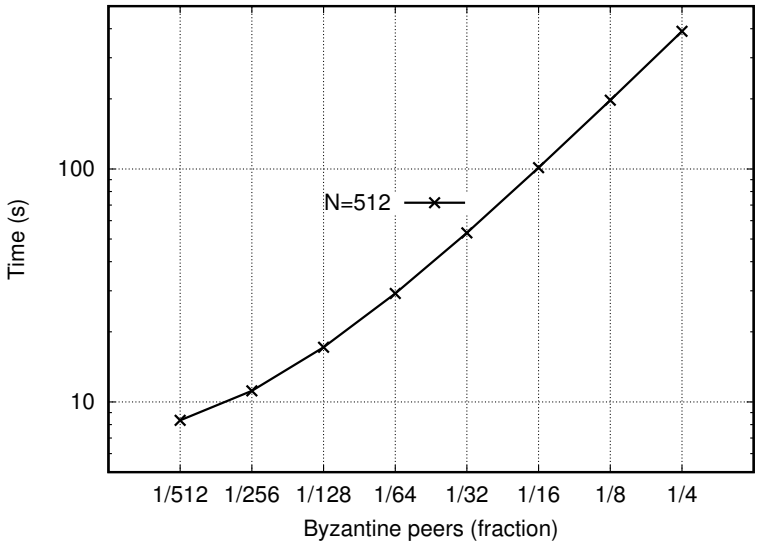


Figure 5.6: Time termination of ERB linearly increase with the number of byzantine nodes in $\mathcal{P}$.

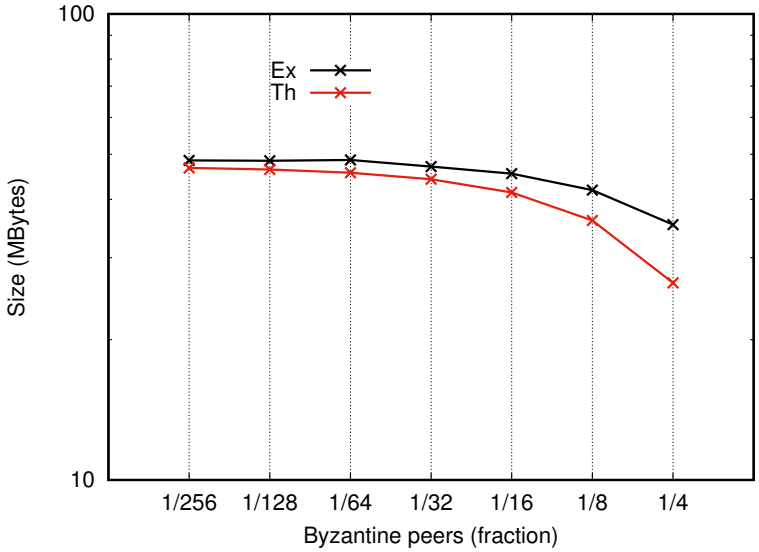


Figure 5.7: Communication overhead of ERB in function of different byzantine peers in $\mathcal{P}$.

to the halt-on-divergence property that will eject the nodes whenever it behaves maliciously. That is when an honest node multicasts a message, the eliminated byzantine node will not acknowledge this message which greatly reduces the communication complexity. For example, for $\frac{1}{4}$ byzantine fraction in a 512-node network, the traffic size equals 35 MB, while in an honest node instance, it was equal to 69 MB, a 50% decrease.

5.7 Primitives and Formal Definitions

In this section, we first start by formally defining the syntax of the communication protocol between two peers, that we denote by Peer channel. Using this definition, we next define various failure modes and primitives. Using SGX, we assume that *execution integrity (P1)* is enforced. We then show that the following properties: *message integrity & authenticity (P2)*, *blind-box computation property (P3)* can be emulated based on the Blinded channel, executing on a particular program. Then we go ahead and formally define the *halt-on-divergence (P4)* property for any program running between two peers. Finally, we show how to reduce the byzantine model to a model where a peer can only replay, omit and delay, dubbed ROD for short, given that a Blinded channel exists.

5.7.1 Peer Channel

Abstractly, a peer can be considered as the composition of two entities: an Enclave and an OS. The OS models the untrusted entity including the operating system and memory. It has access to all the system resources such as file system, network and others. The OS can arbitrarily invoke an enclave program and start its execution. The Enclave models the isolated memory space that loads the program and executes it securely. Thus, Enclave corresponds to the trusted entity of a peer. A concurrent work provides a formal study to show that SGX enclaves can be considered as a trusted entity [194]. The Enclave of the two Peers can interact with each other via their OSs. We formally define a Peer channel as a protocol, $\mathsf{Peer}^{\mathsf{ch}}$, between a sender $\mathsf{Peer}_s = (\mathsf{Enclave}_s, \mathsf{OS}_s)$ and a receiver $\mathsf{Peer}_r = (\mathsf{Enclave}_r, \mathsf{OS}_r)$. A Peer channel can be seen as a generalization of the traditional secure communication channel between two parties. The main difference is that the definition of $\mathsf{Peer}^{\mathsf{ch}}$ protocol is augmented with the program π running within the trusted Enclave. Before defining the Peer channel, we first provide a definition of a program π.

Definition 5.7.1. (Program.) *A program π is a sequence of instructions i.e., $\pi = (\pi_1, \cdots, \pi_n)$ such that the i^{th} instruction π_i takes as an input the state st_i and a message m_i and outputs a message m_{i+1} along with an updated state st_{i+1}. By convention, we write for all $m_i \in \{0,1\}^*$, $(\mathsf{st}_{i+1}, m_{i+1}) \leftarrow \pi_i(\mathsf{st}_i, m_i)$. The initial state is st_1.*

Based on the above definition, for a program π with n instructions the output $\mathtt{out}$ of π is $(\mathsf{st}_{out}, \mathtt{out}) \leftarrow \pi_n(\mathsf{st}_n, m_n)$ where st_{out} is the final state of the program. We denote the set of all such programs by Π. Note that, in a program π, an instruction with $\perp$ state as input always outputs $\perp$ i.e., $(\perp, \perp) \leftarrow \pi_i(\perp, m_i)$. Hence, if $\exists i$ such that $(\perp, \perp) \leftarrow \pi_i(\mathsf{st}_i, m_i)$, then the output of the program π is always $\perp$.

Definition 5.7.2. (Program Transcript.) *Let $\pi \in \Pi$ and messages $m_1, \cdots, m_n \in \{0,1\}^*$ such that $\mathbf{m} = (\mathbf{m_i})_{\mathbf{i} \in [\mathbf{n}]}$, for all initial states $\mathsf{st}_1 \in \{0,1\}^*$ and for all $i \geq 1$ such that $(\mathsf{st}_{i+1}, m_{i+1}) \leftarrow \pi_i(\mathsf{st}_i, m_i)$, a transcript of π with inputs st_1 and $\mathbf{m}$ denoted by $\mathsf{trans}_\pi^{\mathbf{m}}$ equals:*

$$\mathsf{trans}_\pi^{\mathbf{m}} = \big(\pi_1(\mathsf{st}_1, m_1), \cdots, \pi_i(\mathsf{st}_i, m_i), \cdots, \pi_n(\mathsf{st}_n, m_n)\big).$$

Definition 5.7.3. (Transcript Types.) *Let $\pi \in \Pi$ and $\mathsf{trans}_\pi^{\mathbf{m}}$ its transcript for a fixed message $\mathbf{m} = (\mathbf{m_i})_{\mathbf{i} \in [\mathbf{n}]}$. We say that the transcript is:*

- valid, *if* $\forall i \in [n]$, $\mathsf{st}_i \neq \perp$,
- invalid, *if* $\exists i \in [n]$, $\mathsf{st}_i = \perp$,

where $(\mathsf{st}_i, m_i) \leftarrow \pi_{i-1}(\mathsf{st}_{i-1}, m_{i-1})$.

We denote by $\mathcal{V}_\pi$ and $\mathcal{I}_\pi$, the set of all n-messages for which the transcript is valid and invalid, respectively.

Definition 5.7.4. (Peer **Channel.)** *Given $\pi_s, \pi_r \in \Pi$ are programs executing in* $\mathsf{Enclave}_s$ *and* $\mathsf{Enclave}_r$ *with st_s and st_r as respective initial states. A* Peer

channel between $\mathsf{Enclave}_s$ *and* $\mathsf{Enclave}_r$ *is tuple of four possibly interactive algorithms* $\mathsf{Peer}^{\mathsf{ch}} = (\mathsf{Init}, \mathsf{Write}, \mathsf{Transfer}, \mathsf{Read})$ *such that:*

- $(K_s, K_r) \leftarrow \mathsf{Init}\big((1^k, \mathsf{st}_s, \pi_s), (1^k, \mathsf{st}_r, \pi_r)\big)$*: is a probabilistic interactive algorithm between* $\mathsf{Enclave}_s$ *and* $\mathsf{Enclave}_r$*.* $\mathsf{Enclave}_s$ *and* $\mathsf{Enclave}_r$ *take as inputs a security parameter* k*, a program* π_s *and* π_r *and the initial state* st_s *and* st_r*, and outputs keys* K_s *and* K_r *for the sender and receiver, respectively.*

- $(\mathsf{st}'_s, \mathsf{data}'_s) \leftarrow \mathsf{Write}\big((\mathsf{st}_s, K_s, m, \pi_s), \mathsf{data}_s\big)$*: is a probabilistic interactive algorithm between* $\mathsf{Enclave}_s$ *and* OS_s*.* $\mathsf{Enclave}_s$ *has as inputs a state* st_s*, a key* K_s*, a message* m *and a program* π_s*; the* OS_s *has as the input a data block* data_s*; the algorithm outputs an updated state* st'_s *for* $\mathsf{Enclave}_s$ *and the updated data block* data'_s *for* OS_s*.*

- $(\mathtt{null}, \mathsf{data}'_r) \leftarrow \mathsf{Transfer}\big(\mathsf{data}'_s, \mathsf{data}_r\big)$*: is a probabilistic interactive algorithm between* OS_s *and* OS_r *that takes as input the data block* data'_s *and* data_r *respectively, and outputs* $\mathtt{null}$ *for* OS_s *and an updated data block* data'_r *for* OS_r

- $\big((\mathsf{st}'_r, r), \mathtt{null}\big) \leftarrow \mathsf{Read}\big((\mathsf{st}_r, K_r, \pi_r), \mathsf{data}'_r\big)$*: is a probabilistic interactive algorithm between* $\mathsf{Enclave}_r$ *and* OS_r*.* $\mathsf{Enclave}_r$ *has as inputs a state* st_r*, a key* K_r *and the program* π_r*; the* OS_r *has as the input a data block* data'_r*; the algorithm outputs an updated state* st'_r *and a response* r *for* $\mathsf{Enclave}_r$ *and* $\mathtt{null}$ *for* OS_r*.*

When $\pi_s = \pi_r = \pi$, we can write $\mathsf{Peer}^{\mathsf{ch}}_{\pi}$ to denote that $\mathsf{Peer}^{\mathsf{ch}}$ is parametrized with the program π.

5.7.2 Failure Modes

We define four progressively stronger failure modes: *honest*, *general omission*, ROD and *byzantine* modes of $\mathsf{Peer}^{\mathsf{ch}}$. Here we introduce a ROD model as an

intermediate model, wherein the adversary can only a) Replay b) Omit c) or Delay messages during a protocol, or follow it as prescribed. We particularly focus on the sender behavior for simplicity, but our definition extends to both sender and receiver. Note that to capture *delay*, we super-script the Transfer algorithm with Δ such that $\mathsf{Transfer}^{\Delta}$, to denote that the Transfer can take time Δ to complete. We denote by Replay_{π}, the set containing all values generated by Write in polynomial number of executions of program π running concurrently or earlier in time [168].

Definition 5.7.5. *(***Failure Modes.)** *Given a* Peer *channel* $\mathsf{Peer}^{\mathsf{ch}} = (\mathsf{Init}, \mathsf{Write}, \mathsf{Transfer}, \mathsf{Read})$ *between two* Peers*,* Peer_r *and* Peer_s *, for all security parameters* $k \in \mathbb{N}$ *and for all programs* $\pi, \pi_s, \pi_r, \pi' \in \Pi$ *such that*

- $(K_s, K_r) \leftarrow \mathsf{Init}\big((1^k, \mathsf{st}_s, \pi_s), (1^k, \mathsf{st}_r, \pi_r)\big)$.

For all messages $m \in \{0,1\}^*$*, for all state* $\mathsf{st}_s \in \{0,1\}^*$*, for all data block* $\mathsf{data}_s, \mathsf{data}_r \in \{0,1\}^*$ *such that* $|m| \leq |\mathsf{data}_s|$ *and* $|\mathsf{data}_s| = |\mathsf{data}_r|$*,*

- $(\mathsf{st}_s', \mathsf{data}_s^{'}) \leftarrow \mathsf{Write}\big((\mathsf{st}_s, K_s, m, \pi_s), \mathsf{data}_s\big)$*;*
- $(\perp, \mathsf{data}_r^{'}) \leftarrow \mathsf{Transfer}^{\Delta}\big(\mathsf{data}_s^{'}, \mathsf{data}_r\big)$*;*
- $\big((\mathsf{st}_r', r), \perp\big) \leftarrow \mathsf{Read}\big((\mathsf{st}_r, K_r, \pi_r), \mathsf{data}_r^{'}\big)$.

We say that

- $\mathsf{Peer}^{\mathsf{ch}}$ *is in an* **honest mode***, if we have*
  - $\mathsf{data}_s^{'} = \mathsf{data}_r^{'}$ *and,*
  - $\pi_s = \pi$*,*
  - Δ *is bounded.*
- $\mathsf{Peer}^{\mathsf{ch}}$ *is in a* **general omission mode** *, if we have*
  - $\mathsf{data}_s^{'} = \begin{cases} \perp & \text{or,} \\ \mathsf{data}_r^{'} & ; \end{cases}$

- $\pi_s = \pi$,
- Δ *is bounded.*

- $\mathsf{Peer}^{\mathsf{ch}}$ *is in a* **ROD mode***, if we have*
  - $\mathsf{data}_s^{'} = \begin{cases} \perp & \textit{or,} \\ \mathsf{data} \leftarrow \mathsf{Replay}_{\pi} & \textit{or,} \\ \mathsf{data}_r^{'} & ; \end{cases}$
  - $\pi_s = \pi$,
  - $\Delta < \infty$.

- $\mathsf{Peer}^{\mathsf{ch}}$ *is in a* **byzantine mode***, if we have*
  - $\mathsf{data}_s^{'} = \begin{cases} \phi(\mathsf{data}_r^{'}) \textit{ where} \\ \phi \in \{\{0,1\}^* \rightarrow \{0,1\}^*\} & \textit{or}, \\ \mathsf{data} \leftarrow \mathsf{Replay}_{\pi} & \textit{or}, \\ \perp; \end{cases}$
  - $\pi_s = \begin{cases} \pi & \textit{or,} \\ \pi' & \textit{where } \pi' \neq \pi; \end{cases}$
  - $\Delta < \infty$

5.7.3 Core Primitives

We define two primitives: a) Blinded channels and b) halt-on-divergence. Theorem 5.7.2, below, uses the Blinded channel primitive to demonstrate that byzantine mode reduces to the ROD mode. As shown in Section 5.4, we can further leverage additional SGX features, namely properties (P5) and (P6), to reduce the ROD model to the general-omission model. Informally, a Blinded channel guarantees confidentiality and integrity of a message over a Peer channel $\mathsf{Peer}^{\mathsf{ch}} = (\mathsf{Init}, \mathsf{Write}, \mathsf{Transfer}, \mathsf{Read})$.

Definition 5.7.6. (Blinded Channels.) *We say that* $\mathsf{Peer}^{\mathsf{ch}}$ *is* Blinded *if for all p.p.t adversaries* $\mathcal{A}$ *we have:*

$$\Pr[\mathsf{Exp}^{\mathsf{EX}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda), \textit{and},$$

$$\Pr[\mathsf{Exp}^{\mathsf{Priv}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda), \textit{and},$$

$$\Pr[\mathsf{Exp}^{\mathsf{Auth}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda) = 1] \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{EX}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)$, $\mathsf{Exp}^{\mathsf{Priv}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)$, $\mathsf{Exp}^{\mathsf{Auth}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)$ *are:*

$\underline{\mathsf{Exp}^{\mathsf{EX}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)}$:

- two parties generate keys K_s and K_r such that $(K_s, K_r) \leftarrow \mathsf{Init}\big(1^k, \pi\big)$. The entire interaction between both of the parties is saved in a transcript $\mathcal{T}$;
- compute $b \xleftarrow{\$} \{0,1\}$, if $b = 0$, then output $K = (K_s, K_r) \xleftarrow{\$} \{0,1\}^k$, otherwise output $K = (K_s, K_r) \leftarrow \mathsf{Init}\big(1^k, \pi\big)$.
- Given K and $\mathcal{T}$, $\mathcal{A}$ outputs b' and wins if $b' = b$.

$\underline{\mathsf{Exp}^{\mathsf{Priv}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)}$:

- generate keys K_s and K_r such that $(K_s, K_s) \leftarrow \mathsf{Init}\big(1^k, \pi\big)$;
- $\mathcal{A}$ has access to $\emptyset^{\mathsf{write}(K_s,.)}(.)$ and $\emptyset^{\mathsf{read}(K_r,.)}(.)$;
- $\mathcal{A}$ chooses two equal-length messages m_0 and m_1;
- compute $\mathsf{Write}\big((\mathsf{st}_s, K_s, m_b, \pi), \mathsf{data}_s\big)$ where $b \xleftarrow{\$} \{0,1\}$, and output data;
- $\mathcal{A}$ has again access to $\emptyset^{\mathsf{write}(K_s,.)}(.)$ and $\emptyset^{\mathsf{read}(K_r,.)}(.)$;
- $\mathcal{A}$ outputs b', if $b' = b$, the experiment outputs 1, and 0 otherwise.

$\underline{\mathsf{Exp}^{\mathsf{Auth}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(\lambda)}$:

- generate keys K_s and K_r such that $(K_s, K_r) \leftarrow \mathsf{Init}\big(1^k, \pi\big)$;
- $\mathcal{A}$ has access to $\emptyset^{\mathsf{write}}(K_s, .)$. $\mathcal{A}$ queries a polynomial number of messages m and eventually outputs ct, we denote by $\mathcal{Q}$ the set of all queries that $\mathcal{A}$ sent to the oracle;
- Given ct, $\emptyset^{\mathsf{write}}(K_s, .)$ outputs r. If $m \notin \mathcal{Q}$ and $r \neq \perp$. $\mathcal{A}$ outputs 1.

Attaching a program π while defining a $\mathsf{Peer}^{\mathsf{ch}}$ enables us to introduce the *halt-on-divergence* primitive as follows.

Definition 5.7.7. (Halt-on-divergence.) *Let $\pi \in \Pi$ be a program and $\mathsf{trans}^{\mathbf{m}}_{\pi}$ its transcript for a fixed n-messages $\mathbf{m}$, we say that $\mathsf{Peer}^{\mathsf{ch}}_{\pi}$* halts on-divergence *if $\mathsf{trans}^{\mathbf{m}}_{\pi}$ is invalid, i.e., $\mathbf{m} \in \mathcal{I}_{\pi}$*

5.7.4 Implementing Blinded Channel using SGX

We show how we build a $\mathsf{Peer}^{\mathsf{ch}}$ channel using SGX where $\mathsf{Enclave}_s$ and $\mathsf{Enclave}_r$ are trusted entities. Theorem 5.7.1 shows that such a $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ channel is a Blinded channel, and therefore enforces both (P2) and (P3) propertie. In particular, we consider that there is a KeyEx_{π} protocol between $\mathsf{Enclave}_s$ and $\mathsf{Enclave}_r$ that is used to generate a session key for a program π. Whenever there is a new program the key has to be re-generated. The key exchange protocol can be instantiated using Diffie-Hellman key exchange, referring to [158] Chapter 9. We use SGX remote attestation to verify that both parties run their code inside an Enclave. While this step is neither required nor captured in the $\mathsf{Peer}^{\mathsf{ch}}$ definition, it is mandatory to guarantee our *execution integrity (P1)*. We detail our instantiation in Figure 5.8, in our case, we consider that $\pi_r = \pi_s = \pi$. We denote by *parse* and *compute* the actions of decoding a string and running a particular algorithm, respectively.

Let $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a private encryption scheme, $\mathsf{MAC} = (\mathsf{Gen}, \mathsf{Auth}, \mathsf{Vrfy})$ be a message authentication code, KeyEx a key exchange algorithm, and H be a hash function. We define $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}} = (\mathsf{Init}, \mathsf{Write}, \mathsf{Transfer}, \mathsf{Read})$ as follows:

- $\mathsf{Init}\big((1^k, \mathsf{st}_s, \pi), (1^k, \mathsf{st}_r, \pi)\big)$:
  1. $\mathsf{Enclave}_s$ and $\mathsf{Enclave}_r$ fetch the hardware-embedded private keys $\mathrm{sk}_s, \mathrm{sk}_r$ from $\mathsf{st}_s, \mathsf{st}_r$, respectively;
  2. compute $(\mathsf{key}_1, \mathsf{key}_2) \leftarrow \mathsf{KeyEx}_\pi\big(\mathrm{sk}_s, \mathrm{sk}_r\big)$;
  3. $\mathsf{Enclave}_s$ outputs $K_s = \big(\mathsf{key}_1, \mathsf{key}_2\big)$ and $\mathsf{Enclave}_r$ outputs $K_r = \big(\mathsf{key}_1, \mathsf{key}_2\big)$.
- $\mathsf{Write}\big((\mathsf{st}_s, K_s, m, \pi), \mathsf{data}_s\big)$:
  1. parse $K_s = \big(\mathsf{key}_1, \mathsf{key}_2, \mathrm{sk}_s\big)$;
  2. set $(\mathsf{st}'_s, \mathsf{val}) \leftarrow \pi(\mathsf{st}_s, m)$
  3. compute $\mathrm{ct}_1 = \mathsf{SKE.Enc}(\mathsf{key}_1, \langle \mathsf{val}, H(\pi) \rangle)$ and $\mathrm{ct}_2 = \mathsf{MAC.Auth}\big(\mathsf{key}_2, \mathrm{ct}_1\big)$;
  4. set $\mathsf{data}_s = (\mathrm{ct}_1, \mathrm{ct}_2)$
  5. $\mathsf{Enclave}_s$ outputs st'_s and OS_s outputs $\mathsf{data}'_s = \mathsf{data}_s$.
- $\mathsf{Transfer}\big(\mathsf{data}'_s, \mathsf{data}_r\big)$:
  1. OS_r sets $\mathsf{data}_r = \mathsf{data}'_s$;
  2. OS_s outputs $\perp$ and OS_r outputs $\mathsf{data}'_r = \mathsf{data}_r$.
- $\mathsf{Read}\big((\mathsf{st}_r, K_r, \pi), \mathsf{data}'_r\big)$:
  1. parse $K_r = \big(\mathsf{key}_1, \mathsf{key}_2, \mathrm{sk}_r\big)$ and $\mathsf{data}'_r = (\mathrm{ct}_1, \mathrm{ct}_2)$;
  2. if $\mathsf{MAC.Vrfy}\big(\mathsf{key}_2, \mathrm{ct}_1\big) := \mathrm{ct}_2$ and $\mathsf{st}_r \neq \perp$, $\mathsf{Enclave}_r$ computes
     - $\langle r_1, r_2 \rangle = \mathsf{SKE.Dec}(\mathsf{key}_1, \mathrm{ct}_1)$;
     - if $r_2 = H(\pi)$, then compute $(\mathsf{st}'_r, r) \leftarrow \pi(\mathsf{st}_r, r_1)$, output $(\mathsf{st}_r, \perp)$ otherwise.
  3. if $\mathsf{MAC.Vrfy}\big(\mathsf{key}_2, \mathrm{ct}_1\big) \neq \mathrm{ct}_2$ or $\mathsf{st}_r = \perp$, $\mathsf{Enclave}_r$ outputs $r = \perp$ and $\mathsf{st}'_r = \mathsf{st}_r$

Figure 5.8: $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$: SGX-based Peer channel.

Theorem 5.7.1. *If* KeyEx *is a secure key exchange protocol,* SKE *is CPA secure encryption schemes,* MAC *a secure message authentication code, then* $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ *is a* Blinded Peer *channel.*

Proof Sketch. First, we want to show that the Init algorithm is a secure key exchange. Note that both parties run *two* instances of a KeyEx protocol to generate two session keys. That is, if KeyEx is a secure key exchange then $\Pr[\mathsf{Exp}^{\mathsf{EX}}_{\mathcal{A},\mathsf{Peer}^{\mathsf{ch}}}(k) = 1] \leq \frac{1}{2} + \mathsf{negl}(k)$.

Second, we need to show that $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ is a secure communication channel. Note that, we use the variant *encrypt-then-mac* which is shown in [158] Chapter 9 to provide a secure communication channel if SKE is CPA secure and MAC a secure message authentication. This ends our proof sketch. □

Theorem 5.7.2. *Assuming that* $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ *is a* Blinded *channel, then* $\mathsf{Peer}^{\mathsf{ch}}$ *in byzantine is equivalent to* $\mathsf{Peer}^{\mathsf{ch}}$ *in* ROD *mode.*

Proof Sketch. For clarity, we assume that the sender is byzantine while the receiver is not. We can apply an analogous proof for the remaining combinations as well. To prove the theorem, we need to show that the view of the honest node in the ROD and byzantine modes are the same w.h.p. under the assumption that $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ is a Blinded Peer channel. For this, it is sufficient to show the following two steps: first, that any forged message for any $\phi \in \{\{0,1\}^* \rightarrow \{0,1\}^*\} \setminus \{\mathcal{C}\}$ will not change the state of the receiver st_r, i.e., that the forged message is equivalent to receiving nothing, $\perp$, where $\mathcal{C}$ is the set composed of all functions that maps $\mathsf{data}^{'}_r$ to one of the messages in $\mathsf{Replay}_\pi \bigcup \{\mathsf{data}^{'}_r\}$. Second, we need to show that, for any valid data $\mathsf{data}^{'}_s$ output by Write, the receiver state will not change if $\pi_s \neq \pi_r$ (recall that we are assuming the receiver honest and in this case means that $\pi_r = \pi$). We detail below the two steps of the reduction:

Step 1. If $\mathsf{data}^{'}_s = \phi(\mathsf{data}^{'}_r)$ where $\phi \in \{\{0,1\}^* \rightarrow \{0,1\}^*\} \setminus \{\mathcal{C}\}$ such that $\langle \mathsf{ct}_1, \mathsf{ct}_2 \rangle = \mathsf{data}^{'}_s$. Then, we have that $\Pr[\mathsf{MAC.Vrfy}(\mathsf{key}_2, \mathsf{ct}_1) \neq \mathsf{ct}_2] \geq 1 - \mathsf{negl}(k)$ under the assumption that $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ is a Blinded channel. Based on the

$\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ in Figure 5.8, if $\mathsf{MAC.Vrfy}(\mathsf{key}_2, \mathrm{ct}_1) \neq \mathrm{ct}_2$, then $\mathsf{st}'_r = \mathsf{st}_r$ w.h.p. Note that this is valid for any program π_s. The view of the receiver is now equal:

$$\bullet\ \mathsf{data}_s^{'} = \begin{cases} \mathsf{data}_r^{'} & \forall \pi_s \\ \mathsf{data} \leftarrow \mathsf{Replay}_{\pi_s} & \forall \pi_s \\ \perp \end{cases}$$

Step 2. Now, if the node is running a new program $\pi_s \neq \pi$ such that $(\mathsf{st}'_s, \mathsf{data}_s^{'}) \leftarrow \mathsf{Write}\big((\mathsf{st}_s, K_s, m, \pi_s), \mathsf{data}_s\big)$. In this case, $\mathsf{data}_r^{'} = \mathsf{data}_s^{'} = \langle \mathrm{ct}_1, \mathrm{ct}_2 \rangle$. However, based on collision-resistance assumption of the hash function H, the malicious node cannot find any program π_s such that $H(\pi_s) = H(\pi)$.[12] In this case, if $H(\pi_s) \neq H(\pi)$, then based on the $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ protocol, $\mathsf{st}'_r = \mathsf{st}_r$, i.e., the state of the receiver does not change, which is therefore equivalent to receiving nothing, $\perp$. the view of the receiver is then equal:

$$\bullet\ \mathsf{data}_s^{'} = \begin{cases} \mathsf{data}_r^{'} & \text{for } \pi_s = \pi \\ \mathsf{data} \leftarrow \mathsf{Replay}_{\pi_s} & \text{for } \pi_s = \pi \\ \perp \end{cases}$$

Finally, we emphasize that the delay constraint ($\Delta < \infty$) remains valid for both byzantine and ROD modes. Note that this final view is exactly the same of the ROD model. Note that the same holds when we consider the receiver byzantine, or both sender and receiver byzantine. This concludes our proof. □

5.8 Rethinking Reliable Broadcast Protocols

In this section, we explain the shortcomings of classic protocols for reliable broadcast. Reliable broadcast or byzantine generals problem is formally defined in Definition 5.2.1. The crux of such protocol is that all honest nodes eventually agree on the same value, which is the one proposed by the sender (or initiator) if the initiator is honest. Reliable broadcast was first proposed by Lamport *et al.*

[12] This can also be done by signing the program for every message output by the SGX-enabled program.

in 1982, which has $O(N^t)$ message complexity and $t+1$ round complexity. The proposed protocol was also resilient upto $\frac{N}{3}$ byzantine nodes [162]. Since 1980s, reliable broadcast has been extensively studied and various protocols have been developed, which are well summarized in several excellent survey papers [160, 220]. As byzantine nodes can behave arbitrarily, these protocols have to use different techniques to prevent the impact of the proposed biased values by the byzantine nodes, which generally leads to high (like exponential or polynomial) message complexity. Moreover, it has been shown that the optimal resilience cannot exceed third the size of the network [195]. To reduce communication complexity and increase resilience, several ways have been proposed, and using digital signatures is the primary one.

5.8.1 Digital Signature Schemes

Using digital signatures denotes that a node appends its signature (signed with its private key) to every message it sends. This guarantees the integrity and authenticity of the message, which can be easily verified by the other nodes using the sender's public key. It is well known that no nodes can forge the signature of another node w.h.p. This results in restricting the behavior of byzantine nodes, which can in this case only omit to relay messages, or construct different values as an initiator. We present a reliable broadcast protocol using digital signatures in Algorithm 10 adapted from Lamport *et al.*'s work [162].

In RB_{sig}, each node signs every message it multicasts. In the first round, the initiator sends a signed message to the other nodes. Then for any round rnd, a node that receives a *valid* message will sign and forward it in the next round. A message received by a node id_i in round rnd is valid if it contains signatures from rnd different nodes except id_i. In Algorithm 10, we use $[m : \mathsf{id}_1 : \mathsf{id}_2 : ... : \mathsf{id}_j]$ to denote a message, in which m is the value signed by the initiator id_1 and $[m : \mathsf{id}_1]$ is signed by id_2, and so on. This means that id_1 sent the signed message $[m : \mathsf{id}_1]$ to a node id_2 in the first round, and id_2 sent $[m : \mathsf{id}_1 : \mathsf{id}_2]$ in the second round,

Algorithm 10: RB_{sig}: Reliable broadcast protocol using digital signatures (for a node id_i with the initiator $\mathsf{id}_{\mathsf{init}}$ sending a message m).

Input: A P2P network $\mathcal{P}$ composed N nodes, a message m for the initiator $\mathsf{id}_{\mathsf{init}}$
Output: A message $\hat{m}$

- initialization: $\hat{m} \leftarrow \perp$; $SS_m \leftarrow \emptyset$; $\mathsf{rnd} \leftarrow 1$
- **for** $\mathsf{rnd} \leq t+1$ **do**
  - **if** $\mathsf{rnd} = 1$ and $\mathsf{id}_i = \mathsf{id}_{\mathsf{init}}$ **then**
    - $\hat{m} \leftarrow m$
    - Multicast $[m : \mathsf{id}_{\mathsf{init}}]$ to all the other nodes
  - **end**
  - upon receiving $[m' : \mathsf{id}_{\mathsf{init}}]$ from $\mathsf{id}_{\mathsf{init}}$:
    $SS_m \leftarrow \{m'\}$
    Multicast $[m' : \mathsf{id}_{\mathsf{init}} : \mathsf{id}_i]$ to all nodes except $\mathsf{id}_{\mathsf{init}}, \mathsf{id}_i$ in round $\mathsf{rnd}+1$
  - upon receiving $[m' : \mathsf{id}_{\mathsf{init}} : \mathsf{id}_1 : ... : \mathsf{id}_j]$ from peer id_j:
    if $m' \notin SS_m$ **then**
      - $SS_m \leftarrow SS_m \cup \{m'\}$
      - **if** $j < t+1$ **then**
        - Multicast $[m' : \mathsf{id}_{\mathsf{init}} : \mathsf{id}_1 : ... : \mathsf{id}_j : \mathsf{id}_i]$ to all nodes except $\mathsf{id}_{\mathsf{init}}, ..., \mathsf{id}_i$ in round $\mathsf{rnd}+1$
      - **end**
    - **end**
  - $\mathsf{rnd} \leftarrow \mathsf{rnd}+1$;
- **end**
- **if** $\mathsf{rnd} > t+1$ **then**
  - **if** $|SS_m| = 1$ **then**
    - $\hat{m} \leftarrow m$ where $SS_m = \{m\}$
  - **end**
  - return $\hat{m}$
- **end**

until id_j sent the signed message $[m : \mathsf{id}_1 : \mathsf{id}_2 : ...\mathsf{id}_j]$ in the rnd^{th} round.

In RB_{sig}, the initiator id_{init} signs and sends its value to every node in the first round. If any node receives the message, it stores the value in SS_m, signs and sends it to the other nodes for the second round. For round $\mathsf{rnd} < t+1$, every node receives a valid message from other node. In the case where the received value does not belong to SS_m, then the node adds the value to SS_m and multicasts the signed message to the other nodes. After $t+1$ rounds, every node verifies whether SS_m consists of a unique value $\hat{m}$, if that holds then the node outputs $\hat{m}$, otherwise he output is the default value $\perp$.

Based on digital signatures property, the byzantine nodes cannot forge a honest-like message. Therefore, every honest node only requires one valid message sent from either one byzantine or one honest node to determine the value from the initiator. If the initiator is honest, every honest node will receive the

correct value from the initiator during the first round, and will discard invalid messages forged by byzantine nodes for the remaining rounds. If the initiator is byzantine, it can send different values to different honest nodes to bias the result. To ensure the validity of the message, after $t + 1$ rounds, at least one signature in the message is from an honest node, and the honest node will broadcast the signed message to the other honest nodes, thus all honest nodes will receive the same message. Eventually, all honest nodes received the same set of values. If multiple values are received, all honest nodes will agree on a default value, otherwise they agree on the only received value.

Using digital signatures improves network's resilience from $\frac{N}{3}$ to $N - 1$, but communication complexity remains the same $O(N^t)$. Later, an optimized algorithm using digital signatures was proposed to reduce communication complexity to $O(N^3)$ [125]. At a higher level, this improvement is achieved through a new strategy that only retransmits values which have not been previously sent. Even in this case, every node has to relay $O(N)$ messages and a message can contain $O(N)$ signatures, which results in $O(N^3)$ communication complexity for the protocol. Meanwhile, the verification of $O(N^2)$ signatures may lead to a non-negligible performance cost for the honest nodes, especially when the byzantine nodes construct and send enormous number of invalid messages to the honest nodes.

Efficiency Improvement. In the following, we discuss how our properties can lead to better asymptotics. First, by enforcing P1 - P3 and P5 - P6, we confine the byzantine nodes into the general-omission model only allowing to omit messages. We can further use P3, and secure channels in particular, to guarantee the confidentiality of the transmitted messages. In this way, when a node relays a message to the others in this model, it can append its identity instead of signing the message with its private key, which achieves the same effect of using signatures. Therefore, we circumvent the transmission of multi-signature messages and the process of verifying signatures, which reduces the

Algorithm 11: RB_{early}: Reliable broadcast protocol with early stopping (for a node id_i with the initiator $\mathsf{id}_{\mathsf{init}}$ sending a message m).

Input: A P2P network $\mathcal{P}$ composed N nodes, a message m for the initiator $\mathsf{id}_{\mathsf{init}}$
Output: A message $\hat{m}$

- initialization: $\hat{m} \leftarrow ?; \mathsf{QUIET}_i^{\mathsf{rnd}} \leftarrow \emptyset, \mathsf{M}_i^{\mathsf{rnd}}(j) \leftarrow \emptyset; \mathsf{rnd} \leftarrow 1$
- upon $\mathsf{id}_i = \mathsf{id}_{\mathsf{init}}$:
  $\hat{m} \leftarrow m$;
  Multicast m to all the other nodes
  return $\hat{m}$
- **for** $\mathsf{rnd} \leq t+1$ **do**
  - upon receiving $\langle m' \rangle$ from peer id_j:
    $\mathsf{M}_i^{\mathsf{rnd}}(j) \leftarrow \mathsf{M}_i^{\mathsf{rnd}}(j) \cup \{m'\}$
  - $\mathsf{QUIET}_i^{\mathsf{rnd}} \leftarrow \mathsf{QUIET}_i^{\mathsf{rnd}-1} \cup \{\mathsf{id}_j | \mathsf{M}_i^{\mathsf{rnd}}(j) = \emptyset\}$
    if $\hat{m} = ?$ and $\exists \mathsf{id}_j \; where \; \mathsf{M}_i^{\mathsf{rnd}}(j) \neq \emptyset$ **then**
      $\hat{m} \leftarrow \mathsf{M}_i^{\mathsf{rnd}}(j)$
      if $\mathsf{rnd} < t+1$ **then**
        Multicast $\hat{m}$ to all the other nodes in round $\mathsf{rnd}+1$
      end
    else if $\hat{m} = ?$ and $\nexists \mathsf{id}_j \; where \; \mathsf{M}_i^{\mathsf{rnd}}(j) \neq \emptyset$ **then**
      if $\mathsf{rnd} < t+1$ **then**
        if $\mathsf{rnd} > |\mathsf{QUIET}_i^{\mathsf{rnd}}|$ **then**
          $\hat{m} \leftarrow \perp$
        end
        Multicast $\hat{m}$ to all the other nodes in round $\mathsf{rnd}+1$
      else
        $\hat{m} \leftarrow \perp$
      end
    else if $\hat{m} \neq ?$ **then**
      return $\hat{m}$
  - $\mathsf{rnd} \leftarrow \mathsf{rnd} + 1$

end

communication complexity from $O(N^3)$ to $O(N^2)$ and avoids the significant computation cost (as the symmetric decryption is much cheaper than signature verification).

5.8.2 Early Stopping Schemes

Apart from reducing communication complexity, SGX can also aid to decrease round complexity. In the general-omission model, several protocols have been proposed to reduce the round complexity. We recall a classic example of reliable broadcast protocol with early stopping in $min\{f+2, t+1\}$ rounds in Algorithm 11 adapted from Perry et al.'s work [198]. When $f < t$ omission faults take place, then all honest nodes will stop by the end of round $f+2$.

In Algorithm 11, $\mathsf{M}_i^{\mathsf{rnd}}(j)$ represents the message received by id_i from id_j in

round rnd. $\mathsf{QUIET}_i^{\mathsf{rnd}}$ denotes the set of nodes from which id_i has not received a message from round 1 through round rnd. In the first round, the initiator sends a message to the other nodes and halts. For any round, if a node receives a message from another node, it stores the value in $\mathsf{M}_i^{\mathsf{rnd}}(j)$. If a node id_i does not receive any message from another node id_j for round rnd, id_j will be added into $\mathsf{QUIET}_i^{\mathsf{rnd}}$. When a node has not decided the value and it receives a value, it will set the decision as the new value and broadcasts the value to all nodes in the next round ($\mathsf{rnd} + 1 \leq t + 1$). If it does not receive any value and $\mathsf{rnd} = t + 1$, the node will decide the default value $\perp$. If the round number $\mathsf{rnd} < t + 1$ is larger than the size of $\mathsf{QUIET}_i^{\mathsf{rnd}}$, the node will send $\perp$ to all nodes in round $\mathsf{rnd} + 1$, otherwise it will send $\perp$. Finally, once the node decides its value, it halts.

The early-stopping protocol requires every node to broadcast its decision for every round, to inform the other nodes about its liveness. In this way, honest nodes can detect abnormal behaviors of malicious nodes for each round. Based on the detection, all honest nodes can halt and agree on the same value by the end of round $f + 2$, where f nodes behave maliciously (e.g., omit to replay messages). The detailed proof can be found in the work [198]. Based on the proposed broadcast detection mechanism, the protocol can early-stop. However, the communication complexity increases to be in $O(N^3)$, as every node broadcasts its value every round.

Efficiency Improvement. By leveraging the *halt-on-divergence* property (P4), we can actively stop nodes behaving maliciously, which eliminates the t-round broadcasting and reduces the communication complexity to $O(N^2)$ as well as sanitizes the network by removing the malicious nodes. For instance, if a malicious node sends a message to other nodes but omit to receive messages from over half of the nodes in the network, the node will be forced to leave the network. Therefore, any node can actively detect its own anomalous behavior instead of relying on other nodes to send messages every round to passively identify the anomaly. This can lead to reduce communication complexity for

anomaly detection from $O(N^2)$ to $O(N)$.

5.9 Security Analysis

5.9.1 ERB Analysis

In this section, we use the same terminology used in Section 5.7, namely, we assume that between any two nodes of the network, an $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ instantiation of the Blinded Peer channel is enabled. In particular, it provides us with both *message Integrity & authenticity (P2)* and *blind-box computation (P3)* properties. Throughout this section, we implicitly consider that the program is publicly available, and therefore its *execution integrity (P1)* is enforced.

Theorem 5.9.1. *If $N \geq 2t + 1$ where t is the upper bound on the number of byzantine peers, and $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ is a* Blinded Peer *channel, then* ERB *is a* reliable broadcast protocol *as defined in Definition 5.2.1 with worst-case round complexity equal to $t + 2$ and communication complexity equal to $O(N^2)$.*

Proof. We are going to gradually prove the five requirements of terminating reliable broadcast. Note that the assumption that the peers communicates using $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ implies that a byzantine node can only delay, omit or replay messages, as we have shown in Theorem 5.7.2. As long as the network is synchronous with a fixed time interval for a round to complete, delaying is then equivalent to omitting a message, as the message will not be considered by honest nodes past the round, enforcing therefore the *lockstep execution (P5)* property. Replaying a message is also ineffective as every peer is identified by a sequence number as well, that is generated by the trusted enclave in the Peer channel, and therefore enforcing the *message freshness (P6)* property. Under the assumption that $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ is a Blinded channel, we can replace all occurrences of Multicast by communication between two trusted parties. To sum up, and throughout the proof, it is valid to consider that if there is a delay, omission or replay, this will be equivalent to considering that the first party does not send any message.

Lemma 5.9.2. Validity: In ERB, if the sender is honest and accepts message m, then all honest nodes eventually accept m, otherwise if the sender is byzantine, after round $t + 2$, all honest nodes either accept the same message m or $\perp$.

Proof. In the following, we consider two different types of initiators: an honest and a byzantine peer.

(1) Let the sender be the peer p_{init} with identifier $\mathsf{id}_{\mathsf{init}}$. If p_{init} is honest, according to ERB, the sender multicasts its message m in an INIT message for the first round. All honest nodes will receive m in the first round and multicast ECHO to all nodes in the second round, as every node at this stage is going to receive m for the first time. At the end of these two rounds, every honest node will receive at least $t + 1$ ECHO messages for m from all honest nodes. According to ERB, each honest node will accept m.

(2) If the initiator is byzantine, we proceed to show the validity by contradiction. Suppose that the lemma does not hold in the byzantine case, which means that at the end of round $t + 2$, not all honest nodes agree on the same value, i.e., only a strict subset of honest nodes agree on m, but the remaining peers agree on $\perp$. According to the protocol, any node accepting m must have received at least $t + 1$ ECHO messages from different nodes. The upper bound of byzantine nodes is t, thus at least one honest node should have multicasted an ECHO message to nodes accepting m during $t + 2$ rounds. For the proof to go through, we introduce the following claims and then proceed with the contradiction case.

Claim 5.9.1. *There is at least one honest node that does not receive any* ECHO *message after* $t + 1$ *rounds.*

Proof. We prove this claim by contradiction. Suppose that all honest nodes receive an ECHO message during $t + 1$ rounds, then they multicast ECHO after receiving it. Therefore, all honest nodes will receive $t + 1$ ECHO before the end of round $t + 2$, which means that all of them accept m. This contradicts our assumption that only a strict subset of honest nodes agree on the same value

m. □

Claim 5.9.2. *No honest nodes receives* ECHO *messages after round* t.

Proof. This claim can be also shown by contradiction. Suppose one honest node receives an ECHO message before round t, it must multicast ECHO to all nodes, and all honest nodes receive it before the end of round t. However, this contradicts our Claim 5.9.1. □

We can now proceed by induction where the two claims holds for any $i \in [t-1]$. That is, we can show:

- There is at least one honest node that does not receive any ECHO message after $t+1-i$ rounds.
- All honest nodes do not receive ECHO messages after round $t-i$.

In this case, for any $i \in [t]$, we can show that all honest nodes do not receive ECHO messages after round i. That is the only way an honest node can receive a message m is in round $t+1$ transmitted by a byzantine node. However, this is a contradiction as this event cannot occur. A byzantine node holding a message at round $t+1$ means that through all rounds the message was transmitted between byzantine nodes, *only*. Knowing that if a node does not receive $t+1$ ACK it will halt, this means that the best strategy is to transfer the message to only one byzantine node at a time. This means that there is a need to $t+1$ byzantine nodes in the network which contradicts our assumption.

□

Lemma 5.9.3. Agreement: If an honest node accepts m, then all honest nodes eventually accept m.

Proof. If the sender of m is honest, then all honest nodes accept m according to Lemma 5.9.2. If the sender is byzantine, then all honest nodes either accept m or $\perp$ according to Lemma 5.9.2. Therefore, if an honest node accepts m, all honest nodes accept m, no matter the sender is honest or byzantine. □

Lemma 5.9.4. Integrity: For any message m, every honest node accepts m at most once, if m was broadcast by the sender.

Proof. According to Algorithm 7, every honest node only accepts m once, while receiving $t + 1$ ECHO messages. If $m \neq \perp$, all honest nodes accept the message broadcasted by the sender, no matter if the sender is honest or byzantine, Lemma 5.9.2. □

Lemma 5.9.5. Early Stopping: Every honest node will terminate at round $\min\{f + 2, t + 2\}$.

Proof. According to Algorithm 7, if the initiator is honest, then all honest nodes accept m from id_{sint} after two rounds. If it is a byzantine intiatior and f nodes violate the protocol (e.g., receiving less than $t + 1$ acknowledgement responses after sending a message), any of these f nodes can exist in the network for at most f rounds. After f rounds, if the message m is sent from any of the f nodes to the other nodes, then the other nodes will follow the protocol and multicast m to all honest nodes. After two rounds, all honest nodes will agree on the same value m. Otherwise, all honest nodes will wait until the end of round $t + 2$ and accept the default value $\perp$. Therefore, all honest nodes will terminate at round min f + 2, t + 2. Lemma 5.9.2, 5.9.3 and 5.9.4 also hold in the early-stopping case. □

Lemma 5.9.6. Termination: Every honest node eventually accepts m or $\perp$.

Proof. According to Algorithm 7, if an honest node receives $t + 1$ ECHO or INIT messages during $\min\{f + 2, t + 2\}$ rounds, it will accept m immediately; otherwise, it will accept $\perp$ at the end of round $t + 2$. □

Lemma 5.9.7. Efficiency: For any sender, the communication complexity is $O(N^2)$ for one instance of ERB.

Proof. For ERB, every node only broadcasts to all nodes once when receiving INIT or ECHO for the first time, thus every node sends N messages. To reply

requests from other nodes with ACK messages, every node sends at most another N messages. There are N nodes in the network, so the communication complexity for one run of ERB is at most $2N^2$ or $O(N^2)$ in total. □

This concludes our proof. □

5.9.2 P2P Sanitization & Analysis

In ERB, we introduce the concept of *network sanitization* or *faulty node elimination* captured by the *halt-on divergence (P4)* property, or the Halt function for short. This process has an important impact on the P2P topology as whenever a byzantine node misbehaves, the enclave of the node will deterministically stop the node. Thus, the byzantine node gets ejected from the network. This byzantine node cannot generate any new message as its enclave halts. We say that this *sanitizes* the P2P topology.

A byzantine OS gets churned out if it deviates from the sequential execution of ERB. Since it cannot infer the content of each message due to our blinded channel, one of the possible strategy is to behave maliciously uniformly at random. We present in the following our analysis that details the sanitization impact on ERB in this particular scenario, and shows that after a polynomial number of instances, the expected round complexity of the protocol becomes constant[13]. First, we give a characterization of the pace of sanitization considering that for every instance, a byzantine node can behave malicious with a probability that can be independently tuned. We also consider the effect of a new node joining the network before the start of every instance of the protocol to replace eventually an eliminated node.

Theorem 5.9.8. *Let F_r denotes the random variable counting the number of byzantine nodes after r instances of Algorithm 7, then*

$$\Pr[F_r \geq 1] \leq e^{-\lambda},$$

[13] We believe that the network sanitization asymptotic improvement can apply independently of the malicious nodes' strategy

where $\lambda = \frac{rp}{2} - \ln(t)$*,* t *the upper bound of byzantine nodes in the P2P network and* p *the fraction of activated byzantine nodes at any instance.*

Proof. It is easy to see that the number of byzantine nodes in the P2P network at the $(i+1)$th instance of Algorithm 7 equals: $F_{i+1} = F_i - R_i + A_i$, where R_i represents the number of byzantine nodes that have arbitrarily misbehaved and therefore are eliminated from the network, and A_i represents the number of new peers that have joined the P2P network. We can then write: $R_i = \sum_{j=1}^{F_i} X_j^{(1)}$, and $A_i = \sum_{j=1}^{F_i} X_j^{(2)}$, where $X_j^{(1)} \sim \mathcal{B}_p$ and $X_j^{(2)} \sim \mathcal{B}_{\frac{1}{2}}^p$ is a conditional Bernoulli random variables such that , $X_j^{(2)} = 1$ if $X_j^{(1)} = 1$ for $j \in [F_i]$.

$X_j^{(1)}$ is a Bernoulli random variable with parameter p that captures the fact that a node can misbehave in a particular instance of Algorithm 7 with probability p, while the second random variable $X_j^{(2)}$ captures the fact that whenever a node is eliminated from the network, it can be replaced with either a honest or malicious node. This is in phase with our assumption that we can handle a honest majority at the beginning.

Note that both R_i and A_i are both a *random sum* of random variables. As the number of failures at some iteration can be considered independent of the sum of failures occurred throughout all iterations of Algorithm 7, we can consider that both $X_j^{(1)}$ and $X_j^{(2)}$ are independent of F_i.

Based on Wald's equation, we have $E[F_{i+1}] = (1 - \frac{p}{2}) \cdot E[F_i]$. By induction we can show that $E[F_{i+1}] = (1 - \frac{p}{2})^{i+1} \cdot E[F_0]$, where $E[F_0] = E[t] = t$, the initial number of byzantine nodes in the network.

Based on Markov inequality, we show that

$$\Pr[F_r \geq 1] \quad \leq \quad t(1 - \frac{p}{2})^r \leq e^{-\frac{rp}{2} + \ln(t)}.$$

Setting $\frac{rp}{2} - \ln(t) = \lambda$ concludes the proof.

□

For example, for $\lambda = 30$ and $t = \frac{N}{2} - 1$ for $N = 2^{10}$ $p = 2^{-5}$, the number of rounds before the P2P gets sanitized with high probability equals to $r \approx 2500$.

We are now interested in computing the expected number of rounds in average of Algorithm 7 while taking into consideration our sanitization protocol. Theorem 5.9.8 shows that the P2P can get sanitized w.h.p. after a particular number of rounds, however, throughout the different instances, the number of byzantine nodes decreases as well, which suggests that the round complexity can get better. We will show in the following theorem that Algorithm 7 has a constant round complexity in average after a polynomial number of instances.

Theorem 5.9.9. *Algorithm 7 has a constant round complexity in average for a number of instances* $r = \mathsf{poly}(N)$ *w.h.p.*

In this theorem, we consider the same setting of Theorem 5.9.8 where the number of byzantine nodes at the $(i+1)$th instance equals $F_{i+1} = F_i - R_i + A_i$, where $R_i = \sum_{j=1}^{F_i} X_j^{(1)}$, and $A_i = \sum_{j=1}^{F_i} X_j^{(2)}$, $X_j^{(1)} \sim \mathcal{B}_p$ and $X_j^{(2)} \sim \mathcal{B}_{\frac{1}{2}}^p$ is a conditional Bernoulli random variables such that , $X_j^{(2)} = 1$ if $X_j^{(1)} = 1$ for $j \in [F_i]$.

We have shown that in this case the expected value of $E[F_i] = (1 - \frac{p}{2})^i \cdot t$.

To compute the expected number of rounds per instance, we need to count first the total number of possible byzantine nodes to ever occur after r instances, T_r. This equals $T_r = \sum_{i=1}^{r} \sum_{j=1}^{F_i} (X_i^{(1)} + X_i^{(2)})$, Moreover, we also define the average number of rounds per instance as a random variable , R, equal to $R = 2 \cdot \frac{(r - T_r)}{r} + t \cdot \frac{T_r}{r}$, where during $r - T_r$ rounds the protocol is optimal and equals 2, and in T_r rounds the protocol has a worst-case round complexity and equals to f.

We then have that, leveraging Wald's equation, $E[T_r] = \frac{3t}{2} \cdot (1 - (1 - \frac{p}{2})^{r+1})$. Then, $E[R] - 2 \sim \frac{3t^2}{2r} \cdot (1 - e^{\frac{-pr}{2}})$. By Markov, we have $\Pr(R \geq 3) \leq \frac{3t^2}{2r} \cdot (1 - e^{\frac{-pr}{2}})$.

That is, if $r = \mathsf{poly}(N)$ and $p = O(\frac{1}{t})$, then $\Pr(R \geq 3) \in O(\frac{1}{\mathsf{poly}(N)})$.

5.9.3 Unoptimized ERNG Analysis

In this section, similar to Section 5.7, we denote by the ROD mode, a mode where peers in a network $\mathcal{P}$ can only replay, omit and delay messages.

Theorem 5.9.10. *If $\mathcal{P}$ operates in the* ROD *mode, then the bias of G $\beta(G) = 1$.*

Proof. Note that while G can be modeled as a multi-variate function, it does not capture the sequencing of inputs. For our proof to go through, we need to first show that the sequencing of ERNG is guaranteed and a node can only participate with its input if it starts synchronously with all nodes. For this, we have the following two cases:

- *early start*: if a byzantine node transmits its INIT at $\mathsf{rnd} = 1$, then based on Lemma 5.9.2, the node outputs (either m or $\perp$) will be considered as an input for G,
- *late start*: if a byzantine holds the INIT message until seeing the output, then its input will not be added to SS_{final} as the message will be considered delayed. The output of G in this case equals $\perp$

Note that for both cases, the nodes have to start the protocol at $\mathsf{rnd} = 1$ if they want to participate with their inputs in the final output. Moreover, based on the Blinded channel, we know that nodes can only obtain the final output of G while not viewing any internal state of G, which enforce the *blind-box computation (P3)* property. That is, it is valid to consider G as a multi-variate function that is fed all inputs at once. Let us denote by X the random variable that captures the output of G such that $X = X_1 \oplus \cdots X_N$, where X_i's are random variables that capture the input provided by every node in $\mathcal{P}$, for all $i \in [N]$. As $\mathcal{P}$ operates in the ROD mode, Lemma 5.9.2 demonstrates that all honest nodes receive the same set SS_{final} at the end of the protocol. We then can rewrite X such that $X = \bigoplus_{i=1}^{\kappa} X_i \oplus \bigoplus_{i=\kappa+1}^{N} X_i$, where $\kappa = |SS_{\mathsf{final}}|$. In the following, we need to show that $E_G[S] = E[S] = \frac{|S|}{2^k}$, for all $S \subseteq \{0,1\}^k$. Note that $E_G[S] = \Pr[X \in S]$, and therefore it is sufficient to compute $\Pr[X \in S]$.

$$\Pr[X \in S] \quad = \quad \Pr[\bigcup_{x \in S} (X = x)] = \sum_{x \in S} \Pr[X = x]$$

The second equality follows from the fact that all events are disjoint. Now for a given $x \in S$, $\Pr[X = x]$ equals:

$$\begin{aligned}
&= \Pr[\bigoplus_{i=1}^{\kappa} X_i \oplus \bigoplus_{i=\kappa+1}^{N} X_i = x] \\
&= \sum_{x_N \in \{0,1\}^k} (\Pr[\bigoplus_{i=1}^{\kappa} X_i \oplus \bigoplus_{i=\kappa+1}^{N-1} X_i = x \oplus x_N | X_N = x_N] \\
&\cdot \Pr[X_n = x_N]) \\
&= \sum_{x_N \in \{0,1\}^k \setminus \{0\}} (\Pr[\bigoplus_{i=1}^{\kappa} X_i \oplus \bigoplus_{i=\kappa+1}^{N-1} X_i = x \oplus x_N | X_N = x_N] \\
&\cdot \Pr[X_n = x_N]) + \Pr[\bigoplus_{i=1}^{\kappa} X_i \oplus \bigoplus_{i=\kappa+1}^{N-1} X_i = x | X_N = 0] \cdot \Pr[X_n = 0] \\
&= \Pr[\bigoplus_{i=1}^{\kappa} X_i \oplus \bigoplus_{i=\kappa+1}^{N-1} X_i = x | X_N = 0] \\
&= \Pr[\bigoplus_{i=1}^{\kappa} X_i = x | X_{\kappa+1} = 0, \cdots, X_N = 0] \\
&= \sum_{x_\kappa \in \{0,1\}^k} \Pr[\bigoplus_{i=1}^{\kappa} X_i = x \oplus x_\kappa | X_\kappa = x_\kappa, X_{\kappa+1} = 0, \cdots, X_N = 0] \\
&\cdot \Pr[X_\kappa = x_\kappa] \\
&= \frac{1}{2^s} \sum_{x_\kappa \in \{0,1\}^k} \Pr[\bigoplus_{i=1}^{\kappa} X_i = x \oplus x_\kappa | X_\kappa = x_\kappa, X_{\kappa+1} = 0, \cdots, X_N = 0] \\
&= \frac{1}{2^{s \cdot (k-1)}} \sum_{x_2, \cdots, x_\kappa \in \{0,1\}^k} \Pr[X_1 = x \oplus \bigoplus_{i=2}^{\kappa} x_i | \\
&\quad X_2 = x_2, \cdots, X_\kappa = x_\kappa, X_{\kappa+1} = 0, \cdots, X_N = 0] \\
&= \frac{1}{2^{s \cdot k}} |\{x_2, \cdots, x_\kappa \in \{0,1\}^k\}| = \frac{1}{2^s}
\end{aligned}$$

Thus, $\Pr[X \in S] = \frac{|S|}{2^s}$. This concludes our proof.

□

5.9.4 Optimized ERNG

Lemma 5.9.1. *If up to $t = \frac{N}{3}$ nodes are byzantine, then with at least $1 - \mathsf{negl}(\gamma)$ probability, the representative cluster has more than γ honest nodes, and less than γ byzantine nodes.*

Proof. In ERNG at round 1, every node picks uniformly at random a value

from $\{0, \cdots, \frac{N}{2\gamma} - 1\}$. That is, every node has a probability equal to $q = \frac{2\gamma}{N}$ to be chosen as a representative. Let H_i and B_i be two random variable that equal 1 if the i^{th} honest and byzantine node is chosen respectively, otherwise they equal zero. Let us denote by $\mathsf{H} = \sum_{i=1}^{2t} H_i$ and $\mathsf{B} = \sum_{i=1}^{t} B_i$ the number of selected honest and byzantine nodes in the cluster. Then both H and B are distributed following a binomial distribution with a number of trials equal to $2t$ and t, respectively. We have $E[\mathsf{H}] = \sum_{i=1}^{2t} E[H_i] = 2t \cdot \frac{2\gamma}{N} = \frac{4t \cdot \gamma}{N}$. Similarly, $E[Y] = \frac{2t \cdot \gamma}{N}$. Based on two variations of Chernoff bound, considering $t = \frac{N}{3}$, we obtain that

$$\Pr[\mathsf{H} > (1 - \delta_1)\frac{4\gamma}{3}] \geq 1 - e^{-\frac{2\delta_1^2 \cdot \gamma}{3}},$$

similarly, $\Pr[\mathsf{B} < (1 + \delta_2)\frac{2\gamma}{3}] \geq 1 - e^{-\frac{2\delta_2^2 \cdot \gamma}{9}}$, where $\delta_1, \delta_2 < 1$. For a choice of $\delta_1 = \frac{1}{4}$ and $\delta_2 = \frac{1}{3}$, we obtain,

$$\Pr[\mathsf{H} > \gamma] \geq 1 - e^{-\frac{\gamma}{24}},$$

and,

$$\Pr[\mathsf{B} < \gamma] \geq 1 - e^{-\frac{\gamma}{41}}.$$

□

Lemma 5.9.2. *If $\gamma' = \sqrt{\gamma}$, then the probability that $\Omega(\sqrt{\gamma})$ honest nodes are selected to be in the second representative cluster is at least $1 - \mathsf{negl}(\gamma)$.*

Proof. Based on Algorithm 9, every node in the cluster has a probability of $\frac{1}{\gamma'}$ to be chosen. Let us denote by X_i the random variable equal to one if the node is selected. We then denote by, $\mathsf{H}' = \sum_{i=1}^{\mathsf{H}} X_i$, the random variable that counts the number of honest node in the second cluster. Based on Wald's equation, we obtain $E[\mathsf{H}'] = \frac{E[\mathsf{H}]}{\gamma'} = \frac{4\gamma}{3\gamma'}$. Then, based on Chernoff bound, we obtain for $\delta < 1$,

$$\Pr[\mathsf{H}' > (1 - \delta) \cdot \frac{4\gamma}{3\gamma'}] \geq 1 - e^{-\frac{4\delta^2 \cdot \gamma}{3\gamma'}}$$

if we set $\delta = 1 - \frac{1}{\gamma'}$ and $\gamma' = \sqrt{\gamma}$, then we obtain

$$\Pr[\mathsf{H}' > \frac{4\sqrt{\gamma}}{3}] \geq 1 - e^{-\sqrt{\gamma}}.$$

This ends out proof. □

Note that we can obtain better bounds if we consider computing the pmf of H' as it follows a binomial distribution with a binomial number of trials

Corollary 5.9.1. *If $\gamma' = \sqrt{\gamma}$, then the size of the first and second representative clusters is in $O(\gamma)$ and $O(\sqrt{\gamma})$ w.h.p*

The proof of the corollary directly follows from Lemma 5.9.2.

Theorem 5.9.11. Agreement: *All honest nodes eventually agree on the same common set SS_{final} in* ERNG.

Proof. In round 1, $|SS_{\mathsf{chosen}}|$ nodes are uniformly at random selected to be part of the representative cluster. Based on Lemma 5.9.1, we have shown that the cluster contains *strictly* more than γ honest nodes, and *strictly* less than γ byzantine nodes w.h.p. when $t < \frac{N}{3}$ That is, we have created a new smaller P2P network SS_{chosen} in which the honest nodes represent the majority. In the cluster, all honest nodes know each other, but byzantine nodes may deliberately not contact honest nodes on purpose. In this case, the cluster will be more robust with less byzantine nodes. Thus, all the results introduced for ERB will hold for this cluster of nodes.

From round 2 to round $\gamma + 3$, the second cluster has more than $\sqrt{\gamma}$ honest nodes w.h.p. according to Lemma 5.9.2. For each instance of ERB— whether initiated by an honest or byzantine node, the honest representative nodes will agree on a same message according to Lemma 5.9.3. Since there is at least one honest sender, all honest nodes will accept the honest sender's message for its run of ERB based on Lemma 5.9.2. After around $O(\sqrt{\gamma})$ runs, all honest nodes will agree on the same set of random numbers. Since the number of

honest representative nodes is larger than γ and all of them will multicast FINAL messages for the same set of messages in round $\gamma + 4$, then all honest nodes will receive adequate FINAL messages to accept the common set SS_{final}. □

In ERNG, since the message m_i is a random number generated by the SGX and proposed by the peer p_i, then eventually every honest node accepts the same set SS_{final} of random numbers according to Theorem 5.9.11. By performing exclusive disjunction (or XOR) of all the random numbers in SS_{final}, every honest node can obtain a common random number r. We demonstrate next that the random number r is unbiased against byzantine nodes.

Theorem 5.9.12. Unbiasedness: *The output of the* ERNG *protocol is an unbiased random number.*

sketch. Given Theorem 5.9.11, we know that all honest nodes agree on the same set SS_{M}. On the other hand, leveraging $\mathsf{Peer}^{\mathsf{Ch}}_{\mathsf{sgx}}$ Peer channel, we know that all random numbers in the ERNG protocol are generated within the SGX enclave and never tempered with as the network is in the ROD model, based on Corollary 5.7.2. Finally, it is sufficient to show that if all random numbers generated in SGX are truly random then the output of ERNG is an unbiased random number, which holds given SGX primitive generates unbiased random number against the operating system according to Theorem 5.9.10. □

5.10 Discussions

5.10.1 Are Assumptions Reasonable?

S1: Network Size. We start with a fixed size network $\mathcal{P}$ with N peers. We assume that there exists information that publicly identifies every node in $\mathcal{P}$, this can be for example a node IP address. This assumption is reasonable under some common conditions. For example, for banking systems, all involved machines should be registered and publicly available. Fortunately, we can weaken

such as assumption and we can extend our setting to work within a variable size network based on the following technique: whenever a node wants to join $\mathcal{P}$, the joining node contacts another neighbor node and communicates both its sequence number and identifier. The contacted node will use ERB to reliably broadcast the pair to all peers in $\mathcal{P}$ and then send the joining peer a message containing all existing identifiers of $\mathcal{P}$. We can leverage the same technique in a recursive way to even start with a one node network $\mathcal{P}$. Note that in this case the identifier need not to be publicly known.

S2: Synchronous Start. Before initiating the ERB primitive, we assume that any honest node in $\mathcal{P}$ can be triggered at the same reference time. This reference time can be provided in different ways such as periodic execution from a fixed reference date, or simply by starting at a time posted in public servers. Once synchronized, every node uses the trusted elapsed time from SGX to maintain a relative time from the reference time. This therefore will maintain an internal clock within every node's enclave. As the enclaves in all honest nodes will have the (nearly) same internal clock, all nodes will start the next instance of the protocol at the same time. If any byzantine node deviates by omitting or delaying the oracle message, its elapsed time will be different from the one honest nodes have. Consequently, all the byzantine node messages will be delayed as they are going to have a different round number.

S3: Round time 2Δ seconds. The round time (2Δ seconds) is adequately determined to allow any honest round trip message to complete within 2Δ seconds. The round increments are managed using the trusted elapsed time, which implies that even if the OS is byzantine, the round number will be always incremented inside the enclave every 2Δ seconds. We also emphasize that the time interval between any two internal clocks for honest nodes is negligible compared to 2Δ seconds. As ERB does not use any underlying heavy cryptographic primitive, we assert that any sent message will be received in the same round. The 2Δ seconds is mostly dedicated for network latency reasons.

S4: Number of byzantine nodes less than $\frac{N}{2}$. To join a network $\mathcal{P}$, an adversary is required to control machines with SGX-enabled CPUs, in which the number of possible launched enclaves is bounded [110]. To control $\frac{N}{2}$, the adversary needs to control a number of SGX machines. Meanwhile, we can also employ existing sybil defenses in our network to control the number of byzantine nodes, e.g., defenses using computation puzzles or proof of work [83, 167]. The details of deploying these sybil defenses are beyond the scope of this chapter.

S5: Connected Peers. For simplicity of design and to follow the standard model used in previous works, we assume that all the peers in the network are connected to each other. However this assumption can be relaxed such that the network is a sparse but expander or random graph. This will guarantee that there is a path in between any two honest nodes. Thus, the direct point-to-point broadcast in our protocol can be replaced with a flooding algorithm to broadcast messages.

5.10.2 Applications

Both ERB and ERNG primitives can be used as building blocks to solve a wide range of problems in distributed systems. In the following, we review some of the most prominent applications.

Random Beacons. A random beacon protocol [202] offers a way to generate uniformly random strings that are unknown to the nodes before their generation. Random beacons have been extensively studied as they have numerous applications in cryptography and information security, such as secure contract signing protocols [131,202], voting schemes [184], zero-knowledge protocols [68,143], and cryptocurrency protocols [172]. Building random beacons is a difficult task. Practical solutions usually leverage a trusted third party [31,54], or utilize public data available on the Internet such as financial data [106]. However, the data from these services has to be trusted and certified, which unfortunately repre-

sents a strong assumption in practice. Recently, researchers have also proposed several protocols to generate random beacons by using Bitcoin as a source of publicly-verifiable randomness [75, 82]. However, the adversary can bias the beacon by introducing a new monetary cost. With ERNG, the underlying system can easily generate a common unbiased random number in the network.

Random Walks. In order to build a more robust P2P topology, random walk is an essential primitive to distribute nodes uniformly in the network to maintain an expander topology. Guerraoui *et al.* [144] build a virtual overlay on top of the physical nodes, in order to maintain a robust P2P topology. Each virtual node represents a cluster that consists of a set of physical nodes such that at least $\frac{2}{3}$ of the nodes are honest. This guarantees that decisions or agreements of the cluster hold on the behalf of the entire physical nodes of the network. Ensuring that the virtual nodes are honest will guarantee the correctness of the random walks against byzantine nodes. However, this is not sufficient and in order to determine the next hop in the random walk, an unbiased random number is required. With ERNG, we present an efficient solution for this issue in such a way that physical nodes in the cluster can generate a common unbiased random number to designate the next hop, and therefore maintain a robust topology.

Shared Key Generation. By performing ERNG, every honest node will share a common unbiased random number that can be used as a key, salt or initialization vector for symmetric cryptography. ERNG can also be used as a building block for distributed key generation (DKG) where the peers want to compute a shared public and private key. DKG has several applications and in particular in threshold cryptography, we refer the reader to the works by Gennaro *et al.* [140, 141].

Random Load Balancing. Random load balancing is generally performed by a centralized server to distribute tasks to slave servers [111, 203]. A centralized server is often considered as a single point of failure, which is usually the primary target of attackers. Once the centralized server is compromised, the whole

load balancing system fail as well. With ERNG, we distribute the decision generation process to a cluster of nodes instead of a centralized server. When a new request or a task comes to any node, the cluster of nodes evaluate ERNG to generate an unbiased common random number and send the decision to the target slave server. Once the slave server receives adequate confirmations from (say) half of the nodes, it can take upon the task and evaluate it. This way, even if half of the nodes are either compromised/failed, the load balancing system can still work correctly. Note that the nodes can a-priori pre-process many random numbers to speed-up the process. The random numbers can be generated and stored in the hard drive using *sealing* technique enabled by the SGX.

5.11 Related Work

Reliable broadcast has been extensively investigated in various adversarial models. In our work, we show how Intel SGX improves the efficiency of existing protocols in these adversarial models, renewing interest in studying these protocols with SGX-based implementations.

Reliable Broadcast: Reliable broadcast has been extensively studied since the 1980s, and is closely related to the problem of byzantine agreement (BA). Several excellent surveys on the problem are available [160, 220]. Byzantine agreement can also achieve reliable broadcast [88, 92, 94, 155, 176, 185, 201, 220]. We have discussed the related approaches in Chapter 2.

Researchers have proposed byzantine fault-tolerant algorithms using trusted services, such as by using trusted computing primitives, primarily focusing on making PBFT more efficient [73, 98, 105, 107, 108, 166, 170, 221]. These works have observed similar relation to crash-fault-tolerant protocols, as we have. For example, Chun *et al.* introduce an attested append-only memory (A2M) to remove the ability of adversarial replicas to equivocate without detection, which helps to increase the resilience from $\frac{N}{3}$ to $\frac{N}{2}$ [105]. Liu *et al.* further pro-

pose FastBFT using message aggregation to reduce message complexity from $O(N^2)$ to $O(N)$ [170]. However, these works have concentrated on handling asynchronous protocols with weak time assumptions like PBFT. In this paper, in contrast to previous approaches, we work on the round-based synchronous model. Our work extends these ideas to detecting and remediating failures of synchronous network assumptions (e.g. our lockstep execution and halt-on-divergence). Additionally, we investigate the use of our blind-box execution primitive in our new distributed RNG protocol which is bias-resistant, and more efficient using secure sampling for cluster creation. We leave the extension of applying our properties and primitives to asynchronous protocols for future work.

As it is difficult to tolerate $N/2$ byzantine nodes with $O(N^2)$ communication complexity, researchers studied a weaker model, dubbed omission model. We study one of these models called general-omission. Perry *et al.* and Parvedy *et al.* propose protocols that can achieve $O(N^3)$ communication complexity, tolerate $\frac{N}{2}$ faulty nodes, and terminate in $\min\{f+2, t+1\}$ rounds [193, 198]. Chandra *et al.* present a protocol achieving $O(N^2)$ communication complexity, but terminating with $2t+1$ rounds if optimizations apply [99]. In this work, we use SGX features to reduce the byzantine model to the general omission model, and further propose ERB to achieve $\min\{f+2, t+2\}$ round complexity and $O(N^2)$ communication complexity.

Distributed RNG: Generating common coins in a distributed manner for randomized BA in asynchronous networks can also be used for generating unbiased random numbers [77, 93, 201]. However, these protocols either require a trusted dealer to set up the initial states of different nodes or pre-distribute data to the nodes in the network. Other works employing asynchronous verifiable secret sharing (AVSS) protocols do not have the trusted dealer, but can probabilistically execute with errors [74, 88, 96, 217]. Most of these works employ some cryptographic primitives that, in most case, can be considered heavy-weight and

performance unfriendly. Awerbuch *et al.* propose a solution that tolerates up to $\frac{N}{6}$ byzantine nodes, with $O(N)$ round complexity and $O(N^3)$ communication complexity [67] to generate a random number with a constant bias. Other works, such as Andrychowicz *et al.*'s one, generate a common random number based on proof of work [65] with $O(N^4)$ communication complexity, but the output can eventually be biased. Moreover, the large communication cost for most of these approaches prevents scalability to a large number of nodes. We present more efficient (with $O(N \log N)$ communication complexity) and unbiased RNG generation for the synchronous network case.

5.12 Summary

The recent availability of Intel SGX in commodity laptops and servers provides a promising research direction for advancing the area of P2P systems. Our main observation is that leveraging SGX features can restrict a byzantine model to a general-omission model in synchronous systems. We highlight that using SGX we can improve the efficiency of P2P protocols such as reliable broadcast and unbiased random number generator in synchronous settings.

Chapter 6

Conclusion

While peer-to-peer techniques are introduced to the web infrastructure, the privacy issues in P2P systems are also brought to this new P2P web. In this thesis, we first investigate the inference attacks in peer-assisted CDNs on top of web overlays, which can infer users' online activities such as browsing history. Moreover, we propose an anonymous peer-assisted CDN (APAC), which employs onion-routing techniques enabling users to be unidentifiable within a large set of other users. We also design a region-based circuit selection algorithm for APAC to reduce performance overhead and provide options for developers to balance the tradeoff between privacy and performance. Second, to thwart attacks via long-term global traffic analysis, we propose an oblivious peer-to-peer content sharing system (OBLIVP2P) using distributed oblivious RAM and other cryptographic primitives to hide users' online activities (or access patterns). Lastly, the core utilities and security / privacy guarantees of P2P protocols are based on the robustness of these protocols. With the new hardware primitive called Intel SGX, we ensure the robustness of fundamental P2P protocols in the malicious (or byzantine) setting, which is not considered in APAC and OBLIVP2P. By using SGX features, we further improve the efficiency of P2P protocols, which can be widely used for other P2P operations such as random walks, shared key generation and load balancer algorithms. We release

the source code of our implementations for the three works, which are available online [4, 32, 34].

Future Work. OBLIVP2P is the first attempt to adapt ORAM technique to P2P systems to conceal data access patterns. However, OBLIVP2P has several shortcomings, e.g., having a centralized trusted tracker and low throughput as well as only working in the honest-but-curious setting. We have demonstrated that SGX can be employed to propose robust P2P primitives. For future work, we are planning to leverage SGX features to re-design OBLIVP2P to achieve decentralization of the trusted tracker and better throughput with low latency against byzantine adversaries. In the meantime, the majority of current privacy-preserving P2P systems do not have proper incentive mechanisms to engage users to join these system. To address this challenge, we can use secure execution and remote attestation provided by SGX-enabled CPUs to fairly quantify the transferred data to reward the nodes involved in the fetch / eviction processes of OBLIVP2P. Furthermore, we can integrate our SGX-based solutions into WebRTC-enabled web browsers like Chromium as generic primitives for new proposed protocols.

Bibliography

[1] Akamai. http://www.akamai.com/.

[2] Akamai netsession interface. http://www.akamai.com/client.

[3] AMD secure technology. http://www.amd.com/en-us/innovations/software-technologies/security.

[4] An Anonymous Peer-assisted CDN (APAC). https://github.com/jiayaoqijia/APAC-Code.

[5] BemTV. http://bem.tv/.

[6] Bitcoin. https://bitcoin.org/en/.

[7] Bittorrent. http://www.bittorrent.com/.

[8] BitTorrent over Tor isn't a good idea. https://blog.torproject.org/blog/bittorrent-over-tor-isnt-good-idea.

[9] Boost c++ library. http://www.boost.org/.

[10] CLOC. http://cloc.sourceforge.net/.

[11] CloudFlare. https://www.cloudflare.com/.

[12] CrashPlan. http://www.code42.com/crashplan/.

[13] Crypto-js: JavaScript implementations of standard and secure cryptographic algorithms. https://code.google.com/p/crypto-js/.

[14] The "data" URL scheme. http://tools.ietf.org/html/rfc2397.

[15] Datagram transport layer security. https://tools.ietf.org/html/rfc4347.

[16] Deterlab. https://www.isi.deterlab.net/index.php3.

[17] A fast json parser/generator for c++ with both sax/dom style api. http://rapidjson.org/.

[18] Freenet: The free network. https://freenetproject.org.

[19] Geolocation api specification. http://www.w3.org/TR/geolocation-API/.

[20] Gnutella. https://web.archive.org/web/20080525005017/http://www.gnutella.com/.

[21] Htop - an interactive process viewer for unix. http://hisham.hm/htop/. Accessed: 2016.

[22] I2P: The invisible Internet project. https://geti2p.net/en/.

[23] Iftop: display bandwidth usage on an interface. http://www.ex-parrot.com/pdw/iftop/. Accessed: 2016.

[24] Indexed database api. http://www.w3.org/TR/IndexedDB/.

[25] Intel software guard extensions. https://software.intel.com/en-us/sgx.

[26] Intel software guard extensions for linux* os. https://01.org/intel-softwareguard-extensions.

[27] Is WebRTC ready yet? http://iswebrtcreadyyet.com.

[28] Maelstrom. http://blog.bittorrent.com/tag/maelstrom.

[29] Mist browser. https://github.com/ethereum/mist.

[30] Network address translation. http://en.wikipedia.org/wiki/Network_address_translation.

[31] NIST randomness beacon. https://www.nist.gov/programs-projects/nist-randomness-beacon.

[32] Oblivious Peer-to-Peer Protocol. https://github.com/jiayaoqijia/OblivP2P-Code.

[33] Octoshape. http://www.octoshape.com/.

[34] P2P using SGX. https://bitbucket.org/P2PUsingSGX/p2pusingsgx.

[35] P2PSP. http://www.p2psp.org/webrtc-streaming/.

[36] Palo Alto networks application usage & threat report. http://researchcenter.paloaltonetworks.com/app-usage-risk-report-visualization/.

[37] PeerCDN. https://peercdn.com/.

[38] The peerjs library. http://peerjs.com/.

[39] Protocol buffers - google's data interchange format. https://github.com/google/protobuf.

[40] Python cryptography toolkit. https://pypi.python.org/pypi/pycrypto.

[41] Python ECC. https://github.com/johndoe31415/joeecc.

[42] Scaneye's global bittorrent monitor. http://www.cogipas.com/anonymous-torrenting/torrent-monitoring/.

[43] Session traversal utilities for nat (stun). https://tools.ietf.org/html/rfc5389.

[44] Storj.io. http://storj.io/.

[45] Swarmify. http://www.swarmify.com/.

[46] Symform. http://www.symform.com/.

[47] Tor. https://www.torproject.org/.

[48] Tor: Hidden service protocol. https://www.torproject.org/docs/hidden-services.html.en.

[49] Tor suffers traffic confirmation attack. http://www.techtimes.com/articles/11711/20140802/tor-suffers-traffic-confirmation\-attacks-say-goodbye-to-anonymity-on-the-web.htm.

[50] Total transfer size & total requests. http://httparchive.org/trends.php.

[51] Traffic confirmation attack. https://blog.torproject.org/blog/tor-security-advisory-relay-early-traffic-confirmation-attack.

[52] The transport layer security (TLS) protocol. https://tools.ietf.org/html/rfc5246.

[53] Tribler. http://www.tribler.org/.

[54] True random number service. https://www.random.org/.

[55] Trusted platfrom module (TPM). http://www.trustedcomputinggroup.org/work-groups/trusted-platform-module/.

[56] UTorrent & BitTorrent surge to 150 million monthly users. https://torrentfreak.com/bittorrent-surges-to-150-million-monthly-users-120109/.

[57] Velocix. http://www.velocix.com/.

[58] Wbittorrentireshark. https://www.wireshark.org/.

[59] WebRTC. http://www.webrtc.org/.

[60] The Weibull distribution. http://reliawiki.org/index.php/The_Weibull_Distribution.

[61] Ittai Abraham and Danny Dolev. Byzantine agreement with optimal early stopping, optimal resilience and polynomial complexity. In *Proceedings of the 47th Annual ACM on Symposium on Theory of Computing*, 2015.

[62] Paarijaat Aditya, Mingchen Zhao, Yin Lin, Andreas Haeberlen, Peter Druschel, Bruce M Maggs, and Bill Wishon. Reliable client accounting for P2P-infrastructure hybrids. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation*, 2012.

[63] Dakshi Agrawal and Dogan Kesdogan. Measuring anonymity: The disclosure attack. *IEEE Security & privacy*, 2003.

[64] Masoud Akhoondi, Curtis Yu, and Harsha V Madhyastha. LASTor: A low-latency AS-aware Tor client. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, 2012.

[65] Marcin Andrychowicz and Stefan Dziembowski. Distributed cryptography based on the proofs of work. *IACR Cryptology ePrint Archive*, 2014.

[66] Robert Annessi and Martin Schmiedecker. NavigaTor: Finding faster paths to anonymity. In *Proceedings of the 1st IEEE European Symposium on Security and Privacy*, 2016.

[67] Baruch Awerbuch and Christian Scheideler. Robust random number generation for peer-to-peer systems. In *Proceedings of the 20th International Conference on Principles of Distributed Systems*, 2006.

[68] László Babai. Trading group theory for randomness. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, 1985.

[69] Michael Backes, Aniket Kate, Sebastian Meiser, and Esfandiar Mohammadi. (nothing else) mator(s): Monitoring the anonymity of tor's path selection. In *Proceedings of the 21st ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[70] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-resource routing attacks against tor. In *Proceedings of the 6th ACM Workshop on Privacy in the Electronic Society*, 2007.

[71] Kevin Bauer, Damon McCoy, Dirk Grunwald, and Douglas Sicker. Bitblender: Light-weight anonymity for bittorrent. In *Proceedings of the 2008 Workshop on Applications of Private and Anonymous Communications*, 2008.

[72] Kevin Bauer, Damon McCoy, Dirk Grunwald, and Douglas Sicker. Bitstalker: Accurately and efficiently monitoring bittorrent traffic. In *Proceedings of the 1st IEEE International Workshop on Information Forensics and Security*, 2009.

[73] Johannes Behl, Tobias Distler, and Rüdiger Kapitza. Hybrids on steroids: Sgx-based high performance bft. In *Proceedings of the 20th European Conference on Computer Systems*, 2017.

[74] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience. In *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing*, 1994.

[75] Iddo Bentov, Ariel Gabizon, and David Zuckerman. Bitcoin beacon. *arXiv preprint arXiv:1605.04559*, 2016.

[76] Piotr Berman and Juan A Garay. Cloture votes: n/4-resilient distributed consensus int+ 1 rounds. *Theory of Computing Systems*, 1993.

[77] Piotr Berman and Juan A Garay. Randomized distributed agreement revisited. In *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing*, 1993.

[78] Piotr Berman, Juan A Garay, and Kenneth J Perry. Optimal early stopping in distributed consensus. In *Proceedings of the 6th International Workshop on Distributed Algorithms*, 1992.

[79] Ashwin R Bharambe, Cormac Herley, and Venkata N Padmanabhan. Analyzing and improving bittorrent performance. Technical report, Technical Report MSR-TR-2005-03, Microsoft Research, 2005.

[80] Stevens Le Blond, Pere Manils, Chaabane Abdelberi, Mohamed Ali Dali Kaafar, Claude Castelluccia, Arnaud Legout, and Walid Dabbous. One bad apple spoils the bunch: exploiting P2P applications to trace and profile tor users. *arXiv preprint arXiv:1103.1518*, 2011.

[81] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In *Proceedings of the 33rd Annual International Cryptology Conference*, 2013.

[82] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On Bitcoin as a public randomness source. *IACR Cryptology ePrint Archive*, 2015.

[83] Nikita Borisov. Computational puzzles as sybil defenses. In *Proceedings of the 6th IEEE International Conference on Peer-to-Peer Computing*, 2006.

[84] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of service or denial of security? In *Proceedings of the 14th ACM SIGSAC Conference on Computer and Communications Security*, 2007.

[85] Philippe Boucher, Adam Shostack, and Ian Goldberg. Freedom systems 2.0 architecture. *Zero Knowledge Systems, Inc*, 2000.

[86] Justin Boyan. The anonymizer: Protecting user privacy on the web. *Computer-Mediated Communication Magazine*, 1997.

[87] Elette Boyle, Kai-Min Chung, and Rafael Pass. Oblivious parallel RAM and applications. In *Proceedings of the 13th Theory of Cryptography Conference*, 2016.

[88] Gabriel Bracha. An asynchronous [(n-1)/3]-resilient consensus protocol. In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*, 1984.

[89] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 1987.

[90] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 1985.

[91] Florian Burgstaller, Andreas Derler, Stefan Kern, Gabriel Schanner, and Andreas Reiter. Anonymous communication in the browser via onion-routing. In *Proceedings of the 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, 2015.

[92] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Proceedings of the 21st Annual International Cryptology Conference*, 2001.

[93] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 2005.

[94] Christian Cachin and Jonathan A Poritz. Secure intrusion-tolerant replication on the internet. In *Proceedings of 32nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2002.

[95] Christian Cachin and Stefano Tessaro. Asynchronous verifiable information dispersal. In *Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems*, 2005.

[96] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, 1993.

[97] Frank Cangialosi, Dave Levin, and Neil Spring. Ting: Measuring and exploiting latencies between all tor nodes. In *Proceedings of the 15th ACM SIGCOMM Conference on Internet Measurement*, 2015.

[98] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, 1999.

[99] Tushar Deepak Chandra and Sam Toueg. Time and message efficient reliable broadcasts. In *Proceedings of the 4th International Workshop on Distributed Algorithms*, 1990.

[100] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 1981.

[101] Binyi Chen, Huijia Lin, and Stefano Tessaro. Oblivious parallel RAM: improved efficiency and generic constructions. In *Proceedings of the 13th Theory of Cryptography Conference*, 2016.

[102] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *Proceedings of the 31st IEEE Symposium on Security and Privacy*, 2010.

[103] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, 1995.

[104] Tom Chothia, Marco Cova, Chris Novakovic, and Camilo González Toro. The unbearable lightness of monitoring: Direct monitoring in bittorrent. In *Proceedings of the 8th International Conference on Security and Privacy in Communication Networks*, 2013.

[105] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. Attested append-only memory: Making adversaries stick to their word. In *ACM SIGOPS Operating Systems Review*, 2007.

[106] Jeremy Clark and Urs Hengartner. On the use of financial data as a random beacon. In *Proceedings of the 2010 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections*, 2010.

[107] Miguel Correia, Nuno Ferreira Neves, and Paulo Verissimo. How to tolerate half less one byzantine nodes in practical distributed systems. In *Proceedings of the 23rd International Symposium on Reliable Distributed Systems*, 2004.

[108] Miguel Correia, Paulo Veríssimo, and Nuno Ferreira Neves. The design of a COTS real-time distributed security kernel. In *Proceedings of the 4th European Dependable Computing Conference*, 2002.

[109] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *Proceedings of the 36th IEEE Symposium on Security and Privacy*, 2015.

[110] Victor Costan and Srinivas Devadas. Intel sgx explained. In *IACR Cryptology ePrint Archive*, 2016.

[111] George Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 1989.

[112] Dana Dachman-Soled, Chang Liu, Charalampos Papamanthou, Elaine Shi, and Uzi Vishkin. Oblivious network RAM and leveraging parallelism to achieve obliviousness. In *Proceedings of the 21st Annual International Conference on the Theory and Application of Cryptology and Information Security*, 2015.

[113] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. 2002.

[114] George Danezis. Statistical disclosure attacks. In *Security and Privacy in the Age of Uncertainty*. 2003.

[115] George Danezis and Claudia Diaz. A survey of anonymous communication channels. Technical report, Technical Report MSR-TR-2008-35, Microsoft Research, 2008.

[116] George Danezis, Claudia Diaz, and Carmela Troncoso. Two-sided statistical disclosure attack. In *Proceedings of the 7th International Symposium on Privacy Enhancing Technologies*, 2007.

[117] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type iii anonymous remailer protocol. In *Proceedings of the 24th IEEE Symposium on Security and Privacy*, 2003.

[118] George Danezis, Chris Lesniewski-Laas, M Frans Kaashoek, and Ross Anderson. Sybil-resistant DHT routing. In *Proceedings of the 10th European Symposium on Research in Computer Security*, 2005.

[119] George Danezis and Andrei Serjantov. Statistical disclosure or intersection attacks on anonymity systems. In *Proceedings of the 7th International Conference on Information Hiding*, 2005.

[120] George Danezis and Carmela Troncoso. Vida: How to use bayesian inference to de-anonymize persistent communications. In *Proceedings of the 9th International Symposium on Privacy Enhancing Technologies*, 2009.

[121] Srinivas Devadas, Marten van Dijk, Christopher W Fletcher, Ling Ren, Elaine Shi, and Daniel Wichs. Onion oram: A constant bandwidth blowup oblivious ram. In *Proceedings of the 13th Theory of Cryptography Conference*, 2016.

[122] Claudia Diaz, Stefaan Seys, Joris Claessens, and Bart Preneel. Towards measuring anonymity. In *Proceedings of the 3rd International Symposium on Privacy Enhancing Technologies*, 2003.

[123] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX Securitedings of the 13th USENIX Security Symposium*, 2004.

[124] Danny Dolev, Ruediger Reischuk, and H Raymond Strong. Early stopping in byzantine agreement. *Journal of the ACM*, 1990.

[125] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 1983.

[126] John R Douceur. The sybil attack. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, 2002.

[127] Kevin P Dyer, Scott E Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, 2012.

[128] Matthew Edman and Paul Syverson. AS-awareness in Tor path selection. In *Proceedings of the 16th ACM SIGSAC Conference on Computer and Communications Security*, 2009.

[129] Karim El Defrawy, Minas Gjoka, and Athina Markopoulou. BotTorrent: Misusing BitTorrent to launch DDoS attacks. *Proceedings of the 3rd Workshop on Steps to Reducing Unwanted Traffic on the Internet*, 2007.

[130] Manal El Dick, Esther Pacitti, and Bettina Kemme. Flower-cdn: a hybrid p2p overlay for efficient query processing in cdn. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, 2009.

[131] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 1985.

[132] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 1997.

[133] Matthias Fitzi and Juan A Garay. Efficient player-optimal protocols for strong and differential consensus. In *Proceedings of the 22nd Annual Symposium on Principles of Distributed Computing*, 2003.

[134] Michael J Freedman. Experiences with CoralCDN: A five-year operational view. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation*, 2010.

[135] Michael J. Freedman, Eric Freudenthal, and David Mazières. Democratizing content publication with coral. In *Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation*, 2004.

[136] Michael J Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM SIGSAC Conference on Computer and Communications Security*, 2002.

[137] Yan Gao, Leiwen Deng, Aleksandar Kuzmanovic, and Yan Chen. Internet cache pollution attacks and countermeasures. In *Proceedings of the 14th IEEE International Conference on Network Protocols*, 2006.

[138] Juan A Garay and Yoram Moses. Fully polynomial byzantine agreement in t+ 1 rounds. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, 1993.

[139] Juan A Garay and Yoram Moses. Fully polynomial byzantine agreement for processors in rounds. *SIAM Journal on Computing*, 1998.

[140] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, 1999.

[141] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 2007.

[142] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM*, 1996.

[143] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, 1986.

[144] Rachid Guerraoui, Florian Huc, and Anne-Marie Kermarrec. Highly dynamic distributed computing with byzantine failures. In *Proceedings of the 32nd ACM symposium on Principles of Distributed Computing*, 2013.

[145] Vassos Hadzilacos and Sam Toueg. Fault-tolerant broadcasts and related problems. In *Distributed systems (2nd Ed.)*, 1993.

[146] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *Proceedings of the 24th USENIX Security Symposium*, 2015.

[147] Cheng Huang, Angela Wang, Jin Li, and Keith W Ross. Understanding hybrid cdn-p2p: why limelight needs its own red swoosh. In *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2008.

[148] Tomas Isdal, Michael Piatek, Arvind Krishnamurthy, and Thomas Anderson. Privacy-preserving p2p data sharing with oneswarm. In *Proceedings of the 2010 ACM SIGCOMM Computer Communication Review*, 2010.

[149] Sitaram Iyer, Antony Rowstron, and Peter Druschel. Squirrel: A decentralized peer-to-peer web cache. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing*, 2002.

[150] Yaoqi Jia, Guangdong Bai, Prateek Saxena, and Zhenkai Liang. Anonymity in peer-assisted CDNs: Inference attacks and mitigation. In *Proceedings of the 16th International Symposium on Privacy Enhancing Technologies*, 2016.

[151] Yaoqi Jia, Yue Chen, Xinshu Dong, Prateek Saxena, Jian Mao, and Zhenkai Liang. Man-in-the-browser-cache: Persisting HTTPS attacks via browser cache poisoning. *Computers & Security*, 2015.

[152] Yaoqi Jia, Xinshu Dong, Zhenkai Liang, and Prateek Saxena. I know where you've been: Geo-inference attacks via the browser cache. *IEEE Internet Computing*, 2014.

[153] Yaoqi Jia, Tarik Moataz, Shruti Tople, and Prateek Saxena. OblivP2P: An oblivious peer-to-peer content sharing system. 2016.

[154] Yaoqi Jia, Shruti Tople, Tarik Moataz, Deli Gong, Prateek Saxena, and Zhenkai Liang. Robust synchronous P2P primitives using SGX enclaves. In *IACR Cryptology ePrint Archive*, 2017.

[155] Bruce M Kapron, David Kempe, Valerie King, Jared Saia, and Vishal Sanwalani. Fast asynchronous byzantine agreement and leader election with full information. *ACM Transactions on Algorithms*, 2010.

[156] Thomas Karagiannis, Pablo Rodriguez, and Konstantina Papagiannaki. Should internet service providers fear peer-assisted content distribution? In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, 2005.

[157] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In *Proceedings of the 26th Annual International Cryptology Conference*, 2006.

[158] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. 2014.

[159] Dogan Kesdogan and Lexi Pimenidis. The hitting set attack on anonymity protocols. In *Proceedings of the 6th International Conference on Information Hiding*, 2004.

[160] Valerie King and Jared Saia. Scalable byzantine computation. *ACM SIGACT News*, 2010.

[161] Jie Kong, Wandong Cai, and Lei Wang. The evaluation of index poisoning in bittorrent. In *Proceedings of the 2nd International Conference on Communication Software and Networks*, 2010.

[162] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 1982.

[163] Stevens Le Blond, Adina Uritesc, Cédric Gilbert, Zheng Leong Chua, Prateek Saxena, and Engin Kirda. A look at targeted attacks through the lense of an ngo. In *Proceedings of the 23rd USENIX Security Symposium*, 2014.

[164] Stevens Le Blond, Chao Zhang, Arnaud Legout, Keith Ross, and Walid Dabbous. I know where you are and what you are sharing: exploiting p2p communications to invade users' privacy. In *Proceedings of the 11th ACM SIGCOMM Conference on Internet Measurement*, 2011.

[165] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. *arXiv preprint arXiv:1611.06952*, 2016.

[166] Dave Levin, John R Douceur, Jacob R Lorch, and Thomas Moscibroda. TrInc: Small trusted hardware for large distributed systems. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, 2009.

[167] Frank Li, Prateek Mittal, Matthew Caesar, and Nikita Borisov. Sybilcontrol: practical sybil defense with computational puzzles. In *Proceedings of the 7th ACM Workshop on Scalable Trusted Computing*, 2012.

[168] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. On the composition of authenticated byzantine agreement. *Journal of the ACM*, 2006.

[169] Chao Liu, Ryen W White, and Susan Dumais. Understanding web browsing behaviors through weibull analysis of dwell time. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2010.

[170] Jian Liu, Wenting Li, Ghassan O Karame, and N Asokan. Scalable byzantine consensus via hardware-assisted secret sharing. *arXiv preprint arXiv:1612.04997*, 2016.

[171] Steve Lu and Rafail Ostrovsky. Distributed oblivious RAM for secure two-party computation. In *Proceedings of the 10th Theory of Cryptography Conference*, 2013.

[172] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[173] Nayantara Mallesh and Matthew Wright. The reverse statistical disclosure attack. In *Proceedings of the 12th International Conference on Information Hiding*, 2010.

[174] Pere Manils, Chaabane Abdelberri, Stevens Le Blond, Mohamed Ali Kaafar, Claude Castelluccia, Arnaud Legout, and Walid Dabbous. Compromising tor anonymity exploiting p2p information leakage. *arXiv preprint arXiv:1004.1461*, 2010.

[175] Nick Mathewson and Roger Dingledine. Practical traffic analysis: Extending and resisting statistical disclosure. In *Proceedings of the 5th International Symposium on Privacy Enhancing Technologies*, 2005.

[176] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[177] Ming-Wei-Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. T-SGX: Eradicating controlled-channel attacks against enclave programs. In *Proceedings of the 24th Annual Network and Distributed System Security Symposium*, 2017.

[178] Alan Mislove, Gaurav Oberoi, Ansley Post, Charles Reis, Peter Druschel, and Dan S Wallach. AP3: Cooperative, decentralized anonymous com-

munication. In *Proceedings of the 11th ACM SIGOPS European Workshop*, 2004.

[179] Prateek Mittal and Nikita Borisov. Shadowwalker: peer-to-peer anonymous communication using redundant structured topologies. In *Proceedings of the 16th ACM SIGSAC Conference on Computer and Communications Security*, 2009.

[180] Prateek Mittal, Femi Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. PIR-Tor: scalable anonymous communication using private information retrieval. In *Proceedings of the 20th USENIX Security Symposium*, 2011.

[181] Prateek Mittal, Matthew Wright, and Nikita Borisov. Pisces: Anonymous communication using social networks. In *Proceedings of the 19th Annual Network and Distributed System Security Symposium*, 2012.

[182] T. Moataz, T. Mayberry, and E.-O. Blass. Constant communication ORAM with small blocksize. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[183] Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster protocol-version 2. 2003.

[184] Tal Moran and Moni Naor. Split-ballot voting: everlasting privacy with distributed trust. *ACM Transactions on Information and System Security*, 2010.

[185] Achour Mostefaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous byzantine consensus with $t < \frac{n}{3}$ and $o(n^2)$ messages. In *Proceedings of the 33rd ACM Symposium on Principles of Distributed Computing*, 2014.

[186] Gabi Nakibly, Jaime Schcolnik, and Yossi Rubin. Website-targeted false content injection by network operators. *arXiv preprint arXiv:1602.07128*, 2016.

[187] Arjun Nambiar and Matthew Wright. Salsa: a structured approach to large-scale anonymity. In *Proceedings of the 13th ACM SIGSAC Conference on Computer and Communications Security*, 2006.

[188] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, 1997.

[189] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *Proceedings of the 1st IEEE European Symposium on Security and Privacy*, 2016.

[190] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious multi-party machine learning on trusted processors. In *Proceedings of the 25th USENIX Security Symposium*, 2016.

[191] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th ACM Workshop on Privacy in the Electronic Society*, 2011.

[192] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the 34th IEEE Symposium on Security and Privacy*, 2013.

[193] Philippe Raïpin Parvédy and Michel Raynal. Optimal early stopping uniform consensus in synchronous systems with process omission failures.

In *Proceedings of the 16th ACM Symposium on Parallelism in Algorithms and Architectures*, 2004.

[194] Rafael Pass, Elaine Shi, and Florian Tramer. Formal abstractions for attested execution secure processors. *IACR Cryptology ePrint Archive*, 2016.

[195] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 1980.

[196] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Survey of network-based defense mechanisms countering the dos and ddos problems. *ACM Computing Surveys*, 2007.

[197] Fernando Pérez-González and Carmela Troncoso. Understanding statistical disclosure: A least squares approach. In *Proceedings of the 12th International Symposium on Privacy Enhancing Technologies*, 2012.

[198] Kenneth J Perry and Sam Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Transactions on Software Engineering*, 1986.

[199] Andreas Pfitzmann and Marit Hansen. Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management-a consolidated proposal for terminology. *Version v0*, 2008.

[200] Michael Piatek, Tadayoshi Kohno, and Arvind Krishnamurthy. Challenges and directions for monitoring P2P file sharing networks, or, why my printer received a DMCA takedown notice. In *Proceedings of the 3rd Conference on Hot Topics in Security*, 2008.

[201] Michael O Rabin. Randomized byzantine generals. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, 1983.

[202] Michael O Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 1983.

[203] Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, and Ion Stoica. Load balancing in structured p2p systems. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, 2003.

[204] Michael G Reed, Paul F Syverson, and David M Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 1998.

[205] Michael K Reiter and Aviel D Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1998.

[206] L. Ren, C.W. Fletcher, A. Kwon, E. Stefanov, E. Shi, M. van Dijk, and S. Devadas. Constants Count: Practical Improvements to Oblivious RAM . In *Proceedings of the 24th USENIX Security Symposium*, 2015.

[207] Marc Rennhard and Bernhard Plattner. Introducing MorphMix: peer-to-peer based anonymous Internet usage with collusion detection. In *Proceedings of the 1st ACM Workshop on Privacy in the Electronic Society*, 2002.

[208] Daniel S. Roche, Adam J. Aviv, and Seung Geol Choi. A practical oblivious map data structure with secure deletion and history independence. *IACR Cryptology ePrint Archive*, 2015.

[209] Vincent Scarlata, Brian Neil Levine, and Clay Shields. Responder anonymity and anonymous peer-to-peer file sharing. In *Proceedings of the 9th International Conference on Network Protocols*, 2001.

[210] E. Shi, T.-H.H. Chan, E. Stefanov, and M. Li. Oblivious RAM with $O(\log^3(N))$ Worst-Case Cost. In *Proceedings of the 17th International Conference on the Theory and Application of Cryptology and Information Security*, 2011.

[211] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. Preventing page faults from telling your secrets. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, 2016.

[212] Georgos Siganos, Josep M Pujol, and Pablo Rodriguez. Monitoring the bittorrent monitors: A bird's eye view. In *Proceedings of the 10th International Conference on Passive and Active Measurement*, 2009.

[213] E. Stefanov, M. van Dijk, E. Shi, C.W. Fletcher, L. Ren, X. Yu, and S. Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In *Proceedings of the 20th ACM SIGSAC Conference on Computer and Communications Security*, 2013.

[214] Emil Stefanov and Elaine Shi. Multi-cloud oblivious storage. In *Proceedings of the 20th ACM SIGSAC Conference on Computer and Communications Security*, 2013.

[215] Emil Stefanov and Elaine Shi. ObliviStore: High Performance Oblivious Distributed Cloud Data Store. In *Proceedings of the 20th Annual Network and Distributed System Security Symposium*, 2013.

[216] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, 2006.

[217] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. *IACR Cryptology ePrint Archive*, 2016.

[218] Jeff Terrace, Harold Laidlaw, Hao Eric Liu, Sean Stern, and Michael J Freedman. Bringing P2P to the Web: Security and Privacy in the Fireco-

ral Network. In *Proceedings of the 8th International Workshop on Peer-to-Peer Systems*, 2009.

[219] Carmela Troncoso, Benedikt Gierlichs, Bart Preneel, and Ingrid Verbauwhede. Perfect matching disclosure attacks. *Lecture Notes in Computer Science*, 2008.

[220] Vinod Vaikuntanathan. *Randomized algorithms for reliable broadcast*. PhD thesis, 2009.

[221] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Verissimo. Efficient byzantine fault-tolerance. *IEEE Transactions on Computers*, 2013.

[222] Long Vu, Indranil Gupta, Klara Nahrstedt, and Jin Liang. Understanding overlay characteristics of a large-scale peer-to-peer IPTV system. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 2010.

[223] Tao Wang, Kevin Bauer, Clara Forero, and Ian Goldberg. Congestion-aware path selection for tor. In *Proceedings of the 16th International Conference on Financial Cryptography and Data Security*. 2012.

[224] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *Proceedings of the 23rd USENIX Security Symposium*, 2014.

[225] Tao Wang and Ian Goldberg. Improved website fingerprinting on tor. In *Proceedings of the 12th ACM Workshop on Privacy in the Electronic Society*, 2013.

[226] Gilbert Wondracek, Thorsten Holz, Engin Kirda, and Christopher Kruegel. A practical attack to de-anonymize social network users. In *Proceedings of the 31st IEEE Symposium on Security and Privacy*, 2010.

[227] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Proceedings of the 36th IEEE Symposium on Security and Privacy*, 2015.

[228] Hao Yin, Xuening Liu, Tongyu Zhan, Vyas Sekar, Feng Qiu, Chuang Lin, Hui Zhang, and Bo Li. Design and deployment of a hybrid CDN-P2P system for live video streaming: experiences with LiveSky. In *Proceedings of the 17th ACM International Conference on Multimedia*, 2009.

[229] Haifeng Yu, Chenwei Shi, Michael Kaminsky, Phillip B Gibbons, and Feng Xiao. Dsybil: Optimal sybil-resistance for recommendation systems. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, 2009.

[230] Jia Zhang, Haixin Duan, Wu Liu, and Jianping Wu. Anonymity analysis of P2P anonymous communication systems. *Computer Communications*, 2011.

[231] Liang Zhang, Fangfei Zhou, Alan Mislove, and Ravi Sundaram. Maygh: Building a CDN from client web browsers. In *Proceedings of the 8th ACM European Conference on Computer Systems*, 2013.

[232] Mingchen Zhao, Paarijaat Aditya, Ang Chen, Yin Lin, Andreas Haeberlen, Peter Druschel, Bruce Maggs, Bill Wishon, and Miroslav Ponec. Peer-assisted content distribution in Akamai netsession. In *Proceedings of the 13th ACM SIGCOMM Conference on Internet Measurement*, 2013.