A Framework of Privacy Preserving Services for Distributed Online Social Networks

by

Sanaz Taheri Boshrooyeh

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Doctor of Philosophy

in

Computer Science and Engineering

October 8, 2019

A Framework of Privacy Preserving Services for Distributed Online Social Networks

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a doctoral dissertation by

Sanaz Taheri Boshrooyeh

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Assoc. Prof. Alptekin Küpçü (Advisor)

_____

Prof. Öznur Özkasap (Advisor)

_____

Asst. Prof. Didem Unat

_____

Asst. Prof. Selçuk Baktır

_____

Prof. Özgür Barış Akan

_____

Prof. Albert Levi

Date: _____

I dedicate this thesis to my family who showed me what it means to love, and to be loved.

ABSTRACT

Online Social Network (OSN) providers like Facebook and Twitter supply storage and computational resources for the users and enable them to share personal information and establish friendships. The collection of users data at a central OSN provider causes security issues due to which the distributed OSNs with federated server architecture is proposed. The central provider is replaced by multiple servers (running by distinct providers) which run OSN services collaboratively while no server is fully trusted. We utilize the federated server architecture and the trust distribution among the servers to propose a framework of efficient and privacy-preserving services for OSNs that would be otherwise impossible in the centralized versions.

The first module of the framework, Privado, is a privacy-preserving group-based advertising method through which OSN servers can find the advertiser's target customers using users' encrypted profiles. We propose the group-based advertising notion to protect user privacy, which is not possible in the personalized variant. We formally define and prove user privacy against an active malicious adversary controlling all but one server, all the advertisers, and a large fraction of the users. Privado also achieves advertising transparency; the procedure of identifying target customers is operated solely by the servers without getting users and advertisers involved.

The second module of the framework, Anonyma, copes with the inference attack in the invitation-only nature of OSNs' group formation. Anonyma is an invitation-based system where users prove to the administrator that they are invited by a certain number of group members without revealing their inviters. We formally define and prove the inviter anonymity and unforgeability of invitations against a malicious adversary. Also, Anonyma outperforms the state-of-the-art concerning the computational overhead. Besides, Anonyma is efficiently scalable in the sense that the administrator

can issue credentials to the newcomers, enabling them to act as an inviter, without re-keying the existing members. We also design AnonymaX, an anonymous cross-network invitation-based system where the invitations issued by the members of one social network can be used for the registration to another social network.

The third module of the framework, Integrita, is a collaborative data-sharing platform (e.g., Facebook groups) that preserves view consistency against corrupted servers i.e., servers cannot show divergence view of the shared data (e.g., posts of the group page) to the users (e.g., group members) without being detected. Unlike the state-of-the-art, Integrita enables detection of inconsistency neither by using storage inefficient data replication solution nor by requiring users to exchange their views out of the band. Every user, without relying on the presence of other users, can verify any server-side equivocation regarding her performed operation. We introduce and achieve a new level of view consistency called q-detectable consistency in which any inconsistency between users view cannot remain undetected for more than q posts.

For each module of this thesis, the running time and performance of our proposed approaches are examined through extensive simulations, and the results provided accordingly. Privado enables an advertising system with tweak-able parameters to adjust the advertising accuracy w.r.t. user privacy. Privado also excels the state-of-the-art by efficiently and simultaneously achieving advertising transparency and user privacy with the minimum communication complexity at the server-side. Anonyma supports provable invitation unforgeability and inviter anonymity by a computation complexity upper-bounded only by the number of required inviters while the computation complexity of the prior studies grows linearly with the system size. The data-sharing platform of Integrita advances the centralized and distributed counterparts by improving the view-consistency and storage overhead (by the factor of $\frac{1}{N}$ where N is the number of the servers), respectively. Nevertheless, concerning per server storage overhead and cross-server communication, Integrita's overhead is the minimum among all its counterparts.

ÖZETÇE

Facebook ve Twitter gibi Çevrimiçi Sosyal Ağların (ÇSA'lar) hizmet sağlayıcıları, kullanıcılara depolama ve hesaplama kaynakları sunar ve kişisel bilgilerini paylaşmalarını ve arkadaşlıklar kurmalarını sağlar. Merkezi bir ÇSA'da kullanıcı verilerinin toplanması güvenlik riskleriyle birlikte gelir. Bu sebeple federe sunucu mimarisine sahip dağıtık ÇSA'lar önerilmiştir. Merkezi bir sunucu yerine farklı sağlayıcılar tarafından çalıştırılan birden çok sunucu kullanılması önerilmiştir. Sunucuların hiçbiri tam olarak güvenilir değildir. Bu tezde, federe sunucu mimarisini ve sunucular arasındaki güven dağılımını kullanarak ve merkezi yapıda aksi takdirde imkansız olacak verimli ve gizliliği koruyan bir ÇSA hizmetleri çerçevesi öneriyoruz.

Çerçevenin ilk modülü Privado, ÇSA sunucuları tarafından, reklam verenlerin hedef müşterilerini ÇSA kullanıcılarının şifreli profillerini kullanarak bulabilecekleri, gizliliği koruyan bir grup tabanlı reklam yöntemidir. Kişiselleştirilmiş sistemlerde mümkün olmayan kullanıcı gizliliğini korumak için grup temelli reklam kavramını öneriyoruz. Tek bir sunucuyu, tüm reklam verenleri ve kullanıcıların büyük bir bölümünü kontrol eden aktif bir saldırgana karşı kullanıcı gizliliğini resmi olarak tanımlıyor ve kanıtlıyoruz. Tasarımımız aynı zamanda reklam şeffaflığını da sağlar; Hedef müşterileri belirleme prosedürü, kullanıcıları ve reklam verenleri dahil etmeden, sadece ÇSA sunucuları tarafından çalıştırılmaktadır.

Çerçevenin ikinci modülünde, Anonyma, ÇSA'larda davetiye ile grup katılımını ele alıyoruz. Anonyma ile kullanıcılar sistem yöneticisine, kim tarafından davet edildiklerini açıklamadan belirli sayıda davet aldıklarını kanıtlayabilmektedir. Kötü niyetli bir saldırgana karşı davetiyenin anonimliğini ve davetiyelerin orijinalliğini resmi olarak tanımlıyor ve kanıtlıyoruz. Ek olarak tasarımımız performans olarak en iyi rakiplerinden daha iyi sonuçlar göstermektedir. Bir kullanıcı sisteme katıldıktan sonra,

yöneticinin anında ve mevcut üyeleri yeniden anahtarlamadan, yeni gelen kişinin davetci olarak hareket edebilmesi için gerekli bilgileri vermesi anlamında verimli bir şekilde ölçeklenebilir. Ayrıca, bir sosyal ağın üyeleri tarafından verilen davetiyelerin başka bir sosyal ağa kayıt için kullanılabileceği davetiye tabanlı bir sistem olan AnonymaX çözümümüzü de tasarladık.

Çerçevenin üçüncü modülü Integrita, ele geçirilmiş sunucuların varlığında görünüm tutarlılığını koruyan ortak bir veri paylaşım platformudur (Facebook grupları gibi). Diğer bir deyişle, paylaşılan verinin (örneğin, grup sayfasının gönderileri) farklı kullanıcılara (örneğin, grup üyeleri) farklı gösterilemeyeceği garanti edilir. Mevcut çözümlerden farklı olarak, Integrita, yinelemeli depolama veya kullanıcıların bant dışı iletişimini gerektirmeden tutarsızlığın tespitini sağlar. Her kullanıcı, gerçekleştirdiği işlemle ilgili görünüm tutarlılığını doğrulamak için tek başına (diğer kullanıcıların varlığına güvenmeden) işlem yapabilir. Bu durum için q-saptanabilir tutarlılık adı verilen yeni bir görünüm tutarlılığı tanımı ve çözümü ortaya koyuyoruz.

Tezin her modülü için önerilen yaklaşımlarımızın çalışma süresi ve performansı kapsamlı simülasyonlar ve bunlara göre elde edilen sonuçlar üzerinden incelenmiştir. Privado, istenen reklamcılık doğruluğunu ve kullanıcı gizlilik seviyesini ayarlamak için parametrelere sahip bir reklam sistemini sunucu tarafında minimum iletişim karmaşıklığı ile reklam şeffaflığını ve kullanıcı gizliliğini aynı anda sağlayarak teknolojik üstünlük sunar. Anonyma, davetiye anonimliğini ve orijinalliğini kanıtlanabilir bir biçimde sağlarken yalnızca gerekli davetiye sayısına bağlı bir hesaplama karmaşıklığı gerektirir (önceki sistemlerde ise bu durum sistemdeki toplam kullanıcı sayısı ile doğru orantılıydı). Integrita veri paylaşım platformu, merkezileştirilmiş ve dağıtılmış emsallerine kıyasla görünüm tutarlılığını geliştirir ve ÇSA sunucularının depolama yükünü $\frac{1}{N}$ katsayısı ile hafifletir (N sunucu sayısıdır) ve sunucu başına depolama ve sunucular arası iletişim konularında bütün rakipleri arasında en düşük yükü gerektirir.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisors Assoc. Prof. Alptekin Küpçü and Prof. Öznur Özkasap for the continuous support of my Ph.D. study and related research, for their patience, motivation, and knowledge. Their guidance helped me in all the time of research and writing of this thesis.

I would like to express my appreciations to my thesis committee: Asst. Prof. Selçuk Baktır, Asst. Prof. Didem Unat, Prof. Özgür Barış Akan, and Prof. Albert Levi for devoting their valuable time and consideration in evaluating my thesis.

My sincere thanks also goes to my colleagues in DISNET and Crypto laboratories for the stimulating discussions, and for all the memorable times we have had in the last five years. Also, my appreciation goes out to all the incredible researchers from the Crypto community which I met from all over the world. Those who expanded my vision towards my research goals and inspired me both professionally and personally.

I would also like to express my gratitude to all the staff and employees of Koç University, especially the Graduate School of Sciences and Engineering, for their hard-working, effort, and dedication to the students' matters.

I am grateful for being a part of office ENG 142 and to get to know amazing people with whom I have shared thousands of priceless memories. I appreciate their support and kindness which made me feel like at home miles away from my own country.

A special thanks to my family, my dear mother who supported me unconditionally and filled my heart with her care and love. And my sister with whom I had those little deep, happy, and free moments like nothing else exists on this planet. I am also grateful to my grandmother and grandfather for all their love and encouragement.

I would like to say a heartfelt thank you to my loving, supportive, encouraging, and patient husband (and colleague), Yahya, who has been by my side throughout

this PhD, living every single minute of it, who believed in me and supported me in the worst and the best moments of my Ph.D., who gave me the courage when I needed the most and strengthened me in the face of challenges.

Finally, I acknowledge the support of the Royal Society of UK Newton Advanced Fellowship NA140464 and European Union COST Action IC1306.

TABLE OF CONTENTS

List of Tables **xvi**

List of Figures **xvii**

Nomenclature **xix**

Chapter 1: Introduction **1**

1.1 Scope of the Thesis 2

1.2 Original Contributions 6

1.3 Organization 10

Notes to Chapter 1 11

Chapter 2: Literature Review **12**

2.1 Introduction 12

2.2 OSN Architectures 14

2.2.1 Centralized Online Social Networks 14

2.2.2 Distributed Online Social Networks (DOSNs) 15

2.3 Data privacy 17

2.3.1 Information Substitution 17

2.3.2 Symmetric Key Encryption 17

2.3.3 Public Key Encryption 18

2.3.4 Attribute Based Encryption (ABE) 18

2.3.5 Identity Based Broadcast Encryption (IBBE) 19

2.3.6 Hybrid Encryption 20

2.4 Data Integrity 21

2.4.1 Integrity of the data owner and the data content 22
2.4.2 Historical integrity 23
2.4.3 Integrity of the data relations 23
2.5 Secure Social Search 24
2.5.1 Content privacy 25
2.5.2 Privacy of searcher 25
2.5.3 Privacy of the searched data owner 26
2.5.4 Trusted search result 26
2.6 Conclusion and Open problems 27

Chapter 3: Privado: Privacy-Preserving Group-based Advertising using Multiple Independent Social Network Providers 30

3.1 Introduction 30
3.2 Related Works 34
3.3 System Model 37
3.3.1 Model 37
3.3.2 Security Goal 38
3.3.3 Adversarial Model 39
3.4 Definitions and Preliminaries 39
3.4.1 Notations 39
3.4.2 Definitions 41
3.4.3 Preliminaries 42
3.5 Privado 50
3.5.1 Design Challenges 50
3.5.2 Construction Overview 52
3.5.3 Full Construction 53
Initialization: 54
User Registration: 55

Advertisement Registration: 57
Advertising 59
Notes to Chapter 3 66

Chapter 4: Privado: Complexity, Performance and Security 68
4.1 Complexity and Performance 68
4.1.1 Complexity 68
4.1.2 Concrete Performance 69
4.1.3 Advertisement Accuracy Metrics 73
4.1.4 Advertisement Accuracy Results 75
4.2 Security 76
4.2.1 Security Definition 77
4.2.2 Formal Security Proof 79
Notes to Chapter 4 88

Chapter 5: Anonyma: Anonymous Invitation-Only Registration in Malicious Adversarial Model 89
5.1 Introduction 89
5.2 Related Works 94
5.2.1 Electronic Voting (e-voting) 94
5.2.2 (t,N) Threshold Ring Signature 96
5.3 System Model 97
5.3.1 Model 97
5.3.2 Security Goal and Adversarial Model 98
5.4 Notations, Definitions, and Preliminaries 99
5.4.1 Notations 99
5.4.2 Definitions 100
5.4.3 Preliminaries 103
5.5 Construction 106

5.5.1 Construction Overview: . . . 106
5.5.2 Full Construction: . . . 109
SetUp: . . . 109
Token Generation: . . . 110
Invitation Generation: . . . 111
Invitation Collection: . . . 114
5.6 AnonymaX: Anonymous Cross-Network Invitation-Based System . . . 117
Notes to Chapter 5 . . . 120

Chapter 6: Anonyma: Performance and Security 121
6.1 Performance . . . 121
6.1.1 Running Time . . . 121
6.1.2 Communication Complexity . . . 122
6.1.3 Storage . . . 123
6.2 Security . . . 123
6.2.1 Proof of Invitation Correctness . . . 123
Soundness: . . . 123
Special honest verifier zero knowledge: . . . 124
Zero-knowledge POIC (ZKPOIC): . . . 124
6.2.2 Inviter Anonymity . . . 125
Security Definition: . . . 125
6.2.3 Invitation Unforgeability . . . 131
Security Definition: . . . 131
6.2.4 Security of AnonymaX . . . 138

Chapter 7: Integrita: Protecting View-Consistency in Online Social Network with Federated Servers 148
7.1 Introduction . . . 148
7.2 Related Works . . . 152

7.3 System Model 154
7.3.1 Model 154
7.3.2 Security Goal 155
7.3.3 Adversarial Model 155
7.4 Definitions and Preliminaries 155
7.4.1 Notations 155
7.4.2 Definitions 157
7.4.3 Preliminaries 157
7.5 Integrita System Design 158
7.5.1 Shared *object* representation 158
7.5.2 Distributed storage of the shared *object* 162
7.5.3 Construction 168
Notes to Chapter 7 182

Chapter 8: Integrita: Complexity, Performance and Security 183
8.1 Complexity and Performance 183
8.1.1 Storage Overhead 183
8.1.2 Round Complexity and Communication Complexity 184
8.2 Security 186
8.2.1 q-Detectable Consistency and Inconsistency Interval 186
Notes to Chapter 8 194

Chapter 9: Conclusion 195
9.1 Remarks 195
9.2 Future Directions 197

Bibliography 198

LIST OF TABLES

2.1 Classification of security aspects and solutions in OSNs 13

3.1 Notations used in Privado 41

4.1 Computation Complexity. 69

5.1 Notations used in Anonyma 100

6.1 Running time of the server and the inviter. 122

7.1 Notations used in Integrita 157

8.1 Storage overhead of Integrita vs related work. 184

8.2 Communication complexity. 186

LIST OF FIGURES

1.1 An overview of the thesis modules: Privado, Anonyma, and Integrita. 7

3.1 Privado System Overview. 38
3.2 The ideal functionality F_{THRES}. 46
3.3 The ideal functionality F^R_{POPR}. 47
3.4 The ideal functionality F^R_{POCM}. 48
3.5 The ideal functionality F^R_{VS}. 50
3.6 An instance of User Registration protocol (UReg). 55
3.7 Advertiser registration protocol (AdReg). 59
3.8 Advertisement protocol (Ad). 60

4.1 Running time of servers in advertising protocol. 71
4.2 Privado vs PPAD in advertising protocol running time. 74
4.3 *Target accuracy* and *Non-Target accuracy*. 87

5.1 The sample workflow of an invitation-only registration system. 90
5.2 Anonyma overview. 90
5.3 Parties' interaction in Anonyma. 107
5.4 Σ protocol of Proof of Invitation Correctness. 113

6.1 The invitee's running time. 122

7.1 A history tree constructed for the *object* with 4 posts. 159
7.2 The tree of Figure 7.1 after the insertion of $post_5$. 160
7.3 The membership proof of $post_3$ for version 4^{th} of the shared *object*. . . 161
7.4 A sample incremental proof. 161

7.5 Insertion path of $post_1$, $post_2$, $post_3$, and $post_4$. 162
7.6 Insertion path of $post_1$, $post_2$, $post_3$, and $post_4$ under the new addressing. 164
7.7 The labeling of nodes using the labeling function L. 165
7.8 Version 7^{th} of the shared *object*. 169
7.9 Membership proof sample. 169
7.10 Update Status protocol. 174
7.11 *Read* protocol. 174
7.12 $Write$ protocol. 176
7.13 The Audit Threshold for various number of servers. 179
7.14 The Transition point for various number of servers. 179

NOMENCLATURE

Chapter 1

INTRODUCTION

One of the most popular systems for sharing information is **online social networks (OSNs)** such as Facebook and Twitter that gained huge public attention in recent years. In contrast to the World Wide Web which is a content-based system, OSNs are *user based* systems where participating users join the network to link with others and share information. Statistics assert that in an OSN like Facebook there exist 2 billion monthly active users[1]. Also, around 80% of users of the Internet visit one of the OSN sites everyday [1].

Well-known OSNs such as Facebook and Twitter have a **centralized architecture** where a single service provider manages the whole system [2] i.e., supplies storage and computational resources for the users and offer various services through which users are able to share their personal information with one another and make new friendships. Having a centralized architecture and data aggregation improves usability. For example, having one entity for storing the users' data makes the searching process easy. Also, by having a central service provider, the system is more flexible in terms of updating and extending the network and changing the underlying architecture [2].

The centralized architecture also comes with its disadvantages, especially concerning user privacy. The prime observation is that the central service provider has direct access to the users' private data, knows the social graph that represents interconnections among OSN users, and observes user preferences and behavior within the OSN. Having a single large database containing all the information about users and their connection information makes OSNs a wealthy source of information for the hackers

and hence are frequently targeted by the attackers. Moreover, the collection of users information is also prone to the service provider misbehavior [2].

The security problems posed by the centralized nature of OSNs have motivated the research community to develop alternative OSN architecture that is distributed OSN (DOSN). The prime objective of DOSNs is to remove the central service provider and distribute its role (e.g., storage of users' data and serving social networking functionalities) among multiple distinct entities. This architecture comes into two major categories i.e., peer-to-peer (p2p) network architecture [1], and federated servers [3]. In the p2p OSN (e.g., Friendica[2] and RetroShare[3]), every OSN user (referred by peer) contributes a portion of her storage and computational power to the OSN. The OSN services shall run by users collaboration relying on their shared resources. Users act like a server to store and serve other users data, and also act as a client while requesting data from other peers. The main obstacle of p2p OSN is that individual peers are responsible for the availability of the data assigned to them which requires peers to stay online to fulfill this responsibility.

The federated server architecture [4, 5, 6] copes with the data availability issue in the p2p counterparts. Under this architecture, multiple servers, running by independent authorities, are designated for the storage of users data and delivering OSN services. As such, utilizing federated servers enables full data availability without requiring users being online all the time. Additionally, this design comes with a significant security benefit over the centralized OSNs where none of the servers get to have a complete global view and control of the private data stored in the system, which is not the case in centralized OSNs.

1.1 Scope of the Thesis

In the current thesis, the proposed privacy-preserving services are based on the federated server architecture of OSN. The benefits of using this architecture are two-fold; due to the storage and computational power of servers, users will enjoy the service quality and data availability as in a centralized system while the trust distribution

among the servers will enable devising privacy-preserving services that would be otherwise impossible in the centralized counterparts. In practice, the federated server architecture can be realized by having an OSN whose servers are provided by the ISPs of multiple distinct countries, or by using distinct cloud computing platforms.

As the first contribution of this thesis, we conduct a comprehensive study on the privacy techniques in secure centralized and distributed OSNs [3] (Chapter 2). In this study, we also hand researchers with the security holes and vulnerabilities that are not yet investigated. At a high level, security concerns of OSN users fall into one of the following categories, preserving data privacy, assuring data integrity and protecting the confidentiality of social connections. Under the data privacy concern, a user wants to ensure her data is only accessible to the users of OSN that she authorizes. The data integrity assures the data owner that her published content is immune against internal or external tamper, forgery, and censorship. The privacy of user relation is a new paradigm appears in the social networking services where the system is user-oriented rather than content-based (like world-wide-web). Users' relations are a source of important information as some network inferences can be done by the user's friends' information and interests.

In this thesis, we put three open problems (regarding data privacy, integrity, and social connection) under the spotlight which we explain next.

Advertising: The very first problem that we underline, regards OSNs commercial models where OSN providers monetize users' personal information by selling them to the untrustworthy advertisers [7, 8]. This clearly is against user privacy desire. To cope with this issue, existing studies, focusing only on the user privacy, propose secure designs of OSNs in which users encrypt their data before sharing with OSN provider [9, 10, 11, 12, 13, 14]. Although data encryption would mitigate the user data privacy issue, it immediately disables the advertising service and cuts the significant financial benefit of advertising (the advertising revenue for Facebook in 2018 was reported as 30.83 billion dollars[4]). Due to this monetizing inability, providers would have no convincing commercial model to establish secure OSNs [15]. According to

our literature review, no study has been conducted (in centralized and distributed architectures) to hand OSN provider with a financial model that meets user privacy as well. To fill this gap, we propose a privacy-preserving advertising module by which secure OSNs (whose design is based on data encryption) can efficiently perform advertising and find target customers using the privacy-protected profiles of users.

Invitation-only registration policy: The second module is focused on the security concerns raised by the invitation-only nature of group formation in OSNs e.g., Facebook page. In such groups, special content or service is to be accessible only to a trusted or privileged set of users. To comply with the trust assumption, the group administrator accepts the new member if the newcomer gets invitations from a certain number of existing group members where each invitation confirms some level of trust to the invitee. This type of registration is called invitation-only registration policy. Another closely relevant example is the trustee-based social authentication deployed by Facebook as a backup authentication method [16, 17]. A backup authentication method is used where the user fails to pass the primary authentication e.g., forgetting the password [16]. The account holder determines a set of trustees to the server in advance. When the user loses access to his account, the server sends recovery codes to the trustees (equivalent to inviters in invitation-only registration scenario). Upon collection of enough number (recovery threshold) of codes from the trustees and handing the codes to the server, the user regains access to his account.

In invitation-only registration systems, the group administrator needs to know who is invited by whom to authenticate and manage new registrations. In some other cases, like in Telegram [18], not only the administrator but also other members of the group are informed about the correspondence of a newcomer and his inviters. The group members who invite a user are mostly among the user's acquaintances (e.g., colleagues, home mates, family members, close friends) who have many common preferences with the user. Due to this reason, information like location, religious beliefs, sexual orientation, and political views can be inferred about a user by analyzing the common features among his inviters [19, 20]. Thus, the set of user's inviters is privacy-sensitive

information. Concerning this sensitivity, we dedicate the second module of this thesis to the proposal of an invitation-only registration method in which a user is able to authenticate herself to the group administrator needless to disclosing her set of inviters.

Enforcing view-consistency for shared data: OSNs enable various methods of data sharing like via users personal walls or social groups. Using the personal wall, a user may share her personal information (e.g., thoughts, images, and videos), with the social connections she authorizes on the OSN, i.e., the friends or followers. In addition to the user being able to continuously update its wall information, her friends or followers may also update her wall by adding post to it e.g., birthday messages, and commenting. The similar data-sharing paradigm appears in the context of social networking groups like Facebook groups where the members of the group can jointly update the content of the group page by inserting posts.

In the current designs of OSN with a central provider, users' read and write requests over the shared data (being a wall or a group page) is sent to the central OSN server who authorizes the request and acts accordingly. Users interaction with OSN provider is based on trust that is the server processes the requests honestly and according to the designated instructions. However, in the current practice of OSNs, rather than trust, no technique is deployed to enforce such trustworthy behavior of OSN server. A corrupted server may add arbitrary content to the shared data and make users accept them as authentic data or hide some posts from some users. As a historical example, in 2012, several bloggers claimed that Sina Weibo, a Chinese OSN, aimed to practice censorship by serving different views of the walls to their followers via hiding some of their posts [9]. Given such historical incidents, it is vital to tackle *view consistency* of the *object* with a practical solution rather than trusting the service provider. As the third module of this thesis, we spot the view consistency problem of shared data in OSNs and deliver a solution basing on the federated server architecture.

1.2 Original Contributions

The original contribution of this thesis is **A Framework of Privacy Preserving Services for Distributed Online Social Networks**, which embodies the following list of contributions which are also depicted in Figure 1.1.

Throughout the description, we refer to an *honest but curious entity (HbC)* (or passive adversary, interchangeably) as a party who follows all the system protocols but may try to gather more information from the execution of the protocol. An HbC adversarial model assumes that the protocol is run by HbC entities. On the other hand, a *malicious entity* (or active adversary, interchangeably) represents a party which has no obligation to follow the protocol specifications and acts as he wishes. A malicious adversarial model captures a system running by malicious parties. Throughout this thesis, we assume the static corruption model for the adversary i.e., the set of parties controlled by the adversary is fixed. As such, given a party is honest or corrupted, she always acts accordingly.

Privado: Privacy-Preserving Group-based Advertising using Multiple Independent Social Network Providers Privado [21] is a privacy-preserving group-based advertising system that works based on the cooperation of N federated servers each running by an independent authority. Servers receive the advertising requests of advertisers and personal interests of users in an encrypted format and are responsible to identify the target customers of the advertisers. In this module, we introduce the notion of *group-based advertising* to cope with the security issues raised by the personalized advertising methods [22, 23]. In group-based advertising, each advertising request is matched against a group of users' profiles. If a threshold number of profiles match with the advertising request, then the entire group members are shown the advertisement. We present a *formal definition of user privacy* where no adversarial entity must be able to link a particular attribute to a specific group member. We further supply a *formal security proof* of Privado in the setting where $N-1$ *malicious servers* may collude against the user privacy as well as may additionally employ several *fake users and advertisers*. Our design enjoys *advertis-*

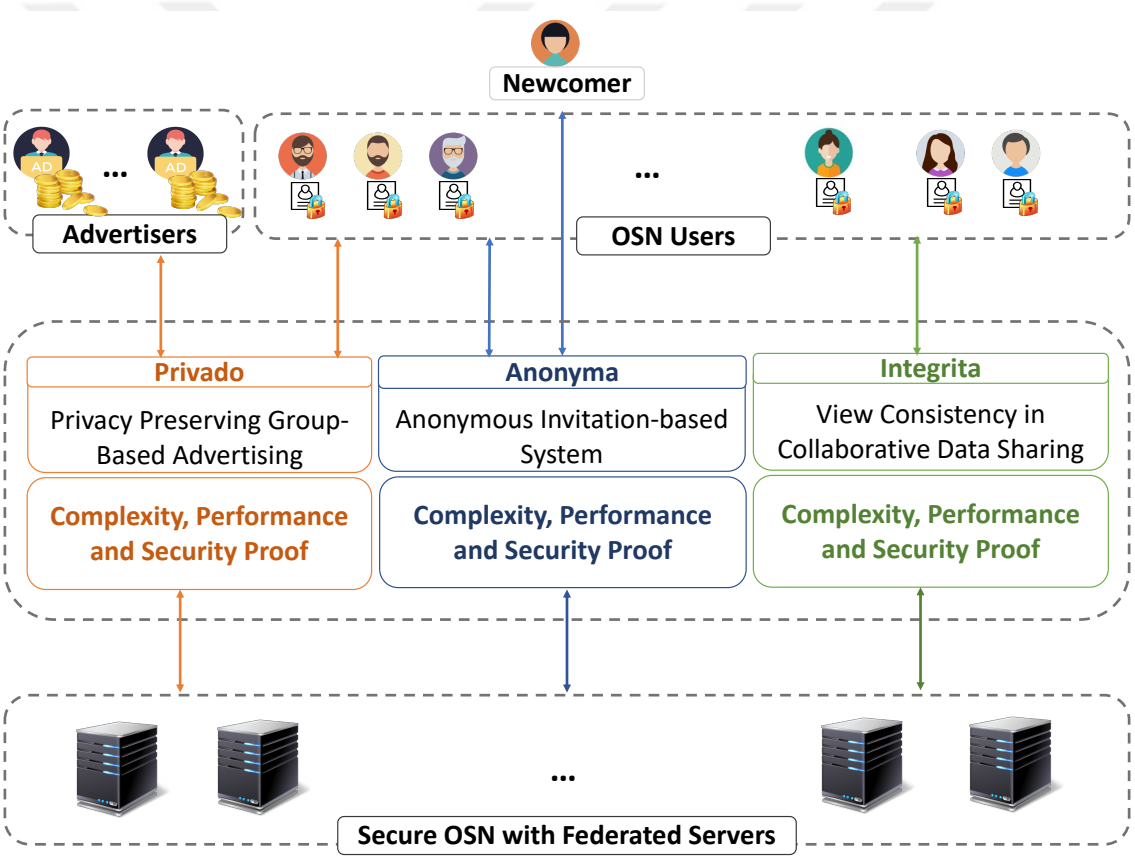

Figure 1.1: An overview of the thesis modules: Privado, Anonyma, and Integrita.

ing transparency, that is, once the profiles of users and request of the advertiser are uploaded, the entire matching procedure is run offline by the servers without further involvement of users or advertisers. The advertising transparency provides benefits for system performance where the servers' execution time (to match the advertisements to profiles) would not be constrained by the active time of users (who may not be online in a regular basis). We define *two performance metrics of Target accuracy and Non-Target accuracy* to be used in group-based advertising systems. Using empirical analysis, we capture the effect of group size and threshold value on the system performance and discuss their security implications. We also carry out experiments to examine Privado's *advertising running time* under a various number of servers and group sizes. Additionally, we argue about the *optimum number of servers* concerning user privacy and advertising running time. Chapter 3 is dedicated to the detailed design of Privado whereas Chapter 4 covers the performance analysis and security definition and proof of user privacy.

Anonyma: Anonymous Invitation-Only Registration in Malicious Adversarial Model Anonyma [24] is an authentication method that works based on the invitation-only policy and protects *inviter anonymity*. That is, a newcomer is able to prove to a group administrator that she has invited by a subset of group members yet does not require to disclose her exact inviters. At the same time, Anonyma preserves *invitation unforgeability* where no user with an insufficient number of inviters is able to authenticate herself to the group administrator. We provide *formal security definitions and proofs* for inviter anonymity and invitation unforgeability in the *malicious adversarial model*. In specific, a malicious adversary, targeting inviter anonymity, may corrupt the group administrator and some of the newcomer's inviters yet would never be able to learn the identity of other inviters. For the invitation unforgeability, a corrupted newcomer with insufficient inviters who additionally can eavesdrop communication channels among other users will be unable to successfully pass the authentication phase. Moreover, Anonyma is *efficiently scalable* in terms of the number of inviters. That is, the server can issue credentials to the new members

(without re-keying other existing members) to be immediately able to invite others.

In this module, additionally, an anonymous cross-network invitation-based system AnonymaX is constructed on top of Anonyma which can be of independent interest. In the cross-network design, a user joins one system e.g., Twitter, by obtaining invitations from members of another network e.g., Facebook. We supply a formal security definition and proof for AnonymaX against a malicious adversary. The construction of Anonyma and AnonymaX [24] are explained in Chapter 5. The performance analysis as well as the security definitions and proofs (of invitation unforgeability and inviter anonymity) are supplied in Chapter 6.

Integrita: Protecting View-Consistency in Online Social Network with Federated Servers Integrita [25] provides a data-sharing platform that empowers view consistency relying on N federated servers. A shared data (or object, interchangeably) can be a Facebook group page or individual walls, modeled as a sequence of posts accessible to a set of authorized users (e.g., members of the Facebook group). Users can perform read and write operations on the object through interaction with the N servers. While users shall act honestly and tend to achieve a consistent view of the object, servers may be malicious/Byzantine entities who may act arbitrarily, collude, compromise the view consistency by dropping, tampering with, and forging posts. Integrita guarantees that no coalition of $N-1$ *corrupted servers* is able to compromise view consistency and display a partial or incorrect content of the shared data to the users without being detected. In Integrita, we introduce a new level of view consistency called *q-detectable-consistency* in which the views of users toward the *object* cannot diverge for more than q sequence of posts without detection. Our proposal outperforms the state of the art (of centralized and distributed architectures) in multiple ways. Integrita is *communication-free* that is users can catch any inconsistency in their views relying neither on the users' collaboration in sharing their views in each operation nor an out-of-band communication. Every user is able alone (without relying on the presence of other users) to verify any server-side equivocation regarding her operation. Integrita provides a *replication-free* where the shared data

is not replicated over all the servers. Instead, only one copy of the *shared data* is present in the entire system and each server retains only (an identical) portion of it. Our numerical analysis asserts that by using Integrita, an OSN with 2 billion users saves up to 6890 Terabyte storage per year compared to the replication-based approach. Integrita matches the asymptotic performance parameters of the best centralized counterparts [9, 26], while enforcing a higher level of consistency. Moreover, our distributed storage management among multiple servers not only does not degrade the experience of user's interaction with the system concerning the performance but also *lowers the computational and storage overhead over each server* (compared to both centralized design and the replication-based proposals). While the storage of the shared data is distributed among N servers, *no communication is required at the servers side* for the coordination of users' read and write requests. Instead, all the communication happens solely between the users and the servers. More details on Integrita is provided in Chapters 7 and 8.

1.3 Organization

The rest of this Ph.D. thesis is organized as follows. In Chapter 2, we provide a comprehensive study of security issues of the distributed and centralized social networks together with the countermeasures. In Chapter 3, we present our proposed privacy-preserving advertising system, Privado, followed by its performance analysis and formal security definition and proof of user privacy in Chapter 4. We describe the design of Anonyma, the second module of our framework, in Chapter 5. Chapter 6 shall provide the the performance analysis of Anonyma, formal security definition and proof of inviter anonymity and invitation unforgeability. The third module of our framework, Integrita, is described in chapter 7. We further evaluate the performance of Integrita and present formal security definition and proof of view consistency in Chapter 8. In Chapter 9, we conclude the thesis and shed light on some future directions.

Notes to Chapter 1

1 https://www.statista.com/statistics/346167/facebook-global-dau/

2 friendi.ca

3 RetroShare.net.

4 https://www.statista.com/statistics/544001/facebooks-advertising-revenue-worldwide-usa/

Chapter 2

LITERATURE REVIEW

2.1 Introduction

In this chapter, we provide a fine grained classification of common security concerns and corresponding solutions in centralized and distributed OSNs [3]. We present state-of-the-art approaches for enabling security in OSNs considering three important aspects: data privacy alongside access control management, data integrity and secure social search. In contrast to prior work, we emphasize solutions of data integrity and secure social search, as well as solutions in DOSNs since this kind of networks provide more autonomy to their users. We also address architectural design principles of OSNs in relation to the security aspects.

Our classification of security aspects and solutions in OSNs is summarized in Table I. Data privacy deals with hiding user's data from illegitimate curious third parties while providing access to the legitimate ones. This hiding can be against service provider (in centralized OSNs) or other users of OSN. While data privacy concern is addressed, another problem, namely data integrity, should be considered. When we are sure only trusted friends see our data by the access policies we defined over our data, what will happen if someone forges a message on behalf of us or temper it. The former is data owner integrity issue and the latter is the data content integrity. The solutions regarding these concerns as well as historical integrity and integrity of data relations are discussed in this chapter. An important functionality in each OSN is to find and establish new friendships, that can be assumed as social search. Moreover, social search also addresses finding any content in a social network. The security and privacy of social search is important since it reveals some information about the searcher and other entities participating in the search process. For example,

if Alice wants to find her old friend Carol, then the relationship of Alice and Carol will be disclosed to service provider, or in the case of DOSN, to the intermediate nodes participating in the search.

Category	**Security aspects/solutions**
Data privacy	Information substitution
	Symmetric key encryption
	Public key encryption
	Attribute based encryption
	Identity based broadcast encryption
	Hybrid encryption
Data integrity	Integrity of data owner and data content
	Historical integrity
	Integrity of data relations
Secure Social Search	Content privacy
	Privacy of searcher
	Privacy of searched data owner
	Trusted search result

Table 2.1: Classification of security aspects and solutions in OSNs

There exist prior studies classifying security and privacy issues in OSNs [2, 27, 28, 29]. Some of them review and classify the proposed approaches in the literature [2, 27, 28], and another restates proposed methods [29]. However, data integrity and secure social search are important security issues which are not considered extensively in prior work. Our approach is to provide a fine grained classification for security issues and solutions in OSNs.

The rest of the chapter is organized as follows. In Section 2.2, we describe the architectural design principles of OSNs. Section 2.3 focuses on user data privacy protection and access control management techniques. Aspects of data integrity problem and the corresponding solutions are described in Section 2.4. Secure social search aspects and solutions are explained in Section 2.5. Finally, we conclude with a discussion of other concerns and open problems as future research directions.

2.2 OSN Architectures

2.2.1 Centralized Online Social Networks

All the users' private and personal data like their relationships, uploaded images, and posts, etc. are observable for the OSN provider. The amount of information available to the service provider and the ability to control them makes it an important source of security problems [28], [30]. The security issues raised by the central service provider are as follows [28]:

- **Data retention**: This issue refers to violation of the information lifetime, which makes the data available longer than intended. Provider takes backups of users' data and when users delete their data, service provider may pretend to delete, but nothing may change from the provider's view.

- **OSN employee browsing private information**: This issue is raised by full accessibility of OSN provider to data that can be misused by the employees of OSN provider.

- **Selling of data**: Advertisers need to know users' interests, habits and, preferences to be able to accurately find the target users who are interested in their products . For this reason advertisers buy users' data from OSNs. In this way, OSNs will be motivated to sell this information to get income.

Two main approaches are used to make the centralized OSN architecture secure:

- Some studies aim to overbear the security problems in existing OSNs, with the idea that it is better to improve the well-designed present OSNs, instead of migrating to a distributed architecture. A prototype Facebook application addressing some security issues of the Facebook platform by *proxy cryptography* has been built [31]. A *virtual private social network* without any collaboration of service provider is made to mitigate the privacy issues of social networking sites [32].

- Other studies propose a framework for a centralized OSN providing additional privacy benefits [33, 9, 34]. Hummingbird [33] tried to improve security and privacy of OSNs which are similar to Twitter. For this purpose, Hummingbird designed a prototype for implementation of Twitter by considering the protection of tweet contents and hashtags from the malicious centralized server. Frientegrity, a framework for social networking applications which is able to detect misbehaviour of malicious service provider, is proposed in [9]. Persona, [34] took the power of OSN providers in the case of determining the accessibility of users data for applications. Indeed, it gave users this autonomy to decide who can see their private data, even for the applications, with fine-grained policies.

2.2.2 Distributed Online Social Networks (DOSNs)

DOSNs can be classified based on system components' organization. There are two main system components: control and storage. *Control* deals with lookup (user and content) and identity management services, and *storage* addresses the data storage and availability. A high level classification extended from [1] is as follows:

- **Structured**: Users participate in a structured overlay, or use a third party structured overlay providing service. In such an organization, queries will be resolved in a limited number of steps. Most of the recent DOSNs use structured

organization and distributed hash tables (DHTs) for the lookup service. Prpl [35], Peerson [36], Safebook [37] and Cachet [38] all utilize structured control overlay. Vis-a-vis [5] designed its own structure *distributed location trees*, which provides efficient and scalable sharing.

- **Semi-structured**: Semi-structured DOSN makes use of super peers, which are a subset of all users who are responsible for storing the index and managing other users as proposed in Supernova system [39]. Such a structure may include lookup services and tracking of users up-time to find the best places for replication.

- **Unstructured**: No user in the system store any index, and operations of system are simply done by the use of flooding or gossip-based communication between users [40]. This kind of management has almost zero overhead.

- **Hybrid**: This kind of systems combine the benefits of the two types of organizations mentioned above. As the storage overlay, Cachet [38] uses hybrid structured-unstructured overlay using a DHT-based approach together with gossip-based caching to achieve high performance. In the hybrid organization of structured and semi-structured storage overlay of Prpl [35], users are allowed to store their data in a distributed and unstructured way, and then there is a process per user that federates the distributed storage of each user and act as a super peer. These super peers form a structured overlay of storage. The hybrid control overlay of Cuckoo [41] uses structured lookup for finding rare items, whereas, the unstructured lookup helps with the fast discovery of popular items.

- **Server Federation:** This is another architecture for decentralization of OSN [2]. The main purpose of this architecture is to distribute users' data among several servers which are running on separate storage entity. In this way none of them will have a complete global view of the private data stored in the system.

2.3 Data privacy

Data privacy protection is defined as the way users can fully control their data and manage its accessibility (i.e., to determine which part of data being shared with whom). The latter is known as access control management, and can be done by defining different groups with various access levels. A group is a set of users having a common feature (e.g., fans of football). To obtain the aforementioned goals, most of the proposed frameworks studied in this chapter use data encryption methods (except [32, 5]). For data privacy protection, the following solutions exist:

2.3.1 Information Substitution

Substitution means replacing real information with fake information. This solution is mostly used for hiding data from the service provider. For example, some predefined settings of OSNs force users to share their information in public (e.g., profile picture and name). In such a case, the user can share some pseudo information with service provider to be shown on his profile page and send the real information only to trusted friends. This information, in the form of XML files, are stored and processed locally on the friends' systems by the use of a browser extension [32]. A variant of this method can be applied for hiding users information from targeted ads. Users' data will be split into smaller parts called *atoms*. Users who trust each other can swap their atoms of the same type, which are associated with a unique index kept in a dictionary. For swapping an atom, its index will be encrypted, and the content of the resulting index will be used for swapping. Dictionary is public and only authorized users will be able to trace swapping results [42].

2.3.2 Symmetric Key Encryption

Symmetric (private) key encryption is a well-known technique for encrypting data. The term of symmetric comes from this fact that the same key is used for both of encryption and decryption [43]. In fact, the key is the shared secret between all parties

to access the private data. Since symmetric encryption methods use simpler operations, they have the advantage of running faster in comparison to other schemes. On the other hand, having the symmetric key for both encryption and decryption causes some integrity problems. In order to obtain integrity of data alongside utilizing the speed of this technique, symmetric key encryption is mostly used with the combination of other data integrity methods (see Section IV).

In terms of access control management in the symmetric key encryption systems, we should encrypt our data by the use of a symmetric key and then share it with the users who we want to be able to decrypt our data. For each new group, a distinct key should be defined. Adding a user to the existing group means sharing the group key with that user. For the revocation, we need to create a new key and re-encrypt the whole data. Of course, if someone already decrypted the data and kept a copy, we cannot revoke that.

2.3.3 Public Key Encryption

In the public-key encryption two different and separate keys are used for encryption and decryption [43]. Based on this reason it is also known as asymmetric encryption. These two keys may seem to be separate, but they are mathematically related. The keys named as **public key** and **private key** (secret key). The former used for encryption and the latter for decryption.

In order to manage users' data accessibility, data should be encrypted under the public keys of all group's members and then sent to them. When a user leaves the group, his public key will be deleted from the list of group members. For joining, the condition is reverse. Systems of Flybynight [31] and Peerson [36] use public key encryption.

2.3.4 Attribute Based Encryption (ABE)

ABE is a kind of public key encryption. In this scheme, some attributes make the secret key related to the ciphertext. For example, assume that there is a set of

attributes like 'relative', 'doctor', and 'painter'. One can decide to assign attributes of 'relative' and 'doctor' to one his friends named Alice. To do so, he must create a key containing 'relative' and 'doctor' attributes and give it to her [44, 45, 46]. After that point, Alice will be able to decrypt every message encrypted under the combination of attributes given in her key. The attributes embedded in the encrypted message are implicitly managing the accessibility of that message i.e., defining a group of members who are the exact audiences of that message.

In ABE, each message should be encrypted with an *access structure* defined over a set of attributes. This access structure can be any logical expression over the selected attributes, for instance ('relative' OR 'painter') or ('relative' AND 'doctor'). When the logic operation between attributes is OR, it means that having one of the listed attributes is enough. However, for the AND, the condition is different and having all the attributes is necessary. In ABE, it is enough to do a single encryption operation to construct a new group. Usual revocation methods for ABE use frequent re-keying. To remove the accessibility of a revoked user, the previous data which were accessible by him must be encrypted and stored again. This kind of re-encryptions causes an extra overhead to the access control management of OSN and makes it time-consuming. There exist two kinds of ABE based on the association of access structure with the users' secret keys or with the encrypted messages. In the **ciphertext policy ABE (CP-ABE)**, access structure is determined in the encrypted message and key contains a set of attributes while the condition in the **key policy ABE (KP-ABE)** is reverse. Ciphertext policy has a wide range of usage for supporting data privacy in OSNs such as Cachet [38] and Persona [34] making use of ABE.

2.3.5 Identity Based Broadcast Encryption (IBBE)

In a **Broadcast Encryption (BE)** scheme, there exist a broadcast channel among the list of the recipients [47]. Each user has a private key. The broadcaster selects a group of identities in order to encrypt the messages for them. The broadcaster then transmits the messages to the recipients listed in the channel. The recipients use their

private keys for the decryption.

In an **Identity Based Encryption (IBE)** scheme, public keys can be any arbitrary string [48] like email addresses. In such schemes, there is a trusted third party named Private Key Generator (PKG) that produces corresponding private keys.

In **Identity Based Broadcast Encryption (IBBE)** schemes, audiences of a broadcast group can use any identifier string as their public keys [49]. Considering the OSNs, the username or e-mail addresses of the members can be used as their public key for sending encrypted messages. From this point of view, IBBE is more flexible than ABE, since it addresses individual recipients instead of the whole group. Removing a recipient from the list would then have no extra cost. Systems such as [50, 34] also use this encryption approach.

2.3.6 Hybrid Encryption

A hybrid encryption is one which combines the convenience of a public-key encryption with the high speed of a symmetric-key encryption. In such systems, access control management is performed in two phases:

- Symmetric encryption of data by the use of a symmetric key.
- Applying public key encryption under the public keys of all group's members to encrypt that symmetric key.

While many implementations share this hybrid encryption framework [51, 33], there are differences in the choices of the symmetric and asymmetric-key encryption used. In Hummingbird [33], the symmetric key is derived by applying a combination of a **pseudo random function (PRF)** and a hash function on a particular part of message (hashtag). For the key dissemination an **oblivious pseudo random function** protocol must be followed between user and his friends.

Informally, a **PRF** [43] family is a set of polynomial time functions such that no one can distinguish between a function randomly chosen from this set and a function that its output is completely random. A PRF f takes two inputs: a secret s and a

variable x, and outputs $f_s(x)$.

An **Oblivious PRF (OPRF)** [52] is a protocol running between two parties, sender and receiver. The goal of the protocol is to compute $f_s(x)$ in a secure way. Receiver is the person who wants to know the value of function f in x and of course he determines x. The sender is the person who knows and determines the secret value of s, so he is able to compute f_s for any input. At the end of the execution of the protocol, receiver will learn $f_s(x)$ from the interactions while sender nothing.

The hybrid structure of the access control lists (ACLs) in Frientegrity [51] is organized in a persistent authenticated dictionary (PAD). Thus, ACLs are PADs, making it possible to access in logarithmic time. Persona [34] uses CP-ABE for data encryption and PKI to share the keys between friends. Cachet [38] uses a hybrid scheme of symmetric key encryption and CP-ABE: the symmetric key which is chosen randomly for data encryption, must be encrypted with ABE for the set of audiences to make them able to decrypt the data. Another hybrid scheme with combination of public key encryption and CP-ABE is applied to grant friends the ability of adding a comment to a post.

2.4 Data Integrity

A common security issue of centralized and distributed OSNs is data integrity [53, 54, 1]. The data integrity is defined as the protection of data from unauthorized or improper modifications and deletions [43, 55]. For the sake of simplicity, we explain and classify different aspects of data integrity in OSNs by the use of a short and simple scenario. Assume that Bob is organizing a party and wants to invite his friends to the party. Alice receives an invitation letter in a packet from Bob, containing this message: “Come to my party held at my home on Friday”. Considering this scenario, the different aspects of data integrity are listed as follows:

- **Integrity of the data owner**: How Alice can be sure that the sender of the message is Bob?

- **Integrity of the data content**: Is the content of the message valid? For example, did Bob really say that there will be a party on Friday at his house?

- **Integrity of data history**: Assume that Bob holds several parties per month, all on Fridays. Alice had been invited to some of them (by receiving invitation letters). Is this invitation letter valid for an upcoming event or has it already expired? Also, was this message delivered to Alice at the proper time or in an even weaker assumption, was this message delivered to Alice at any time?

- **Integrity of the data relations**: Is this message issued for Alice or is it Bob's invitation to someone else but sent to her?

Commonly used methods to protect data integrity are based on digital signatures [56, 37, 36, 40, 38, 35]. A digital signature is a mathematical scheme used by the issuer of a digital message in order to convince the recipient about the integrity of the message. Digital signatures are based on public key cryptography [43]. In most of the cases, the message is signed indirectly. In the other word, first the hash of the message obtained by employing a hash function. After that, the hash of the message is signed by a the digital signature scheme. Hash functions can map inputs with different sizes into a fixed length values [43]. For the sake of saving time and space, signing a hashed message is preferred. Moreover, for security, it is needed that the hash function is collision-resistant; so it is very hard to find different messages with the same hash output.

2.4.1 Integrity of the data owner and the data content

The integrity of the data owner and content can simply be guaranteed by the use of digital signature [56], assuming the public key distribution problem is solved. For the signature verification, it is important to know the valid verification key of each signer. One solution is distributing proper keys out-of-band like physical meeting [36, 51] or transferring the keys via e-mail [5].

2.4.2 Historical integrity

For the data history integrity, one solution is to use *hash chaining* alongside digital signature. In this method, the digital signature must be applied on each entry published by a user, and includes the hash of at least one of his prior posts. This causes a provable partial ordering for his posts [40]. Another solution is to establish a dependency between the timelines of different publishers [40]. In this solution, the publisher adds the hashes of prior events from other participants alongside using the digital signature. In this way, a provable order between their messages will be established.

Fork-consistent systems can be used for ensuring historical integrity. In [51], authors proposed object history tree accompanied by a fork-consistency approach. The object history tree data structure addresses historical integrity problem where a malicious service provider or any data storage utility cannot present different clients with divergent views of the system's state. As an example of this scenario, assume the situation where the service provider hides some parts of our friends' updates by providing just a partial view to us. Clients share information about their individual views of the history by embedding it in every operation they perform. As a result, if the clients who have been equivocated by the service provider communicate to each other, they will discover the provider's misbehaviour. In this method, the service provider also digitally signs the root of object history tree in order to prevent the client from later falsely accusing the server of cheating.

2.4.3 Integrity of the data relations

To guarantee the links between two entities in the system, for example a post and corresponding comments, one solution is to embed a proper signing key for signing the comments of that post. The signing key is encrypted in a way that only authorized users can decrypt and use it for posting a comment to that particular post. Corresponding verification key is also located in the content of the post. This verification key can be used to verify whether the comments belongs to the post or not, and

also to verify the privileges of the commenter [38]. Each post will contain a different signature key, which enables a different sub-groups of the users to write a comment for different posts.

2.5 Secure Social Search

Searching in digital social space is a crucial component of OSNs [57] as the social network users mostly do not prefer to be restricted to the existing friends and they intend to find new friends with common interests. Hence, for supporting social search, disclosing some information about users' profiles is required. The more information that is available, the more accurate the social search results would be. Thus, a trade-off between search capabilities and privacy is raised. While finding friends is one application of social search, advertising is another kind of searching where an advertiser searches for target users with a related interest to advertise products. Security concerns related to social search can be classified as follows:

- **Content privacy**: Privacy of content addresses information leakage by searching a content. Since the content of searched subject can reveal the interest of the searcher, its privacy must be guaranteed.

- **Privacy of searcher**: Hiding the identity of searcher is an important issue as it is also supported in the existing OSNs like Facebook. For instance, if Alice is searching for one of her old friends, Facebook will list a series of friends close to the criteria Alice is searching for, while none of the listed result persons would be informed about this search. Hiding the identity of searcher from the service provider and other OSN's users who may participate in the searching process (in the DOSNs) is also a concern.

- **Privacy of the searched data owner**: It is important for other users to be able to determine to which extent their data would be available for the system's searches.

- **Trusted search result**: How much trust-able is the result of the search? In the case of finding friends with common interests, one user in the search output may be a better choice among all the results when level of trust and popularity is considered.

Based on the system's objective, different levels of privacy can be applied considering the security concerns. For example, privacy of searcher in an advertising scenario might not be important. Since the advertiser wants to introduce itself to the users, there is no concern about its identity. On the other hand, in the case of finding a friend, the privacy must be guaranteed for most of the security concerns.

2.5.1 Content privacy

Blind Signatures can help to provide the privacy of content. Blind signature means signing the document without knowing what the document contains [58]. Hummingbird [33] follows an interesting approach where a signature of a massage's keyword is used as a key to encrypt the message. By considering this idea, anyone who gets the signature on that keyword can also decrypt the message. This method can be used in Twitter for publishing and subscribing. Each subscriber will get the signature on the main keyword (hashtag) of each tweet, by the use of the blind signature, while his interest will not be revealed to the publisher.

2.5.2 Privacy of searcher

A solution to support privacy of searcher is to use **proxy**. In this method, the real identity of users will be replaced by aliases via the proxy server. Since the proxy server knows all the aliases of their users, it can forward messages correctly. Servers cannot see the real names of other servers' users. However, the security of this approach can be under the risk by collusion of proxy servers [2].

Trusted friends network is another approach that can be used to support privacy of searcher. In this solution, each user connects directly to trusted friends to forward messages. It will cause a concentric circle of friends around each user, which

makes it possible to communicate with the user without revealing identity or even IP address [37]. In the above solution, some relaxation considered that friends of a user are trusted parties. That is, the user is sure that his trusted friends would not cause any security problems by knowing that he is the source of a request.

Zero Knowledge Proof (ZKP) alongside using pseudonyms is another solution. By employing the ZKP, one can prove that a given statement is true without revealing any extra information about that statement. In the other word, during a ZKP, the only information that is revealed about the statement is that it is true or not [59]. A user can use a pseudonym while searching in the network, and when (s)he wants to reach a content belonging to another person, (s)he uses ZKP to prove having privileges to access [60].

2.5.3 Privacy of the searched data owner

Regarding the security concern about the privacy of the searched data owner, one solution is to define **resource handler** for data. In this way, every data item has a handler as a reference to that data. For example "Alice's birthday" instead of "26 October 1990". When one is interested in knowing the content of that handler, he must prove himself to the data owner and then get access to the real content [60].

2.5.4 Trusted search result

For finding the best choice among the social search results, one solution works based on the real life assumption that the trust between friends are the means for delivery. It means that if Alice trusts Bob and Bob trusts Sara, then Alice can trust Sara too. The amount of trust assigned to Sara by Alice, based on the search chain from Alice to Sara, is a function of trust levels of every intermediate friend of that chain to the successor friend of that chain [61]. In this way, the target users can be ranked and then chosen.

2.6 Conclusion and Open problems

Popular OSNs have hundreds of millions of active users, and these networks serve new forms of communication, organization, and information sharing. Most of OSNs present the followings functionalities: profile creation, access control management, commenting and social search (finding friend and friendship establishment). These functionalities must be served in a secure manner and they should not affect the privacy of users. Security issues raised in OSNs can be classified into three general categories: Data privacy, data integrity and secure social search. Most of the existing attacks in OSNs threat these categories. In this chapter, we reviewed recent solutions of the aforementioned security challenges. However, there exist several security problems which have been discovered but have not been fully solved yet.

- **Implicit information leakage**: Being sure that your data has been shared with authorized users is a different problem when compared to the problem of recognizing which part of data is sensitive. Certain kind of information can implicitly be derived from published data. For example, a phone number by itself does not contain any important information but it can be shown that the name of phone's owner alongside with the name of his parents can be extracted by having the phone number [62]. It is important to identify what kind of information can be inferred from a published and seemingly simple data. This problem, related to such information which are not explicitly noticeable but implicitly extractable, is called implicit information leakage. To the best of our knowledge, no solution for the implicit information leakage has been proposed so far.

- **Data resharing**: As long as the friends of a user are trust-able and do not reshare the data which the user shared with them, no problem will be faced. However, there is no control if they want to reshare the user's data with others. In fact, security and privacy is a collective phenomenon [63]. The main problem is how it would be possible to prevent a user's friends from re-sharing the user's

data. This problem is similar to showing a hard-copy of an image to a person while he can see the image under the owner's consent, and he cannot make a copy of it and he has to turn it back to the owner.

- **Privacy preserving advertising**: Another problem is to provide privacy preserving advertising for a service provider storing encrypted data of users in order to get income. It is trivial that the service providers commonly reject a business model that prevents them from users' plaintext data mining for marketing purpose. However, such business model has not been well studied yet. Although there has been some work on privacy preserving advertising systems [64, 65], the development of business models that can support privacy preserving services hosted with third-party providers needs to be investigated further.

There are also other concerns out of the scope of this chapter, however there exist other studies that covered these concerns:

- **Protection of data from API** : Online social networks usually employ application systems. In most of them, there is no fine grained policy that can control the access of the application to the personal content. In other words, after the user employs an application, he implicitly gives the application all the accesses to the personal content it wants. A recent study [29] reviews proposed works related to this concern.

- **Network inference**: The network inferences show that it is possible to gain access to the users' information that do not consider explicitly in their OSN's shared data. In some cases, this access possibility proved to obtain even very easily. The concepts of identifying hubs, making link predictions [66] and inferring user attributes have been discussed in [29].

- **Sybil attacks**: Sybil attacks are considered very dangerous for preserving the proper operation of an OSN. In a sybil attack, the reputation system of a network will be subverted by attacker who makes (usually multiple) pseudonymous

entities. There are multiple ways in which the attacker can subvert the reputation system. For example, by sending spam, trying to de-anonymize the social network by use of sybil friendship, and simply impersonating other users to accuse them [29].

- **Hiding the social graph**: A social graph represents the users connections among each other, their preferences and behaviour in the social network. Users' relations are source of important information. Some network inferences can be done by user's friends information and interests. Work of [2] has reviewed potential approaches to protect the social graph.

- **OSN anonymization and de-anonymization**: OSN providers publish their data for the research activities in the industry and academia. However, very sensitive information could be revealed explicitly and implicitly from the social graph. There should be an "anonymized" way that let the OSN providers to publish these data sets in a way that minimizes the possible risks for their users. Obtaining the anonymized data, one can reverse the anonymization process and identifies the corresponding nodes to the data set (which is known as de-anonymization). Prior studies of [66, 29] cover this concern.

Chapter 3

PRIVADO: PRIVACY-PRESERVING GROUP-BASED ADVERTISING USING MULTIPLE INDEPENDENT SOCIAL NETWORK PROVIDERS

3.1 Introduction

Motivation: OSNs offer advertising service where the advertisers pay OSNs to find their targeted customers out of social network members. In particular, in an advertising scenario, every user fills and uploads a profile (like the Facebook profile) that contains a set of attributes varying from demographic information to personal interests and hobbies (e.g., age, education, music interests, sports activities. etc.). Likewise, advertisers submit their requests indicating a set of attributes for the intended customers. OSN provider examines the profiles to find target customers and serves them with proper advertisements. Advertiser will be charged accordingly.

The collection of user's personal data at the OSN side raises privacy issues [3] (as we discussed in Chapter 2) for which existing studies propose utilizing data encryption. However, that immediately disables advertising service, and hence leaves secure OSNs with no convincing commercial model. To fill this gap, we propose Privado [21] as a privacy-preserving advertising system by which secure OSNs can efficiently perform advertising and find target customers using the privacy-protected profiles of users. Privado follows *group-based advertising* notion and also satisfies *advertising transparency* (we elaborate on these two terms in the followings). The former helps user privacy and the latter is essential for the system performance.

Personalized vs Group-based Advertising: Group based advertising is a notion that originated from PPAD [67] as a solution for the security issues that arise in

personalized advertising. Any secure personalized advertising ultimately compromises user privacy. That is, when an encrypted profile is matched against an advertising request (that contains a set of attributes), the success or failure of the matching indicates the presence or absence of those attributes inside that encrypted profile. As a concrete example, let F represent the personalized advertising function that is run by a trusted third party (TTP). The OSN provider, who has received advertising requests from the advertisers, would like to query F and find the target customers for each advertising request. F receives two inputs: a set of attributes from a user U which indicates her interests like $AttSet$=("art", "football", "music", "programming") and another set of target attributes like $TAud$=("football", "music"). F outputs 1 or 0 to the OSN provider indicating whether $AttSet$ matches $TAud$ (more formally, $TAud \subseteq AttSet$) or not. In the example above, $F(AttSet, TAud)$ would return 1 as $AttSet$ contains the attributes indicated in $TAud$. As such, the OSN provider would immediately learn that U is interested in "football" and "music" (even though the OSN provider never gets to see the plain set of user attributes since everything went through a TTP). Thus, after enough trials, the OSN provider can learn all the attributes inside an encrypted profile. Note that this leakage is inherent to any personalized advertising approach regardless of how the secure matching is carried out. In essence, this leakage is through the output of the matching but not the way the matching result is computed. To mitigate this privacy issue, Taheri et al. [67] introduce the notion of group-based advertising and provide a cryptographic solution for it. In group-based advertising, users are split into equal size groups during their registration and the group formation is static. Once the groups are set, every advertising request is matched against the encrypted profiles of each group. If the number of the matched profiles inside the group exceeds a threshold, the group is marked as targeted and all of its members are shown the advertisement. As such, and assuming users are divided into groups of size k, we redefine the advertising function as $F(AttSet_1, ..., AttSet_k, TAud)$. F receives k sets of attributes $AttSet_1, ..., AttSet_k$ from k members of a group as well as another set of target attributes $TAud$ and

outputs the number of matches in the group i.e., $|\{i|1 \leq i \leq k \text{ and } TAud \subseteq AttSet_i\}|$ where $|.|$ indicates the size of the set. We remark that while the functionality is defined such that the users submit their attributes to the TTP directly, in practice, this is done via the OSN provider who collects the privacy protected attributes of users (we consider the idealized privacy protection where the OSN provider does not receive users attributes and only gets to see the output of F). Note that the group matching outcome does not (and must not) indicate attributes of which user exactly matched the request; instead, it only asserts the total number of matches. When that number exceeds a threshold, the OSN provider serves the advertisement to all group members. Hence, unlike the personalized advertising, the group matching result would not be linkable to the individual members. In other words, the OSN provider only learns the aggregate of attributes rather than the exact targeted individuals.

Advertising Transparency: In addition to user privacy, *Advertising Transparency* [67] is another requirement to be fulfilled in OSN advertising systems. That is, once the profiles of users and request of the advertiser are uploaded, the entire matching procedure is run offline by the servers without further involvement of users or advertisers. Users are served by proper advertisements (which are found via the matching procedure) in their next log-in. The advertising transparency provides benefits for system performance where the server's execution time (to match the advertisements to profiles) would not be constrained by the active time of users (who may not be online in a regular basis). As another requirement of advertising transparency, users and advertisers should not need to know or communicate with each other. This property is essential considering the fact that in an OSN like Facebook there are 2 billion monthly active users[1] and 6 million monthly active advertisers[2], and making them communicate is impractical.

Related Works: Advertising problem in secure OSNs with the aforementioned requirements results in a unique setting that has never been addressed in prior studies except for [67]. In the context of secure online behavioral advertising [68, 69, 65, 64], provable user privacy and advertising transparency are lacking features. Proposals in

server-aided private set intersection [70, 23, 71, 72], as well as server-aided two-party computations, cannot achieve user privacy under the coalition of server and advertiser [23, 22, 73] or they lack advertising transparency [70, 23, 71, 72]. The similar problem applies to the context of public-key encryption with key search [74, 75, 76, 77, 78]. Secure Multi-party Computation (SMPC) protocols [79] come with an issue where the communication complexity of parties (i.e., servers in our case) is linear with the depth of computed circuit (i.e., matching procedure). In Privado, while we utilize some SMPC techniques [80, 79], we propose a very carefully designed matching function which avoids multiplication gates and the subsequent communication overhead (which can be of independent interest).

Privado vs PPAD: Taheri et al. proposed PPAD [67] as the first privacy-preserving advertising proposal for secure OSNs. PPAD achieves both user privacy and advertising transparency but under two constraints. First, PPAD relies on two non-colluding servers whose cooperation compromises user privacy. Second, user privacy is guaranteed under *honest but curious* (HbC) adversarial model where all the parties must follow the protocols precisely.

In Privado, we mitigate the constraining conditions of PPAD. Firstly, we relax the assumption of PPAD about having two non-colluding servers by proposing a distributed design consisting of N servers each being operated by an independent authority. User privacy is protected even if $N-1$ servers collude. As the second enhancement on top of PPAD, Privado offers a stronger security guarantee by withstanding the malicious adversarial model where entities are allowed to deviate from the protocol specifications. In Privado, we assume that up to $N-1$ servers may maliciously collude. Also, the adversary can register fake advertising requests and user profiles.

PPAD to Privado non-triviality: A naive attempt to extend PPAD to the N server and malicious setting can be applying SMPC techniques. However, simply running SMPC between N servers would be inefficient compared to Privado. In particular, SMPC assumes the servers know the inputs, whereas, in the current setting,

the inputs of the servers are encrypted user profiles.

Another non-triviality lies in the fact that PPAD makes use of a privacy service provider as a trustworthy entity through which group members receive some secret information (this secret data shall be integrated into users profiles and would help in computing the group matching results). However, in Privado, none of the servers are trusted. Thus, another challenge is to perform the same confidential data dissemination using N servers which are additionally malicious. Notice that any solution to this problem should also comply with user privacy and advertising transparency.

N Server Motivation: In Privado, we utilize N servers each run by an independent provider. As long as at least one server would not collude with the rest, we protect the privacy of the users. Such setting has been similarly utilized in outsourced multi-party computations [81, 79, 82, 83]. In practice, this can be realized by having an OSN whose servers are provided by the ISPs of multiple distinct countries. As such, collusion is less likely since colluding countries would have to mutually compromise the privacy of users of their own country to their opponents. Another motivation for such a realization is that all the contributing authorities would financially benefit from such a design where they share the advertising revenue. Deploying multiple conflicting parties for the sake of privacy is similarly sought in the electronic voting (e-voting) systems [84, 85]. There, the conspiracy of tallying authorities is restrained by deploying multiple conflicting entities such as political parties of a country.

3.2 Related Works

In this section, we describe the related works and compare them with Privado. We first discuss PPAD [67] as our main counterpart. Then, we additionally investigate four other different areas that are the most similar to the present context and identify their shortcomings when applied to the OSN advertising scenario.

PPAD: Privacy-Preserving Group-Based Advertising in Online Social Network: PPAD is a secure advertising system which deploys group-based matching notion and makes use of two servers called the OSN server and the Privacy Service

Provider (PSP). PPAD satisfies advertising transparency as well as provable user privacy. However, it has two main shortcomings. Firstly, in PPAD, the data decryption power is given to the PSP, and hence user privacy is tied to the fact that the OSN provider would not be able to corrupt that single PSP. This is a huge trust assumption to be placed on a single party and might be very hard to achieve in real life. Secondly, the adversary is an honest but curious entity which is supposed to precisely follow the protocol descriptions.

Secure Online Behavioral Advertising Systems (SOBA): In SOBA, a server named broker is connected to a set of web page owners. As users visit different web pages, the web page owners communicate this data with the broker. The broker then is able to create a behavioral profile for each user according to the visited web pages. The advertising companies pay to the broker to gain users profiles and be able to advertise for their target users on the web pages. The proposals in the context of SOBA are not applicable to the OSN advertising due to the lack of advertising transparency [68, 69, 65] and provable security [64].

Server Aided Private Set Intersection (S-PSI): In an S-PSI protocol, two parties, holding their own private sets of elements, are willing to compute their sets' intersection with the help of a third server. In this context, solutions either violate the advertising transparency [70, 23, 71, 72] or when transplanted in the OSN advertising they fall short in providing user privacy (as we define in advertising context) [23, 22]. In fact, PSI protocols usually make two assumptions that conflict with our required features: 1- they assume that the data holders can directly communicate some secret information unbeknown to the server which is against advertising transparency 2- they assume that set holders and the server are mutually untrusted and non-colluding [86, 73] whereas in our setting advertiser (one of the set holders) is cooperating with the server. Moreover, none of the prior PSI protocols offer a group-based method (matching multiple sets with one) and such an extension is not trivial.

Public Key Encryption with Keyword search (PEKS): PEKS is an attempt to outsource searching over data that is encrypted using a public key system. Encrypted

data is usually outsourced to an external server who is responsible to respond to the data owner's search queries. The first issue with PEKS solutions is that they achieve privacy by assuming that the querier and the server are not colluding. This is in contradiction to our assumption (advertiser and server collude). Server and querier collusion enables keyword guess attack [74, 75, 76, 77, 78] which is against user privacy in the advertising scenario.

Server Aided Two/Multi-Party Computation (S-2PC/MPC): In S-2PC/MPC protocols two/multiple parties holding their private data wish to compute a function of their inputs by the help of server(s). Parties only learn the output of the function and nothing else.

The major incompatibility of MPC/S-2PC protocols for secure OSN advertising is that they cannot support advertising transparency. Advertising transparency forbids any direct communication among users (group members), between users and the advertisers, and user/advertiser with the OSN servers except the registration phase. The inability of MPC protocols [87, 88, 89, 90, 91, 92, 93, 94] to support advertising transparency roots in the fact that at some pint in the offline or online phase of the protocol players need to communicate some secret information with each other. This communication has to be repeated per function evaluation. When we apply this scenario to group-based OSN advertising, this would cause the users and the advertisers to stay in contact (either to each other or with the OSN provider) for every execution of the function. This violates advertising transparency.

In S-2PC proposals, the collusion between one of the data holders and the server [95, 96] will compromise the privacy of the other data holder. Adoption of such solutions for our advertising problem in which we allow servers collude with the advertisers result in the violation of user privacy.

MPC protocols [79] also come with an issue where the communication complexity of parties (i.e., servers in our case) is linear with the depth of computed circuit (i.e., group-based advertising/matching procedure). As such, most of MPC designs aim at proposing an efficient protocol to evaluate the multiplication gates. However, none of

the MPC proposals deal with how to design a function whose circuit representation ends up with the minimum number of multiplication gates (to lower the communication rounds among the parties who are geographically distant and require to run the protocol many times). As such, designing applications out of MPC protocols is not a trivial or straightforward task and counts as an orthogonal research problem.

3.3 System Model

3.3.1 Model

Privado is comprised of *users*, *advertisers*, and N *servers* denoted by S_1,...,S_N. The system overview is illustrated in Figure 3.1. N servers jointly create an encryption key pk and distribute the corresponding decryption key shares $dk_1, ...dk_N$ among themselves. Hence, performing decryption requires all the servers to contribute using their decryption key shares. Servers also decide on the group size and threshold value to be used in the group matching procedure.

Users fill in a profile (similar to Facebook profile) in which they provide their preferences as a list of attributes varying from demographic information to personal interests and hobbies (e.g., age, education, music interests, sports activities, etc.). Then, the user encrypts his profile under pk and hands it over to all the servers. On the other side, advertisers create their advertising requests, which include the attributes they seek in their target customers, e.g., {painting $\wedge$ football $\wedge$ computer engineer}. Requests are then submitted to each server in the plaintext form. Each server has a local database in which it stores all the advertising requests and encrypted user profiles registered in the system.

Privado applies the group-based advertising notion. For that, users are randomly divided into groups of identical size at their arrival time. As discussed earlier, the grouping of users is essential for user privacy and should not be done based on similar interests. For every advertising request, N servers jointly examine all the groups and identify the target ones: i.e., groups whose number of matched profiles exceeds a system-wise threshold. All the members of a target group are served by the adver-

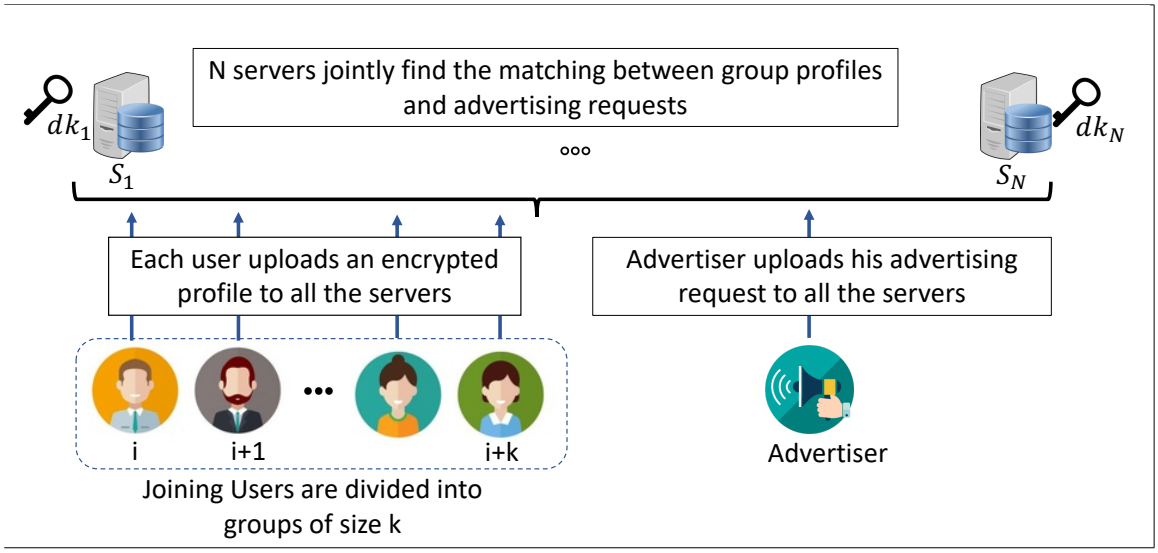

Figure 3.1: Privado System Overview. Users encrypt their profiles using an encryption key whose corresponding decryption key is split among the N servers as $dk_1, ..., dk_N$.

tisement (since advertising to a subset would violate user privacy). The non-target groups are skipped. Subsequently, the advertiser is charged based on the number of target groups (hence users) to which the advertisement is shown. We remark that group-based advertising degrades advertising accuracy (losing some target customers) where we do not advertise to the groups with insufficient target members. Also, there would be some non-target users who receive irrelevant advertisements since they belong to a target group. Hence, this is the cost of achieving user privacy. We refer the interested readers to PPAD [67] for an extensive analysis of the advertising accuracy in group-based advertising over various group sizes and threshold values.

3.3.2 Security Goal

The security goal of Privado is to protect user privacy. User privacy indicates the inability of any adversarial entity to bind the group matching result to a particular group member, as defined in PPAD [67]. This means that for a target group, none of the servers would be able to conclude whether a particular group member has the attributes of the advertising request or not. However, it is important to realize that

any advertising system is inherently prone to some implicit privacy leakage regardless of how secure it is designed. That is, once a group is qualified/disqualified as a target for an advertising request, this reveals the inclusion or exclusion of the queried attributes among the group members. This leakage is inevitable even though the matching occurs entirely on the encrypted profiles. In personalized advertising, this leakage immediately breaks user privacy [23, 22, 70, 23, 71, 72] whereas in the group-based counterpart the inclusion or exclusion is not attributable to a particular group member. PPAD supplies a formal security definition for user privacy that we present in Section 4.2.1. We utilize this definition to formally prove the security of Privado.

3.3.3 Adversarial Model

Privado considers the adversarial model where an adversary may corrupt any subset of $N-1$ servers, create arbitrarily many advertising requests and fake $k-2$ profiles per group (k is the group size). Notice that, having more than $k-1$ fake profiles in a group converts the group-based advertising to the personalized variant (adversary knows the content of $k-1$ profiles hence can immediately link the group-matching results to the unknown profile), which, as discussed earlier, violates user privacy. We consider **static corruption**, i.e., the adversary selects which party to corrupt before the protocol starts. Corrupted parties are **active/malicious adversaries** who may refuse to follow the protocols' specifications. We assume parties communicate through **insecure but authenticated channels**.

3.4 Definitions and Preliminaries

3.4.1 Notations

We write $x \leftarrow X$ to denote picking an element x uniformly at random from set X. $a||b$ represents concatenation of a with b. $|A|$ stands for the number of elements in set A. $\boxed{a}$ represents a ciphertext embodying the value a. Likewise, $\boxed{B}$ where B is a vector, means the element-wise encryption of B. For shorthand, we use PPT for a

probabilistic polynomial time entity. We write $Enc_{pk}(m, r)$ to denote encryption of a message m using an encryption key pk and a randomness r. Table 3.1 summarizes the notations used in Privado.

1^{λ}: The security parameter
k: The group size
N: Number of servers
S_i: The i^{th} server
Thr: The group-based advertising threshold
p: The size of Bloom filter
$TEnc$: (N, N)-threshold homomorphic encryption scheme
Enc: The encryption algorithm
$DDec$: Distributed decryption protocol
ek, dk: Encryption and decryption key
r: Randomness used for the encryption
GID: The group number
$\pi(i)$: The pseudo random permutation of i
$\pi^{-1}(i)$: The inverse of π at i
Δ: The membership identifier set
δ_i: The i^{th} element of the membership identifier set
$\boxed{\Delta}$: The element-wise encryption of δ
$\boxed{\delta_i}$: Encryption of δ_i
Δ_{π}: The shuffled version of set Δ according to the pseudo random permutation π
σ_j^{POPR}: Non-interactive zero knowledge proof of plaintext range generated by the j^{th} user
σ_j^{POCM}: Non-interactive zero knowledge proof of correct multiplication generated by the j^{th} user
$AttSet_j$: The attribute set of j^{th} user
$UName_i$: The username of i^{th} user
BF: Bloom filter

BF_j: The j^{th} user's Bloom filter generated over $AttSet_j$

$\boxed{BF_j}$: Element-wise encryption of BF_j

$\boxed{Pf_j}$: The encrypted profile of j^{th} user

$TAud$: The set of attributes selected by the advertiser for the target audience

$Req = \{r_1, ..., r_p\}$: The Bloom filter of the advertising request

R: The index of the set bits of the Req

$|R|$: The size of R

RID: The ID of the advertising request

Φ: The aggregate of group profiles

$\boxed{\Phi}$:The Encryption of Φ

α_j: The sum of bits of Bloom filter of j^{th} user according to R

$\boxed{\phi_j}$: The encryption of the product of $\delta_{\pi(j)}$ and α_j i.e., $\boxed{\delta_{\pi(j)} \cdot \alpha_j}$

Table 3.1: Notations used in Privado

3.4.2 Definitions

Negligible: A function f is called negligible if for all positive polynomials p, there exists a constant C such that for every value $c > C$ it holds that $f(c) < \frac{1}{p(c)}$.

Computational indistinguishability: A probability ensemble $X = \{(a, \lambda)\}_{a \in \{0,1\}^*, \lambda \in \mathbb{N}}$ is a series of random variables which are indexed with a and λ where a is the input and λ is the security parameter. Two distribution ensembles X and $Y = \{(a, \lambda)\}_{a \in \{0,1\}^*, \lambda \in \mathbb{N}}$ are said to be computationally indistinguishable (written as $X \equiv_c Y$) if for every non-uniform polynomial-time distinguisher D, there exists a negligible function $negl(.)$ such that $\forall a \in \{0,1\}^*$ and $\forall \lambda \in \mathbb{N}$ [97]:

$$|Pr[D(X(a, \lambda)) = 1] - Pr[D(Y(a, \lambda)) = 1]| \leq negl(\lambda) \tag{3.1}$$

Secure Multi-Party Computation: Consider the ideal functionality $F(in_1, ..., in_N) = (f_1(in_1, ..., in_N), ..., f_N(in_1, ..., in_N))$ running by a trusted third

party that receives inputs (in_i) from i^{th} party and delivers $f_i(in_1, ..., in_N)$. We refer to such execution as the IDEAL world. Let γ^F be a multi-party protocol to compute F. The execution of γ^F by the interaction of parties constitutes the REAL world. γ^F is said to securely realize F if for every PPT adversary A with auxiliary input $aux \in \{0,1\}^*$ attacking protocol γ^F, there exists a PPT simulator Sim for the ideal functionality F, that $\forall$ security parameter λ:

$$\{IDEAL_{F,Sim(aux),P_c}(in_1, ..., in_N, \lambda)\}\} \equiv_c \{REAL_{\gamma^F,A(aux),P_c}(in_1, ..., in_N, \lambda)\} \tag{3.2}$$

The left and right side of this equality represent the output of parties in interaction with F and γ^F, respectively. P_c is the set of corrupted parties controlled by the adversary/simulator. $\equiv_c$ stands for computational indistinguishability.

Hybrid Model: Let γ^F be a multi-party protocol that securely realizes ideal functionality F and assume θ is another protocol that makes use of γ^F as a sub-protocol. In the hybrid model, the security of θ can be proven by replacing γ^F with its ideal functionality F (as if there is a trusted third party running F). This would be called F-hybrid model [97].

3.4.3 Preliminaries

Bloom Filter [98] is a data structure for the set representation which also supports insertion and membership check queries. A Bloom filter is formed as a bit array of size p, and d hash functions denoted by $H_1(.), ..., H_d(.)$. To insert an element x to the Bloom filter, all the hash functions are evaluated on x; i.e. $H_1(x)..., H_d(x)$. The output of the hash functions indicate the indices of the bit array that shall be set to 1. To check the membership of an element y, similarly, the hash functions are evaluated on y which results in d indices. If all the corresponding bit values are 1, then y counts as an element of the set with the probability of $1 - (1 - ((1 - \frac{1}{p})^d)^e)^d$. e is the total number of elements inserted into the Bloom filter. Otherwise (if at least one of the checked indices is 0), y is not a member. Throughout the paper, we refer

to the creation of a Bloom filter with $BFCreate(Att)$ where Att is the set of elements to be inserted.

Super-Increasing Set: A super-increasing set $A = \{a_1, ...a_k\}$ of size k is a set of k positive real numbers such that each element of the set is greater than the aggregate of its preceding elements in the set [99]. That is,

$$\forall i, a_i > \sum_{j=1:i-1} a_j \tag{3.3}$$

IND-CPA Encryption Scheme: For a public key encryption scheme $E = (Gen, Enc, Dec)$ we define the IND-CPA game $PubK^{CPA}_{A,E}(\lambda)$ as [100]

$$\begin{aligned} &(pk, dk) \leftarrow Gen(1^\lambda); (M_0 = \{m_{0,i}\}_{i=1:L}, M_1 = \{m_{1,i}\}_{i=1:L}, history) \leftarrow A(pk) \\ &\text{s.t. } |m_{0,i}|_{i=1:L} = |m_{1,i}|_{i=1:L}; b \leftarrow \{0,1\}; C = \{c_i \leftarrow Enc(Pk, m_{b,i})\}_{i=1:L}; \\ &b' \leftarrow A(history, C) : \text{output is 1 if } b == b' \end{aligned} \tag{3.4}$$

Encryption E is IND-CPA if for every probabilistic polynomial time adversary A, there exists a negligible function $negl(\lambda)$ s.t.

$$Pr[PubK^{CPA}_{A,E}(\lambda) = 1] < \frac{1}{2} + negl(\lambda) \tag{3.5}$$

or equivalently [100],

$$|Pr[output(PubK^{CPA}_{A,E}(\lambda, 0)) = 0] - Pr[output(PubK^{CPA}_{A,E}(\lambda, 1)) = 0]| < negl(\lambda) \tag{3.6}$$

where $PubK^{CPA}_{A,E}(\lambda, b)$ indicates the output of IND-CPA experiment when the challenge bit is b.

(N,N)-Threshold Additive Homomorphic Encryption Scheme with a Distributed Key Generation: An additive homomorphic encryption scheme is a public key encryption scheme with key generation Gen, encryption Enc, and decryption Dec algorithms which additionally enables computation over plaintexts using ciphertexts. That is, for any two messages m_0, m_1 (from the message space) encrypted as $c_0 = Enc_{pk}(m_0), c_1 = Enc_{pk}(m_1)$, one can compute the summation of messages in the

encrypted format as $Enc_{pk}(m_0 + m_1) = c_0 \odot c_1$ where pk is the encryption key and $\odot$ is the homomorphic operation over the ciphertext.

An example is Paillier encryption in which multiplication of the ciphertexts results in the summation of the underlying plaintexts i.e., $c_0 \cdot c_1 = Enc_{pk}(m_0 + m_1)$. In the (N, N)-threshold homomorphic encryption, the decryption key is distributed among N parties such that the presence of all of them (N out of N) is required to make correct decryption. Generation of the key in the threshold settings usually relies on a trusted party who creates and distributes the decryption key shares and then leaves the system. However, in the present work, we leverage a distributed key generation protocol proposed by [101] for the Paillier setting. We refer to that encryption scheme by $TEnc = (DKeyGen, Dsk, Enc, DDec)$ whose details come next.

1. $pk \leftarrow DKeyGen(1^\lambda)$ is a distributed protocol run by N parties $S_1, ..., S_N$ to compute the public key pk as a composite modulus N with an unknown prime factorization $q_1.q_2$.

2. $dk_1, ...dk_N \leftarrow Dsk(pk)$ is a distributed protocol run by $S_1, ...S_N$ to generate decryption key shares of the given public key pk. Each party S_i receives one share i.e., dk_i.

3. $C \leftarrow Enc_{pk}(m, r)$ is the encryption algorithm to convert the plaintext m to a ciphertext C. r is a random number from Z_N^*. The ciphertext is set to $C = g^m \cdot r^N \bmod N^2$ where $g = 1 + N$. When we write $C \leftarrow Enc_{pk}(m)$ we mean the randomness r is generated inside the encryption algorithm.

   Additionally, Paillier encryption comes with re-encryption algorithm to re-randomize a given ciphertext C i.e., $C' \leftarrow ReEnc(C, r') = C \cdot r'^N \bmod N^2$ where $r' \in Z_N^*$. We may eliminate r' and write $ReEnc(C)$ for shorthand.

4. $m = DDec(dk_1, ..., dk_N, C)$ is a distributed protocol in which all the parties $S_1, ..., S_N$ contribute their respective shares of the decryption key i.e., $dk_1, ..., dk_N$ to decrypt a ciphertext C to the plaintext m.

We instantiate an (N,N)-threshold encryption scheme $TEnc$ using the proposal of [101, 102]. We present the security guarantees of $TEnc$ as an ideal functionality F_{THRESH} as shown in Figure 3.2 (generalizing two-party definitions of [102]).

Zero-Knowledge Proof (ZKP) of Knowledge: is a proof system $< P, V >$ for a language L defined over relation R i.e., $L = \{x| \exists\, \omega : (x, \omega) \in R\}$ by which a prover P knowing witness ω can prove the validity of a statement i.e., $x \in L$ to a verifier V. Let $(P(\omega), V(z, r))(x)$ be the output of V (namely, 1 if V accepts the proof, 0 otherwise) in interaction with P upon the common public statement x. The verifier holds the auxiliary input z and the random tape r whereas P owns the private witness ω. A zero-knowledge proof system satisfies three properties which are abstracted as follows [103]:

- Perfect Completeness: an honest prover can always convince the honest verifier on a valid statement $x \in L$. In principle, for every $(x, \omega) \in R$

$$Pr[(P(\omega), V)(x) = 1] = 1 \tag{3.7}$$

- Computational Soundness: A dishonest prover is unable to make a valid proof for an invalid statement $x \notin L$ unless with a low probability. That is, $\forall (x, \omega) \notin R$ and for all dishonest PPT prover P^*,

$$Pr[(P^*(\omega), V)(x) = 1] = 2^{-t} \tag{3.8}$$

  2^{-t} is called soundness error and can get arbitrarily small for the large values of t.

- Computational Zero-knowledge: The proof system does not reveal anything beyond the correctness of the statement $x \in L$. More formally, a proof system (P, V) is computational Zero-knowledge if there exists a PPT simulator Sim s.t. for every PPT verifier V^* we have

$$\{(P(w), V^*(z, r))(x)\} \stackrel{c}{\equiv} \{(Sim^{V^*(x,z,r,.))}\} \tag{3.9}$$

$\mathbf{F}_{THRES}$

Key Generation: Upon receiving request $(Generate, 1^\lambda)$ from party P_i $(i \in [1, N])$, F_{THRES} records $(P_i, Generate, 1^\lambda)$ in the database and sends it to the adversary. Upon the receipt of request $(P_i, Generate, 1^\lambda)$ from all the N parties $(\forall i \in [1, N])$, F_{THRES} sends (RandInput) to the adversary and receives (GenInput, r). Then, F_{THRES} uses its randomness together with r to generate an encryption key pk and its corresponding decryption key dk. F_{THRES} records dk and outputs pk to the adversary. If the adversary responds with *continue* then F_{THRES} delivers pk to all the other parties and ignores any message of this form, otherwise, sends *abort* to all the other parties.

Decryption: Once $(Decrypt, c)$ message is received from some party P_i $(i \in [1, N])$, F_{THRES} operates as below:

1. If no key was created, F_{THRES} ignores the request.

2. If a key was already created, then F_{THRES} records $(P_i, Decrypt, c)$ and sends it to the adversary. Once requests for the decryption of c are received from all parties $P_{i=1:N}$ then F_{THRES} sends $(Decrypt, c)$ to the adversary. Adversary responds with the receiver set $RC \subseteq \{1, ..., N\}$. If RC is empty, F_{THRES} sends *abort* to all the parties. Otherwise, F_{THRES} decrypts the ciphertext c as $Dec_{dk}(c)$ and sends the result to the parties specified in RC.

Figure 3.2: The ideal functionality F_{THRES} for the distributed (N, N)-threshold encryption scheme $TEnc = (DKeyGen, Dsk, Enc, DDec)$. F_{THRES} captures the security properties of Key Generation (i.e., $DKeyGen$ and Dsk) and Decryption ($DDec$) which are multi-party protocols. However, Enc is a single party algorithm (not a multi-party protocol) and thus is not presented as a functionality.

F^R_{POPR} with a relation R

- F_{POPR-R} receives (Prove, P, V, (pk, $\boxed{m}$, $Range$), ω) from prover P and hands over (Prove, P, V, (pk, $\boxed{m}$, $Range$)) to the adversary.
- If the adversary responds with *abort*, then F^R_{POPR} sends *abort* to the verifier V.
- If the adversary responds with *continue*, If ((pk, $\boxed{m}$, $Range$), ω)$\in R$, F^R_{POPR} sends (proof, P, V, (pk, $\boxed{m}$, $Range$)) to the verifier V. Otherwise, outputs (disproof, P, V, (pk, $\boxed{m}$, $Range$)) to V.

Figure 3.3: The ideal functionality F^R_{POPR} for proof of plaintext range.

$Sim^{V^*(x,z,r,.)}$ indicates the output of simulator with oracle access to $V^*(x, z, r, .)$.

We refer to an *interactive proof system* as the proof that is an interactive two-party protocol run between the polynomial-time prover and the verifier. On the other hand, in a *non-Interactive proof*, the proof generated by the prover can be verified without further interaction with the prover.

Non-Interactive Zero-Knowledge Proof of Plaintext Range (NI-POPR): This is a proof system to prove that a ciphertext $\boxed{m}$ encrypts a value of a particular range $Range$ i.e., $m \in Range$. We illustrate a non-interactive version of such POPR with NI-POPR($\boxed{m}, R$).

We present functionality F^R_{POPR}, as shown in Figure 3.3, to capture the security requirements of an ideal $POPR$ protocol.

In section 3.5, we will make use of $POPR$ for ciphertexts embodying 0 or 1 values. Hence, we deploy the proof system proposed by [104] over the following relation R (Equation 3.10). μ belongs to Z^*_N and is the randomness used for the encryption of

F^R_{POCM} with a relation R

- F^R_{POCM} receives (Prove, P, V, (pk, $\boxed{a}$, $\boxed{b}$, $\boxed{c}$), ω) from prover P and hands over (Prove, P, V, (pk, $\boxed{a}$, $\boxed{b}$, $\boxed{c}$)) to the adversary.
- If the adversary responds with *abort*, then F^R_{POCM} sends *abort* to the verifier V.
- If the adversary responds with *continue*, if ((pk, $\boxed{a}$, $\boxed{b}$, $\boxed{c}$), ω)$\in$ R, F^R_{POCM} sends (proof, P, V, (pk, $\boxed{a}$, $\boxed{b}$, $\boxed{c}$)) to the verifier V. Otherwise, outputs (disproof, P, V, (pk, $\boxed{a}$, $\boxed{b}$, $\boxed{c}$)) to V.

Figure 3.4: The ideal functionality F^R_{POCM} for the proof of correct multiplication.

m:

$$R = \{((pk, \boxed{m}, Range), \omega =< m, \mu >) |\ m \in Range \wedge \boxed{m} = Enc_{pk}(m, \mu)\} \tag{3.10}$$

Non-Interactive Zero-Knowledge Proof of Correct Multiplication (NI-POCM): The purpose of POCM is to prove that a ciphertext is the correct multiplication of two given ciphertexts under a given public key pk. Namely, $\boxed{c} = \boxed{a * b}$ where $\boxed{a}$ $\boxed{b}$ and $\boxed{c}$ are given. We illustrate a non-interactive version of such POCM with NI-POCM($\boxed{a}$, $\boxed{b}$, $\boxed{c}$). The ideal function F^R_{POCM}, presented in Figure 3.4, captures soundness, completeness and zero-knowledge properties of a secure $POCM$ protocol.

We base our POCM on the technique presented in [79] for Paillier encryption setting. The corresponding relation R is formulated in equation 3.11. μ and γ are elements of Z^*_N.

$$R = \{(pk, \boxed{a}, \boxed{b}, \boxed{c}), \omega =< a, \mu, \gamma >) |\ \boxed{a} = Enc_{pk}(a, \mu) \wedge \boxed{c} = ReEnc(\boxed{b}^a, \gamma)\} \tag{3.11}$$

We further leverage Fiat-Shamir method [105] to achieve perfect zero-knowledge and a non-interactive POCM in random oracle model.

Mix Network: Mix network (mix-net) is a multiparty system which converts a set of input data to an untraceable output [106]. The input is usually a set of ciphertexts i.e., $\overleftarrow{C}_{in} = \{C_1, ..., C_k\}$ and the output is the re-encrypted and permuted version of the input i.e., $\overleftarrow{C}_{out} = \{C'_{\pi(1)}, ..., C'_{\pi(k)}\}$ where π is a permutation and $C'_{\pi(i)}$ denotes the re-encryption of $\pi(i)^{th}$ element of C_{in}. A mix-net comprises multiple servers (mixers) $\{S_1, ..., S_N\}$ where each S_i in turn computes a randomly permuted and re-encrypted output C_{out_i} from $C_{out_{i-1}}$.

Verifiable Shuffles are used to implement mixers. A verifiable shuffle VS is a tuple $VS = (E, SH, (P, V))$ where $E = (Gen, Enc, Dec, ReEnc)$ is an encryption scheme with key generation Gen, encryption Enc, decryption Dec and re-encryption $ReEnc$ algorithms. SH denotes shuffle algorithm whose input is a set of ciphertexts i.e., $\overleftarrow{C}_{in} = \{\boxed{m_1}, ..., \boxed{m_k}\}$ and the output is the re-encrypted and permuted version of the input i.e., $\overleftarrow{C}_{out} = \{\boxed{m_{\pi(1)}}, ..., \boxed{m_{\pi(k)}}\}$. (P, V) is a proof system used to prove that there exists a permutation π and some randomnesses which can covert the input ciphers to the output ciphers.

A verifiable shuffle should satisfy the following properties: 1) *shuffle privacy* that is the permutation must remain secret to any outsider (this features usually relies on the IND-CPA security of the underlying encryption E) and 2) *shuffle verifiability* which means the correct construction of the output should be verifiable (that relies on the soundness of the proof system). The shuffle verifiability guarantees the robustness of a mix-net even when some number of mixers are corrupted. We define the ideal functionality F^R_{VS} (Figure 3.5) to capture the security properties of a verifiable shuffle proof system. $\overleftarrow{C}_{out}$ is the correct permutation of $\overleftarrow{C}_{in}$ if the prover P knows a witness ω for which $(\overleftarrow{C}_{in}, \overleftarrow{C}_{out}, w) \in R$.

We employ the Pallier-based mix-net protocol proposed by [107]. Their underlying shuffle scheme has an interactive proof system which is transformable to a non-interactive version using Fiat-Shamir method [105] in random oracle model. R

F_{VS}^{R} with a relation R

- F_{VS}^{R} receives (Prove, P, V, (pk, $\overleftarrow{C}_{in}$, $\overleftarrow{C}_{out}$), ω) from prover P and hands over (Prove, P, V, (pk, $\overleftarrow{C}_{in}$, $\overleftarrow{C}_{out}$)) to the adversary.
- If the adversary responds with *abort*, then F_{VS}^{R} sends *abort* to the verifier V.
- If the adversary responds with *continue*, if ((pk, $\overleftarrow{C}_{in}$, $\overleftarrow{C}_{out}$), ω)$\in R$, F_{VS}^{R} sends (proof, P, V, (pk, $\overleftarrow{C}_{in}$, $\overleftarrow{C}_{out}$)) to verifier V. Otherwise, outputs (disproof, P, V, (pk, $\overleftarrow{C}_{in}$, $\overleftarrow{C}_{out}$)) to V.

Figure 3.5: The ideal functionality F_{VS}^{R} for the proof system of a verifiable shuffle scheme.

is defined as in Equation 3.12. π is the permutation, $\overleftarrow{C}_{in} = \{\boxed{m_1}, ..., \boxed{m_k}\}$ and $\overleftarrow{C}_{out} = \{C_1, ..., C_k\}$.

$$R = \{(pk, \overleftarrow{C}_{in}, \overleftarrow{C}_{out}), \omega =< \pi, \{\mu_i\}_{i=1}^{k} >) | \ \forall C_i \in \overleftarrow{C}_{out},\ C_i = ReEnc(\boxed{m_{\pi(i)}}, \mu_i)\} \tag{3.12}$$

3.5 Privado

3.5.1 Design Challenges

In this section, we list some possible threats and attacks that can be attempted by a malicious adversary in our system model. We also sketch our proposed solution next to each item. The attacks are not limited to this list; our goal is to highlight some of our design difficulties as well as provide an intuition of our design choices.

In Section 4.2, we present a formal security definition capturing user privacy followed by a concrete security proof of Privado. Our formal proof does not impose any strategy on the adversary and considers a black box adversary. This implies that

our advertising system defeats any misbehavior of adversary including the ones listed below.

Profile Replay Attack: Profile replay attack refers to the situation where a malicious server attempts registering a (corrupted) user inside a group employing the profile of an honest member of that group. That is, the corrupted server re-encrypts the honest user's profile and submits it as a new one. Note that due to the IND-CPA security of the encryption scheme, the honest servers would not notice such profile duplication. This duplication influences the pattern in the group matching result (i.e., the total number of matched users in each group). For clarification, assume a group of size k with 2 honest members and $k-2$ dishonest ones. Let P_1 and P_2 denote the profiles of honest members. One of the corrupted group members submits a duplicate of P_1 and the rest of $k-3$ corrupted members leave their profiles empty (without any attributes). Therefore, the matching result of P_1 is always counted twice in the group matching result. Thus, if P_1 matches an advertising request then the total number of target users would be a value larger than 2, otherwise not. This indeed enables a corrupted server to link the result of matching to a particular member (the genuine holder of P_1 in the above example).

Privado defeats this attack by requiring each user to prove in zero-knowledge that he knows the content of the submitted encrypted profile. Consequently, the adversary cannot make valid proof of an encrypted profile with unknown content. The proof includes *proof of plaintext range* as well as *proof of correct multiplication*. We supply the details in Section 3.5.3.

Compound Group Matching: Compound group matching attack occurs when a corrupted server does not commit to the initial grouping of users, i.e., for each advertising request, it groups profiles in an arbitrary way. This enables an adversary to deliberately group a victim profile with multiple different groups and according to the changes in the matching results (before and after inclusion of the victim profile) learns which advertising requests match the profile.

Privado stands this attack by employing an (N, N)-threshold encryption scheme

whose decryption key is divided among N servers each run by an independent provider. Servers are all aware of the initial grouping of the users. Also, at least one of the servers is non-colluding by assumption, i.e., would not contribute its decryption power with other servers for fake groups. Thus, compound group matching, which relies on the decryption operation, would be impossible.

Servers Equivocation: This attack refers to any deviation of the servers from the execution of instructed protocols. Our approach stands against the attack using replicated computations. That is, multiple servers run an identical set of computations and they shall end up with the same local results. In the case of inconsistency, an equivocation is detected. We make use of additive homomorphic encryption (which is secure against malicious entities) to devise the servers computations.

Privado vs PPAD: Technically, because we have a harder problem compared to PPAD [67], our solution employs many other tools, including threshold additive homomorphic encryption, zero knowledge proofs of knowledge, verifiable shuffles, and mix networks.

3.5.2 Construction Overview

In this part, we provide an overview of Privado's protocols and their objectives. Protocols are **Initialization**, **User Registration**, **Advertiser Registration**, and **Advertising**. The detailed construction is provided in section 3.5.3.

Initialization: The system life-cycle starts by servers running the initialization protocol to set up the necessary protocol parameters. This includes a threshold homomorphic encryption scheme whose decryption key is shared among the servers. The key generation is a distributed protocol without making use of any trusted third party. Each server also utilizes a database to keep a copy of every profile and advertising request.

User Registration: A user joins the system by executing the user registration protocol. Each user is assigned to a particular group at his arrival time and obtains a group membership identifier (MID) that uniquely identifies him among his group-

mates. The membership ID assignment should be at random and private; otherwise user privacy can be violated (see Section 4.2). Servers distribute MIDs among the group members privately and randomly by the help of a mix-net protocol. MIDs shall help in the calculation of the group matching results. Upon the receipt of MID, the user creates his encrypted profile. Each profile is comprised of a set of attributes which are modeled by a Bloom filter. The final encrypted profile comprises element-wise encryption of the Bloom filter (in which the MID is also integrated). Additionally, the user proves in zero knowledge that the profile was constructed properly.

Advertiser Registration: Similar to the profile, advertising request consists of a set of attributes that shall be converted to a Bloom filter format. The advertiser submits his request to all the servers. The servers insert the request into their local databases.

Advertising: Servers run advertising protocol to determine the target groups for a given advertising request. Recall that a target group is the one in which the total number of matched profiles (i.e., target users) exceeds a system-wide threshold. In a nutshell, the advertising protocol consists of two parts: *Aggregation* and *Matching*. During the aggregation phase, each server locally computes the matching results of individual members in the encrypted format and then aggregates them into a single ciphertext. Aggregation helps break any linkability between the matching results and individual group members. Next, the servers collaborate to decrypt the aggregate by running a threshold decryption protocol. Each server de-aggregates the aggregate value (that is now in plaintext) to identify the total number of profiles that match to the advertising request.

3.5.3 Full Construction

This section presents the detailed construction of Privado. We assume that the servers have synchronized states. However, the synchronization of servers does not serve any privacy purpose, that is, the lack of synchronization would not cause any privacy issue.

Initialization:

Initialization protocol is run by N servers to set the system parameters as follows.

- Servers initially agree on protocol parameters and publish them publicly. This includes the group size k, the threshold value Thr which is used for the group matching, the Bloom filter size p and its hash functions. Also, servers construct a super-increasing set $\Delta = \{\delta_1, ..., \delta_k\}$ of size k called *Membership Identifier Set* whose elements satisfy Equation 3.13.

$$\forall m \in \{1, ..., k\}, \delta_m > \sum_{i=1}^{m-1} \delta_i * p \tag{3.13}$$

  where p is the size of the Bloom filter. Elements of Δ are used in the user registration protocol.

  The maximum number of membership identifiers δ_k is a function of the largest data that gets encrypted in our design. This data is the aggregate of users profiles in each group i.e., ϕ that is bounded by $\sum_{j=1}^{k} \delta_j \cdot p$ (see Equation 3.18). Thus, we can increase the value of k till ϕ does not exceed the Paillier encryption message space i.e., Z_N. In Privado, N is 2048 bits length and the Bloom filter size p is 6848 bits (to accommodate 437 many attributes [67]). Using the above formulation and numbers, we examine different values of k and find out that the largest value is 161.

- Servers jointly establish a distributed (N,N)-threshold additive homomorphic encryption scheme $TEnc = (DKeyGen, Dsk, Enc, DDec)$. As such, servers run $DKeyGen$ to generate an encryption key pk which shall be publicized to the whole system. Servers engage in the execution of Dsk protocol to create the shares of the corresponding decryption key. As a result, every (i^{th}) server obtains an additive share of the decryption key ($dk_{i=1:N}$) which keeps it private. Correct decryption requires all servers contributing their decryption key shares. Henceforth, for shorthand, we write Enc to denote encryption under pk.

- One of the servers, namely S_j, encrypts Δ as $\boxed{\Delta} = (\boxed{\delta_1} = Enc(\delta_1, r_1), ..., \boxed{\delta_k} = Enc(\delta_k, r_k))$ using the randomnesses $r_{i=1:k}$. S_j communicates $\boxed{\Delta}$ together with the randomnesses $r_{i=1:k}$ to all the other severs $S_{i=1:N, i\neq j}$. Each server $S_{i=1:N}$ recomputes the encryptions, namely, for $i \in [1, k]$ computes $Enc(\delta_i, r_i)$ and compares against $\boxed{\delta_i} \in \Delta$. Servers abort in the case of mismatch, otherwise, store $\boxed{\Delta}$ in their local databases. As we will present in the user registration part, $\boxed{\Delta}$ shall be used as the input to the mix-net.

User Registration:

Figure 3.6 depicts user registration protocol by which the user registers his profile to N servers. User and servers interact through an authenticated channel as given below.

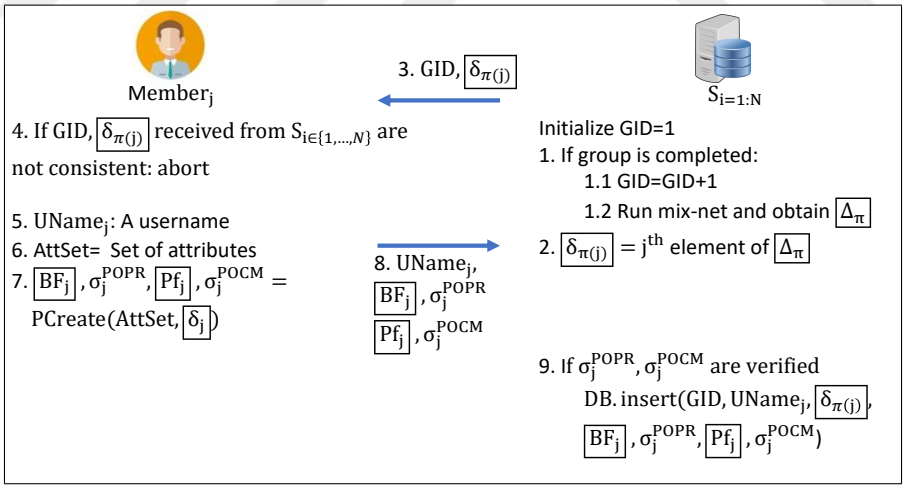

Figure 3.6: An instance of User Registration protocol (UReg) between jth group member ($j \in [1, k]$) and ith server ($i = 1 : N$). When a group of profiles is registered (step 1), servers increment the GID as well as jointly run mix-net to make a new shuffle of the membership identifier set (step 1.2). The user obtains the group information (at step 3) from all the servers and delivers the profile (at step 8) to all of them.

Servers:

- *Group assignment:* Servers decide on the user's group identifier i.e., GID which determines the group the user belongs to (step 1.1 of Figure 3.6). The group

assignment must be at random and can rely on users arrival order. Servers assign GIDs to the users incrementally. Namely, servers assign $GID = 1$ to the first set of k registered users, and $GID = 2$ for the second set of k registered users and so on. For this sake, each server keeps track of the number of joining users.

- *Membership ID assignment:* Next, servers assign a private integer called *membership identifier* (for shorthand, membership ID denoted by δ) to each user that uniquely identifies him inside his group (step 2 of Figure 3.6). Since the uniqueness of δs must be preserved within each group, servers only create one set of identifiers i.e., Δ in the initialization phase, and keep assigning the same identifiers for every group but under different permutations.

  For every group of k users, servers shuffle the encrypted membership identifier set i.e., $\boxed{\Delta}$ ($\boxed{\Delta}$ is generated in the initialization phase) through a mix-net execution (step 1.2 of Figure 3.6). We remark that **mix-net is only run once for each group**. Let $\boxed{\Delta_\pi} = (\boxed{\delta_{\pi(1)}}, ..., \boxed{\delta_{\pi(k)}})$ be the output of mix-net where π indicates the permutation. Each server $S_{i=1:N}$ stores $\boxed{\Delta_\pi}$ in its local database. For the jth member of the group, each server $S_{i=1:N}$ delivers the jth element of $\boxed{\Delta_\pi}$ (i.e., $\boxed{\delta_{\pi(j)}}$) to that user (step 2 of Figure 3.6). Note that since mix-net preserves shuffle privacy, none of the servers knows the permutation π and hence does not know which δ is given to which group member. Thus, the private assignment of membership identifier is satisfied. Additionally, shuffle verifiability of mix-net guarantees that $\boxed{\Delta_\pi}$ is the correct permutation of $\boxed{\Delta}$ (even in the presence of $N - 1$ corrupted servers).

- Each server $S_{i=1:N}$ sends GID and the membership identifier $\boxed{\delta_{\pi(j)}}$ to the user [3] (step 3 of Figure 3.6).

User:

- The user obtains GID as well as $\boxed{\delta_{\pi(j)}}$ from each server $S_{i=1:N}$ (step 3 of Figure

3.6). If the GID and the ciphertext $\boxed{\delta_{\pi(j)}}$ sent by all the servers are identical (step 4 of Figure 3.6), user continues to the next step. Otherwise, an equivocation from the server side is detected and user aborts.

- *Profile creation and submission:* The procedure of profile creation is shown in Algorithm.1. The user inserts his set of attributes $AttSet$ into a Bloom filter data structure (line 1 of Algorithm.1). Then, the user multiplies each element of BF with $\boxed{\delta_{\pi(j)}}$ and re-encrypts the result (line 2 of Algorithm.1). The resultant vector $\boxed{Pf_j}$ constitutes the profile of user. Additionally, the user must create a proof asserting that the profile is well-formed. Namely, the element of $\boxed{Pf_j}$ are either encryption of zero or the assigned $\boxed{\delta_{\pi(j)}}$. As such, user performs the following.

  1. The user encrypts his Bloom filter (line 3 of Algorithm.1). Then, for every element of $\boxed{BF_j}$, he creates a proof of plaintext range $[0, 1]$ (line 4 of Algorithm.1).
  2. Using proof of correct multiplication, the user proves that every element of $\boxed{Pf_j}$ is the correct multiplication of the corresponding element in $\boxed{BF_j}$ with $\boxed{\delta_{\pi(j)}}$ i.e., $\boxed{Pf_{j,i}} = \boxed{b_{j,i} \cdot \delta_{\pi(j)}}$ (line 5 of Algorithm.1).

- Finally, the user submits $\boxed{BF_j}, \boxed{Pf_j}$ together with the proofs $\sigma_j^{POPR}, \sigma_j^{POCM}$ to all servers (steps 7-8 of Figure 3.6).

Servers:

- Each server $S_{i=1:N}$ verifies the proofs $\sigma_j^{POPR}, \sigma_j^{POCM}$ and accepts or rejects accordingly (step 9 of Figure 3.6). If the verification is successful, servers insert the profile together with its GID, and $\boxed{\delta_{\pi(j)}}$ into their local databases.

Advertisement Registration:

This protocol, given in Figure 3.7, is run between the advertiser and all the servers to register an advertising request to the system. They interact as follows.

Algorithm 1 PCreate(AttSet, $\boxed{\delta_{\pi(j)}}$)

1: $BF_j = \{b_{j,i}\}_{i=1:p} = BFCreate(AttSet)$
2: $\boxed{Pf_j} = \{\boxed{Pf_{j,i}}\} = \{ReEnc(b_{j,i} * \boxed{\delta_{\pi(j)}})\}_{i=1:p}$
3: $\boxed{BF_j} = \{\boxed{b_{j,i}}\}_{i=1:p} = \{Enc(b_{j,i})\}_{i=1:p}$
4: $\sigma_j^{POPR} = \{\text{NI-POPR}(\boxed{b_{j,i}}, \{0,1\})\}_{i=1:p}$
5: $\sigma_j^{POCM} = \{\text{NI-POCM}(\boxed{b_{j,i}}, \boxed{\delta_{\pi(j)}}, \boxed{Pf_{j,i}})\}_{i=1:p}$
6: **return** $\boxed{BF_j}, \sigma_j^{POPR}, \boxed{Pf_j}, \sigma_j^{POCM}$

Advertiser:

- Advertiser creates a Bloom filter from the set of attributes (denoted by $TAud$) he seeks in his target users (steps 1-3). The target audience of the advertising request are the profiles which contain the conjunction of attributes in $TAud$. Thus, if an advertiser wants to find users with attributes X **OR** Y, he must split the request and submit it as two separate requests: one for X and the other for Y.

- Let $Req = \{r_1, r_2, ..., r_p\}$ be the created Bloom filter (p is the size of Bloom filter). The advertiser hands over Req as well as the product advertisement (denoted by $Product$) to each server $S_{i=1:N}$ (step 4).

Servers:

- Servers assign a request number RID to the advertising request (step 5) and return it to the advertiser (step 7). RIDs are assigned incrementally, thus, servers keep track of the registered requests. RID shall be used to follow up the advertising result.

- Each server $S_{i=1:N}$ inserts the advertising request, the product advertisement, and the request number RID in its local database (step 6).

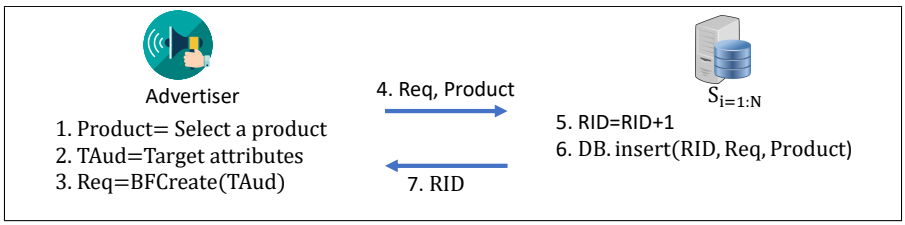

Figure 3.7: Advertiser registration protocol (AdReg). The advertiser submits the request to all the servers and gets an identical RID from all of them (we assume servers have synchronized states).

Advertising

Servers run this protocol for every unmatched pair of advertising request and a group of profiles, and store the matching result in their databases. Let GID and Req denote the group and the request that are to be matched, respectively. Figure 3.8 exhibits the overall interaction of servers for the advertising protocol. First, each server retrieves the profiles of the intended group, i.e., $Pf_1, ..., Pf_k$ and the request from its database (steps 1-2 of Figure 3.8). Next, servers proceed with three main phases 1) Aggregation, 2) Distributed Decryption, and 3) De-aggregation/Matching. While phases 1 and 3 are non-interactive, phase 2 requires servers interaction to run distributed decryption protocol. We stress that **both users and advertisers are offline during the matching procedure**, which is one of our main contributions.

1. *Aggregation:* Aggregation (step 4 of Figure 3.8) is run by each server locally. Algorithm 2 summarizes the entire procedure. Let R be the indices of the set bits in the advertising request as given in Equation 3.14.

$$R = \{i | r_i \in Req \text{ and } r_i == 1\} \tag{3.14}$$

For each profile $Pf_{j=1:k}$ in the group, each server extracts the profile's elements in accordance to the index set R and then homomorphically sums them up (step

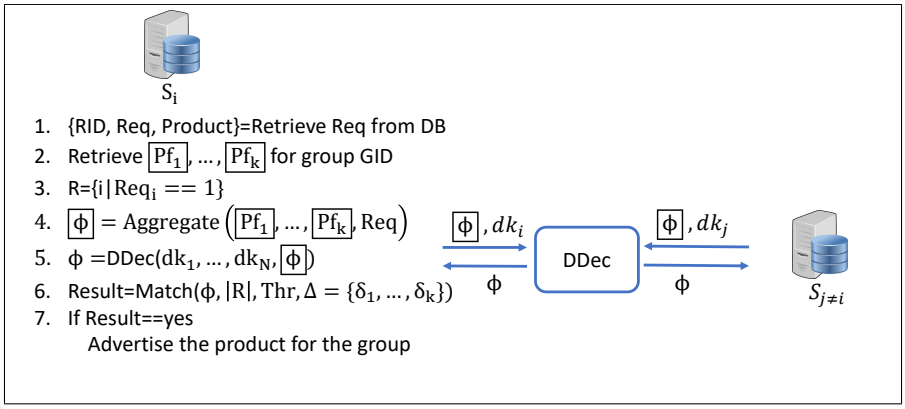

Figure 3.8: Advertisement protocol (Ad) to match an advertising request (Req) to a group (GID). S_i communicates with all the other servers at step 5 to execute distributed decryption DDec protocol. All the servers run an identical set of instructions as indicated for S_i.

3 of Algorithm 2). This is indicated in Equation 3.15.

$$\boxed{\phi_j} = \prod_{\substack{i=1:p \\ i \in R}} \boxed{Pf_{j_i}} = \boxed{\sum_{\substack{i=1:p \\ i \in R}} b_{j_i} \cdot \delta_{\pi(j)}} = \boxed{\delta_{\pi(j)} \cdot \sum_{\substack{i=1:p \\ i \in R}} b_{j_i}} \tag{3.15}$$

Pf_{j_i} represents the i^{th} element of the j^{th} profile where $i \in [1, p]$ and $j \in [1, k]$. Observe that the final summation i.e., $\boxed{\phi_j}$ embodies the multiplication of the user's membership ID i.e., $\delta_{\pi(j)}$ with $\sum_{i \in R} b_{j_i}$ that is the sum of bit values of the profile in accordance to the set R. Let α_j denote $\sum_{i \in R} b_{j_i}$ (Equation 3.16).

$$\alpha_j = \sum_{\substack{i=1:p \\ i \in R}} b_{j_i} \tag{3.16}$$

Thus, $\boxed{\phi_j}$ would be

$$\boxed{\phi_j} = \boxed{\delta_{\pi(j)} \cdot \alpha_j} \tag{3.17}$$

After the computation of $< \boxed{\phi_1}, ..., \boxed{\phi_k} >$, each server aggregates them as given in Equation 3.18 (line 5 of Algorithm 2). Let $\boxed{\phi}$ be the final aggregate. Since all the servers hold a database with the same set of encrypted profiles, their local computation should all result in the same ciphertext $\boxed{\phi}$. Otherwise, an

equivocation is detected[4].

$$\boxed{\phi} = \prod_{j=1}^{k} \boxed{\phi_j} = \boxed{\sum_{j=1}^{k} \delta_{\pi(j)} \cdot \alpha_j} \tag{3.18}$$

Algorithm 2 Aggregate($\boxed{Pf_1}, ..., \boxed{Pf_k}, Req$)

1:
2: **for** $j = 1 : k$ **do**
3: $\quad \boxed{\phi_j} = \prod_{\substack{i=1:p \\ i \in R}} \boxed{Pf_{j_i}}$
4: **end for**
5: $\boxed{\phi} = \prod_{j=1}^{k} \boxed{\phi_j}$
6: **return** $\boxed{\phi}$

2. *Distributed Decryption :* All the servers use their decryption key shares to jointly decrypt $\boxed{\phi}$ through $DDec$ protocol (step 5 of Figure 3.8). We use ϕ to be the decrypted result as shown in Equation 3.19.

$$DDec(\phi) = \phi = \sum_{j=1}^{k} \delta_{\pi(j)} \cdot \alpha_j \tag{3.19}$$

Recall that we employ a $DDec$ algorithm which stands malicious parties. As such, each server accompanies its computation result with a zero-knowledge proof of correct decryption [101]. Thus, if a corrupted server attempts decrypting a ciphertext different from $\boxed{\phi}$, or using a fake decryption key share then its proof will not be validated by honest servers and the cheating server is caught.

3. *De-aggregation/Matching:* Matching procedure is a non-interactive protocol that each server runs individually (step 6 of Figure 3.8). Algorithm 3 illustrates this procedure where the aim is to identify the total number of targeted profiles from Φ. It starts by dividing ϕ with the first largest membership ID namely, δ_k (line 3). The quotient of this division is equal to $\alpha_{\pi^{-1}(k)}$ where $\pi^{-1}(k)$ indicates the index of a group member with the membership ID δ_k. To realize

why this is the case, let reformulate ϕ by extracting out the term of δ_k (to be the largest δ) as in Equation 3.20.

$$\frac{\phi}{\delta_k} = \frac{\delta_k \cdot \alpha_{\pi^{-1}(k)} + \sum_{m=1}^{k-1} \delta_m \cdot \alpha_{\pi^{-1}(m)}}{\delta_k} \tag{3.20}$$

Having known that $\alpha_{j=1:k}$ values are bounded by p together with Equation 3.13, we derive the following inequalities:

$$\sum_{m=1}^{k-1} \delta_m \cdot \alpha_{\pi^{-1}(m)} < \sum_{m=1}^{k-1} \delta_m \cdot p < \delta_k \tag{3.21}$$

Relying on Equation 3.21, the quotient of division in Equation 3.20 would be $\alpha_{\pi^{-1}(k)}$ and the remainder is $\sum_{m=1}^{k-1} \delta_m \cdot \alpha_{\pi^{-1}(m)}$.

If the quotient of division, i.e., $\alpha_{\pi^{-1}(k)}$ equals to $|R|$ (line 3) then one match is found in the group (line 4). This is true as explained next. First note that $\alpha_{\pi^{-1}(k)}$ contains the sum of bit values of the Bloom filter of the $\pi^{-1}(k)^{th}$ group member in accordance to the index set R (also see Equation 3.16). Hence, equality of $|R|$ and $\alpha_{\pi^{-1}(k)}$ indicates that the number of the set bits in the request (i.e., $|R|$) and the sum of the corresponding bits in a profile i.e., $\alpha_{\pi^{-1}(k)}$ are equal. This happens only when a profile contains all the attributes in the advertising request (i.e., every element of the profile corresponding to the non-zero elements of the request is also non-zero).

The procedure continues by setting the value of ϕ to the remainder of the division (line 6) and using the next largest δ as the divisor (line 2). At each step, the matching of one group member is identified and augmented to the total matches (line 4). If the number of matched users hits the threshold (line 8), then the advertisement is served for the entire group, otherwise, the group is skipped (lines 9-11).

Algorithm 3 $Match(\phi, |R|, Thr, \Delta = \{\delta_1, ..., \delta_k\})$

```
count = 0
for δ_j ∈ Δ, j = k : 1 do
    if Φ/δ_j == |R| then
        count = count + 1
    end if
    φ = φ  mod  δ_j
end for
if Thr ≤ count then return Yes
else
    return No
end if
```

Profile update: The profile update corresponds to replacing the existing profile with a new one. The procedure is identical to the profile registration except that servers do not redistribute membership identifiers and the updater uses his prior identifier and remains in the same group. In fact, the user interacts with servers by following only the steps 5-9 of Figure 3.6.

Performing profile update in a group-based advertising approach comes with the security concern [67], where servers can analyze the group matching results before and after the update operation and hence identify which attributes are modified in the updated profile. This leakage is independent of Privado's design. To demonstrate this independency, we refer back to the ideal group-based advertising functionality $F(AttSet_1, ..., AttSet_k, TAud)$ (as we discussed in Section 3.1). In that definition, F receives users profiles (set of attributes) directly from the users and not through the OSN servers. The OSN servers only query F using arbitrary advertising requests. F shall return the number of matches to all the servers. Now, assume that the OSN servers have queried F over $TAud =$ ("$music$") for a particular group of users $UName_1, ..., UName_k$ holding $(AttSet_1, ..., AttSet_k)$. Assume that F returns 2 as

the output to the servers; namely, there are two members of the group interested in "music". Later on, a member of that group, holding user-name $UName_k$, updates his attributes to $AttSet'_k$. Note that while the action of update operation under the user-name $UName_k$ is known to the servers, the content of the update (and hence the new attribute set) is not. When the update takes place, the servers again query F over the same group and the same advertising request i.e., $TAud = (\text{"}music\text{"})$. This time, the number of matched users, let's say, increases to 3. This implies that $UName_k$'s update on his set of attributes $AttSet_k$ includes the insertion of "music". As such, in group-based advertising, such linkability happens as soon as individual updates become effective in a real-time fashion, regardless of the underlying solution.

To address this issue, what we propose is to perform *group-based update* which follows the group-based advertising semantics. In group-based update, profile updates are applied when all the group members (of a single group) hand over at least one update. Assume the j^{th} member of the group wants to update his profile. He runs $PCreate$ algorithm (algorithm 1) over a new set of attributes. Let Op_j be the output tuple. User submits Op_j to all the servers. Servers copy this update in their databases but would not replace the old profile with this new one (thus will not consider it for any advertising). Instead, servers wait until there would be an update for every member of that group. Till then, the user may attempt multiple updates and servers would only consider the latest one. Once $Op_1, ..., Op_k$ are received from all the k members of the group, servers replace the all old profiles in that group with the updated ones. Henceforth, advertising is run over the new set of profiles. Notice that from now on any changes that happen to the group matching result (after the group-based update) cannot be linked to a particular group member since all of the group members updated their profiles (and by the minimal assumption, at least two updates were performed by the honest members).

Any solution for the profile update in group-based advertising system should follow the nature of "batch update" or "group update" where profile updates are applied when all the group members (of a single group) hand over at least one update. This

complies with the group-based advertising semantics when we treat all the group members as a unit and wait for a group to get completed to start the advertising. Though this approach may be not efficient in the sense that updates are not applied instantly, it breaks the tie between an update and the user who has performed the update (and stay consistent with the unlinkability definition). In the case that waiting for other group-members' updates is not feasible, an alternative solution (which is more costly) would be that all the group members may resign from the current group, create new accounts and join the system as new users into new groups. To preserve user privacy (unlinkability), such regrouping should happen in such a way that the users' new accounts would not be linkable to their past accounts. Namely, if one user holding $UName_1$ re-registers to the system using $UName_2$, none of the servers should know that these two user-names belong to the same user. Such information about the connection of two different usernames may leak, e.g., by performing timing attacks, which is out of the scope of the current work.

Notes to Chapter 3

1 https://www.statista.com/statistics/346167/facebook-global-dau/

2 https://www.statista.com/statistics/778191/active-facebook-advertisers/

3 In Privado, for the ease of explanation, we assume servers are synchronized. That is, all the requests coming to the system arrive at all the servers simultaneously, and servers' processing speed are identical hence servers attempt execution of a protocol (e.g., mix-net, and distributed decryption protocol) in a synchronized manner. As such, we eliminate the coordination requirement among servers for the mix-net and distributed decryption protocol execution. However, one may consider servers run a Byzantine agreement protocol to agree on a specific protocol over a particular set of inputs to be executed. Or one may designate one of the servers to be the coordinator (similar to Byzantine agreements based on election). Every registration request is organized by the coordinator. For each new group, the coordinator triggers the mix-net execution by sending a message to all the servers. Servers jointly run mix-net and compute $\boxed{\Delta}$. For the registration of each group member, the coordinator asks other servers to hand over the membership identifier directly to the user. Similar to the mix-net execution, the coordinator can lead the execution of distributed decryption protocol as follows. As soon as each server computes the encrypted aggregate result and performs partial decryptions (using its decryption key share), it communicates the result to the coordinator (with some extra information about the group and the advertising request for which the result is computed). Servers can communicate such data with the coordinator asynchronously. Once N partial decryptions for the same group and advertising request are received by the coordinator, it can obtain the plaintext aggregate. The coordinator then can put all the N partial decryption results in one massage and transmit to the rest of the servers. Please note that as we discuss about network overhead in section

4.1.2, servers can exchange their partial decryption results of all the distributed decryption protocols at once and in a batch format. Thus, any timing overhead that the synchronization among the servers may cause will be amortized over all the executions of advertising protocol. Thus such synchronization overhead would not affect the performance of our advertising protocol.

4 By detecting equivocation, we aim to protect user privacy against malicious servers who may attempt performing decryption of arbitrary data (like the individual profiles). Thus, the equivocation detection will enforce the correct execution of protocol at the server-side. Due to the presence of at least one non-colluding server, the equivocating server will gain no information (cannot perform decryption on arbitrary data) to compromise user privacy (as the attack will fail). However, the equivocating server may disable finding the matching result by not contributing its decryption key for decryption of the correct encrypted aggregate value. While we can detect the equivocating server, enforcing the server to makeup its behavior is a different scenario. Any of the servers may step out of the protocol execution and stop the system. This is analogous to the denial of service which we treat as an orthogonal problem to be addressed in the future. However, one may adopt an existing fair exchange protocol like [108] to address such service disability.

Chapter 4

PRIVADO: COMPLEXITY, PERFORMANCE AND SECURITY

4.1 Complexity and Performance

4.1.1 Complexity

We demonstrate the computational complexity based on the number of group multiplications. The complexity analysis are illustrated in Table 4.1.a.

User: User performs $O(p)$ encryption operations to create $\boxed{BF}$ and $\boxed{Pf}$ as part of his profile. Each encryption requires $O(\lambda)$ group multiplications. Further, for each element of $\boxed{BF}$ and $\boxed{Pf}$, the user generates proof of plaintext range and proof of correct multiplication. Each proof is of $O(\lambda)$. Thus, the overall running time complexity of user is $O(p \cdot \lambda)$.

Advertiser: An advertiser only creates a plaintext Bloom filter, hence carries no cryptographic operations.

Servers: We analyze the running time complexity of servers for each protocol separately as follows.

- **User registration**: During this protocol and for each group of profiles, servers run mix-net to create a fresh permutation of the membership identifier set. We use the verifiable shuffle scheme proposed by [107]. As such, in each instance of mix-net, each server carries $O(k \cdot \lambda)$ operations (or equivalently $O(\frac{k \cdot \lambda}{k}) = O(\lambda)$ operations per group member).

  Additionally, servers receive the encrypted profiles whose correctness must be checked. That is, servers verify σ^{POPR} and σ^{POCM} that are generated for p

Overhead\Entity	User	Advertiser	$S_1, ...S_N$
User Registration	$O(p \cdot \lambda)$	-	$O(p \cdot \lambda)$
Advertisement	-	-	$O(k \cdot \|R\| + N \cdot \lambda)$

(a) Privado Asymptotic Performance

Overhead\Entity	User	Advertiser	Provider	PSP
User Registration	$O(p)$	-	-	-
Advertisement	-	-	$O(k \cdot \|R\|)$	$O(k \cdot \|R\|)$

(b) PPAD [67] Asymptotic Performance

Table 4.1: Computation Complexity based on the group multiplications. k: number of users per group. $|R|$: number of set bits in the advertising request. p: size of the Bloom filter. N: the total number of servers. Advertiser registration does not involve any cryptographic operation hence is not included in the tables.

distinct elements of each profile. Verification of each proof incurs constant overhead in λ. Thus, in total, profile verification costs $O(p \cdot \lambda)$ at each server.

In total, $O(p \cdot \lambda)$ is the overhead of profile registration on each server.

- **Advertising**: In this protocol, servers locally compute the $\boxed{\phi}$ value then decrypt it. The computation of $\boxed{\phi}$ requires $(k \cdot |R|) + k - 1$ operations. Then, each server uses its own share of the decryption key to decrypt $\boxed{\phi}$ which is done in $O(\lambda)$. Additionally, each server must verify the computation (decryption) of $N - 1$ other servers which results in $O((N - 1) \cdot \lambda)$ more operations. In total, $O(k \cdot |R| + N \cdot \lambda)$ is a load of advertising on each server. We emphasize that N adds only an additive overhead to the advertising run-time.

4.1.2 Concrete Performance

We investigate the performance of Privado by simulating the advertising protocol on a single computer with an Intel Xeon 2.93 GHz CPU, 80GB RAM, and Ubuntu 16.4 operating system. We deploy Paillier encryption scheme with 2048 bit modulus. Our

performance results are taken using Fiat-Shamir heuristic implementation.

We generate 1000 random profiles with 400 attributes (based on our personal experience from Facebook advertising as well as due to [67], 400 attributes are approximately the maximum number) as well as 100 advertising requests with 30 random attributes (for randomly generated profiles, almost no match is found for an advertisement with more than 30 attributes [67]). A Bloom filter accommodating 400 attributes has a size of $p = 6848$. An advertising request with 30 attributes contains 294 set bits in its Bloom filter representation (i.e., $|R| = 294$). We group the profiles using group sizes 2-20, create their encrypted format and attempt to run advertising protocol over the resultant groups. The results are shown in Figure 4.1.

The results assert that the server's overhead linearly scales with the group size and the number of servers. In particular, the group size impacts the server's running time to aggregate a group of profiles, i.e., the computation of $\boxed{\phi}$, whereas the number of servers influences the running time of decryption of $\boxed{\phi}$ (during which each server should verify the decryption integrity of $N - 1$ other servers). Adding a new server to the system will increase the run time of each server by $30ms$ which is the time required to verify the decryption of one server.

Recommended Number of Servers: In Privado, we utilize multiple servers in order to provide better privacy for the users, i.e., the privacy holds unless an adversary obtains control of all the servers. Thus, increasing the number of servers would make the job of the adversary harder (hence results in a stronger privacy guarantee). In contrast, the number of servers negatively influences the advertising running time and degrades the performance. Thus, the selection of N is a trade-off between privacy and performance. Note that, due to the privacy concern, N cannot be less than 2. For any value of N greater than 2, the best candidate can be set according to the computational power of servers and the desired performance. For instance, in a system with group size 11, if the desired running time of advertising protocol is less than 400 milliseconds, then the maximum value of N (which is the best for user privacy) would be 8 (in Figure 4.1, the running time line of $N = 8$ is the closest line

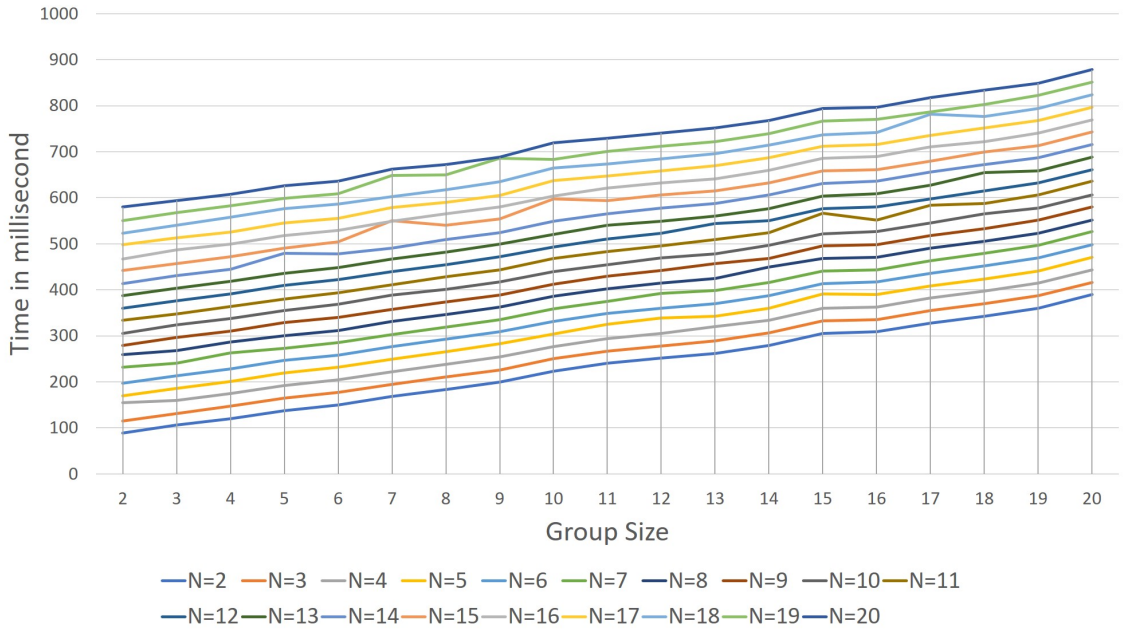

Figure 4.1: Running time of servers in advertising protocol over the different number of servers and group sizes. N indicates the number of servers.

to the intersection of group size 11 with the running time 400 milliseconds).

Privado vs PPAD: We compare the advertising running time of Privado with PPAD [67]. For the results to be comparable, we deploy 2 servers for Privado similar to PPAD. The computation complexity of PPAD is given in Table 1.b.

With respect to the computation complexity, both servers in Privado and PPAD run an identical set of instructions to perform aggregation (computation of $\boxed{\phi}$) and decryption. However, in Privado, servers have to verify each others' decryption results in order to provide security against malicious servers. In fact, the additional term $N \cdot \lambda$ in Table 1.a under the advertising row asserts this fact. That is, each server of Privado has to carry $O(\lambda)$ group multiplications to verify the decryption result of another server.

The simulation results (shown in Figure 4.2) comply with the asymptotic analysis. In order to clarify this observation, in Figure 4.2, we plot the run time of Privado under both honest-but-curious (HbC) and malicious adversarial models. In the HbC curve of Privado, we exclude the verification run time (i.e., servers do not verify the

ZKP of correct decryption of one another) assuming that they all trustfully follow the protocol. As it is apparent from the graphs, PPAD and Privado run identically in the HbC model. The malicious setting of Privado imposes 30 ms to the total run time (that is the verification run time), which is the time that each server spends to verify other server's output. This difference causes PPAD to be 1.2 times faster than Privado in the two-server setting. For instance, under the group size 7, PPAD performs advertising in 140 ms whereas Privado runs in 168 ms ($\frac{168}{140} = 1.2$). But, we emphasize that this computational cost has enabled Privado to stand a stronger adversarial model and deliver more robust security guarantee for its users.

Privado performance under Facebook Settings: In this part, we analyze the time required for a real OSN like Facebook to run our proposed advertising protocol over 2 billion users. First, we present the the computational power of Facebook and then measure the running time of our advertising protocol under that setting. Next, we also investigate the impact of network delay on the advertising performance.

Facebook's computational power for serving 2 billion users come from deploying 830,000 servers[1]. Under such configuration, each Facebook server gets to serve 2410 ($= 2*10^9/830,000$) users. Each Facebook server typically has two or more high-speed 64-bit Intel Nehalem processors of 3.2 GHz speed with 4 or 8 cores[2]. Given that in our simulation setting we used a 2.9 GHz processor, and the fact that the matching for each group and advertising request can be done independently in parallel, the running time of advertising protocol on Facebook servers would be almost ($3.2*8/2.93 = 8.8$) times faster than our simulation environment. Therefore, in a 2 server setting and under the group size of 5, the advertising protocol running time would drop from 149 ms (under our simulation setting) to 17 ms (under Facebook servers), i.e., $\frac{149ms}{8.8} = 17$ ms. Hence, in practice, an OSN like Facebook can find the target groups of an advertising request within (2410/5) * 17 ms= 8194 ms= 8.2 seconds (note that each server of Facebook serves 2410 users which corresponds to 482 = 2410/5 groups of size 5).

The running time of advertising protocol is also influenced by the network delay

(or the Round Trip Time (RTT) between every pair of servers), where the servers need to run distributed decryption protocol, hence, send their partial decryption results to $N-1$ other servers. This imposes one RTT to the running time of each advertising protocol (this overhead is independent of the number of servers since each server can communicate with other servers simultaneously). Furthermore, since the result of the advertising protocol for each group is independent of the other groups, all servers can transmit all their partial decryption results of all the groups in one message with the $N-1$ other servers. To measure the RTT value, we carried out experimental analysis on Google cloud servers each with 1Tbit/s bandwidth. In our experiment, we set up multiple Google cloud servers distributed all around the globe and measure their pairwise RTT[3]. The average pairwise RTT is calculated as 20 ms. Thus, the overall impact of network delay on advertising protocol running time (to find all the target groups of an advertising request) would be 20 ms. As such, in the OSN with 2 billion users with group size 5 (which results in $\frac{2*10^9}{5} = 4*10^8$ groups of size 5), the overall impact of network delay on each instance of advertising protocol would be $\frac{20ms}{4*10^8} = 5*10^{-8}ms = 0.05$ nanoseconds.

4.1.3 Advertisement Accuracy Metrics

In order to analyze the effect of different group sizes and threshold values on the advertising performance, we define two performance metrics, namely ***Target accuracy*** and ***Non-Target accuracy***.

Target accuracy indicates the fraction of target users who are served by the advertisement, as formulated in Equation 4.1

$$\textbf{\textit{Target accuracy}} = \frac{\text{Number of target users served by the advertisement}}{\text{Total number of target users}} \tag{4.1}$$

This metric is in compliance with the advertiser desire who wants to reach as many target users as possible. Due to the nature of group-based advertising, the *Target accuracy* is not always 100% since the target users in groups with fewer than threshold-many target users are not shown the advertisement.

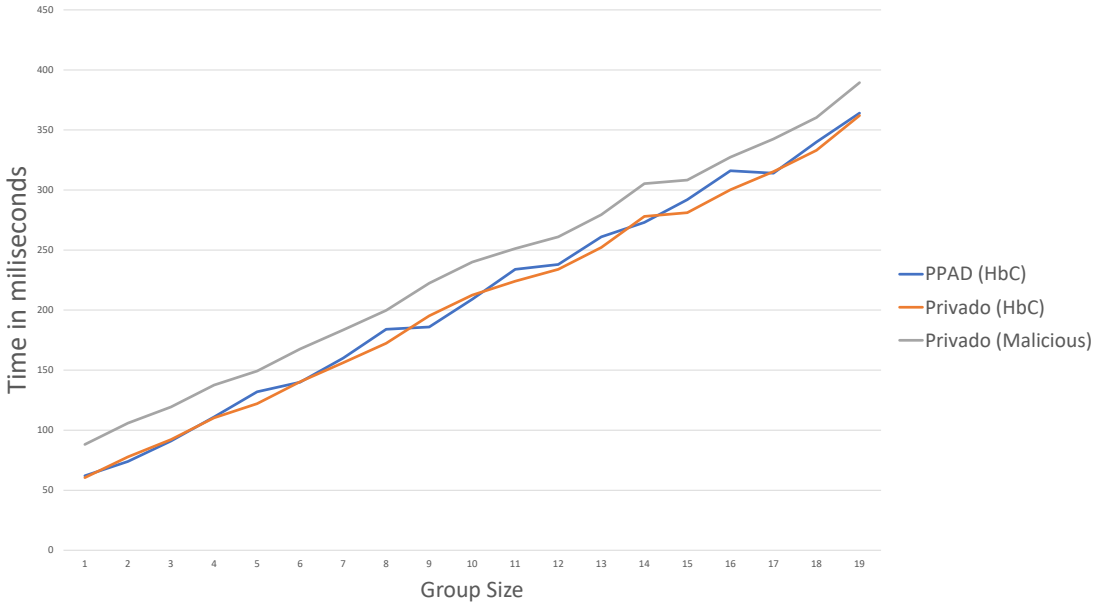

Figure 4.2: The comparison of running time of advertising protocol in Privado and PPAD over different group sizes. Both systems deploy two servers. In the HbC model of Privado, servers do not verify each other's computations (assuming servers trustfully follow the protocol).

Non-Target accuracy as shown in Equation 4.2 is the fraction of non-target users that are **not** served an (irrelevant) advertisement.

$$\textbf{\textit{Non-Target accuracy}} = \frac{\text{Number of non-target users not served by the advertisement}}{\text{Total number of non-target users}} \tag{4.2}$$

The higher value of this metric indicates that users are less likely to be shown irrelevant advertisement (hence more accurate is the advertising and less disturbing).

Note that the *Target accuracy* and *Non-Target accuracy* are meaningful only in the group-based advertising paradigm and not in personalized counterparts (where both measures are perfectly satisfied with the cost of privacy loss).

We additionally define the notion of **target coverage**, which is the fraction of target users, as follows:

$$\textbf{Target Coverage} = \frac{\text{Number of target users}}{\text{Total number of users}} \tag{4.3}$$

The coverage value depends on the attribute distribution in profiles as well as the content of the advertisement. In our experiments, we target various levels of coverage and analyze the effect of our system parameters.

4.1.4 Advertisement Accuracy Results

We explore the effect of group size and threshold value on the *Target accuracy* and *Non-Target accuracy*. The results are taken over 100,000 profiles with three different target coverage values (10%, 50% and 90%) as demonstrated in Figure 4.3. The results present that under a specific group size, increasing the threshold value improves the *Non-Target accuracy*. This behavior is expected since having a higher threshold guarantees that more target customers are in the target groups (compared to the lower thresholds). Hence in such settings, the higher percentage of target group members are real target customers i.e., the *Non-Target accuracy* is higher. On the contrary, the *Target accuracy* has the inverse relation with the threshold value. Indeed, higher threshold imposes more constraint on the group for being selected as a target. Consequently, the advertiser loses some of his target customers in the groups which do not have enough target users.

On the other hand, with a fixed threshold, as the group size increases, *Target accuracy* increases but *Non-Target accuracy* decreases. This happens for all target coverages, since in a larger group with the same threshold, it is easier to find matching groups, but it also means that potentially more non-target users are shown an irrelevant advertisement.

By inspecting the behavior of *Target accuracy* and *Non-Target accuracy*, we find out that a perfect balance between these two metrics is met when the ratio of the threshold to the group size i.e., $\frac{Thr}{\text{Group Size}}$ is close to the target coverage. We refer to this threshold value as **"balanced threshold"**. For instance, under the target coverage 50% and group size 19, the balanced threshold is 10 with $\frac{10}{19} = 0.52 \approx 0.5$. At this balance threshold, *Target accuracy* and *Non-Target accuracy* are 58% and 60%, respectively. We refer to the accuracy achieved at the balance threshold by

balanced accuracy. In Figure 4.3, the x coordinate of the point where two curves of the same color (i.e., same group size) collide indicates the balanced threshold and the accuracy at that point (y coordinate) is the balanced accuracy. After the balanced threshold, the *Target accuracy* drops while the *Non-Target accuracy* increases. The inverse occurs for values less than the balanced threshold.

The simulation results demonstrate that as the group size increases, the balanced accuracy degrades. For example, under the target coverage of 50%, the balanced accuracy of group size 7 (at balanced threshold 4) is 65% whereas in group size 19 (at balanced threshold 10) it drops to 58%. The correctness of this fact can be verified by coverage 10 and 90 as well. This implies that smaller group sizes are better for accuracy at their respective balanced thresholds.

In general, threshold being equal to group size k would mean that all users in a matched group have the same attributes in the advertisement in common. Similarly, threshold of 1 where the advertisement is not matched would reveal that no user in that group contains all the attributes in the advertisement. Such leakages are independent of the underlying methodology, and hence are not analyzed, but should be considered when selecting the parameters.

4.2 Security

We analyze the security of Privado against an active adversary who controls at most $N-1$ servers, $k-2$ users per group, and an arbitrary number of advertisers (as discussed earlier, more than $k-2$ colluding users per group cannot be made secure by any technique, and clearly, if all servers are adversarial then privacy is not applicable). We consider a malicious adversary which may refuse to follow the protocols' instructions and act in an arbitrary way. We present the security definition of user privacy in Section 4.2.1 (originally proposed by [67]) and provide a full security proof against this most powerful adversary in Section 4.2.2.

4.2.1 Security Definition

We utilize the following game to illustrate the formal definition of user privacy. The game is played between an adversary and a challenger. The adversary directs all the colluding malicious entities in the system i.e. $N-1$ servers, $k-2$ users per group, and any number of advertisers. On the other hand, the challenger controls the rest of parties which are honest, i.e., 2 of the group members, 1 server and some of the advertisers. In this game, the adversary selects two sets of attributes for the honest users of a group and asks the challenger to register the honest users using those sets. The challenger assigns the sets randomly to the honest users and registers them accordingly. The adversary's task is to guess which attribute set is registered under which username. In secure group-based advertising, the adversary would not be able to guess the correct assignment with better than a negligible advantage. Thus, even though the adversary has the power to know and even set the attributes of the two honest users, he cannot decide which user has which attribute set.

Note that, this security definition perfectly copes with the security challenges we discussed in Section 3.5.1. In particular, if the adversary manages to mount any of those attacks, then it would be able to break this game. For example, if the adversary manages to reuse one of the honest members profile for a corrupted member (mount replay attack), then (as we explained in Section 3.5.1) this would result in a specific pattern in the total number of target users of the group. As such, adversary can precisely learn the assignment of attributes to honest users and win the game. Therefore, the failure of the adversary in this game means the resilient of the design against all those attacks.

We consider $UPrivacy_{\mathcal{A}}(\lambda)$ a probabilistic experiment defined in terms of a game played between adversary $\mathcal{A}$ and a challenger, as given below [67].

User privacy experiment $UPrivacy_{\mathcal{A}}(\lambda)$**:**

1. Adversary and challenger are given the security parameter λ. They execute the Initialization protocol.

2. Adversary registers $k-2$ users to the group with the GID^* to be attacked. Subsequently, the challenger takes the role of the other two users whose usernames are denoted by $Uname_0$ and $Uname_1$. This step can be executed after step 4 as well.

3. Adversary outputs two attribute sets Att_0 and Att_1 to be assigned to the two honest users in the group to be attacked.

4. Challenger selects a bit value b randomly. He assigns Att_b and $Att_{\hat{b}}$ to $Uname_0$ and $Uname_1$, respectively. Finally, the challenger creates two profiles accordingly and runs the UReg (User Registration) protocol on behalf of these two users together with the adversary.

5. challenger registers advertisers upon adversary's request with the attributes the adversary provides. This step can be run polynomially-many times.

6. The advertising protocol is executed jointly by the challenger and the adversary for polynomially many advertising requests.

7. The adversary outputs a bit b'. If $b == b'$ then the output of the experiment is 1 indicating that the adversary succeeds, otherwise it is 0.

Definition 1 *Privado protects user privacy against an active adversary if for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* $negl(\lambda)$ *where* λ *is the security parameter such that:*

$$Pr[UPrivacy_{\mathcal{A}}(\lambda) = 1] \leq \frac{1}{2} + negl(\lambda) \tag{4.4}$$

4.2.2 Formal Security Proof

Proof Overview: Our proof consists of two parts. In the first part, illustrated in Theorem 1, we show that breaking user privacy of Privado would compromise the IND-CPA security of the underlying encryption scheme $TEnc$. This is done through the construction of a simulator B which ties user privacy game to the IND-CPA game. As the second part, we supply Theorem 2 in which we prove that B runs indistinguishable from the real challenger due to the IND-CPA security of the deployed encryption $TEnc$.

Note that in Privado, for efficiency, we deploy Fiat-Shamir heuristic to enable non-interactive ZKPs for POPR, POCM and the proof system of the verifiable shuffle scheme. Thus, the security of F^{R}_{POPR}, F^{R}_{POCM}, and F^{R}_{VS} are in the random oracle model (and subsequently, the proof of Theorem 1). However, we emphasize that our security proof also works intact in the standard model by utilizing interactive ZKPs.

Theorem 1 *Privado preserves user privacy as given in Definition 1, in F^{R}_{POPR}, F^{R}_{POCM}, F^{R}_{VS}, F_{THRES} hybrid model, assuming that the $TEnc$ encryption scheme is IND-CPA secure.*

Proof: If there exists a PPT adversary $\mathcal{A}$ who breaks user privacy with $\epsilon(\lambda)$ advantage, then we can construct a PPT adversary B who breaks the IND-CPA security of $TEnc$ with the same advantage. Let $h \in [1, N]$ be the index of the honest server that is run by B whereas $\bar{h}$ be the set of $N-1$ servers controlled by $\mathcal{A}$. B also simulates the ideal functionalities F^{R}_{POPR}, F^{R}_{POCM}, F^{R}_{VS}, and F_{THRES}. B works as follows.

1. Simulator B engages with $\mathcal{A}$ to execute the initialization phase. B, receives the encryption key pk and the security parameter 1^{λ} from IND-CPA challenger. B, simulating F_{THRES}, waits to obtain $(S_i, Generate, 1^{\lambda})$ message from each server $S_i \in \bar{h}$ and then asks $\mathcal{A}$ for a randomness. Upon the receipt of randomness,

B hands over pk (obtained from IND-CPA challenger) to $\mathcal{A}$. If $\mathcal{A}$ responds with *continue*, B proceeds with the rest of executions, otherwise sends *abort* to $P_{i\in h\cup\bar{h}}$ and terminates.

Next, one of the servers namely S_j creates the membership identifiers and their encryption. If $S_j \in \bar{h}$ (controlled by A) then two situations may happen. $\mathcal{A}$ either outputs the membership identifier set $\Delta = \{\delta_i\}_{i=1:k}$, its encryption $\boxed{\Delta} = \{\boxed{\delta_i} = Enc(\delta_i, r_i)\}_{i=1:k}$, together with the randomnesses $\{r_i\}_{i=1:k}$ or A outputs an empty set. In the former case, B verifies whether Δ satisfies Equation 3.13 and also whether $\boxed{\Delta}$ is the correct encryption of Δ using the given randomnesses. If $\mathcal{A}$ outputs an empty set then B generates Δ and $\boxed{\Delta}$ by itself and communicates Δ and $\boxed{\Delta}$ and the encryption randomnesses to $S_{i\in\bar{h}}$ to be verified by A.

2. (a) B and $\mathcal{A}$ run the mix-net. We assume shuffling starts from S_1 and ends with S_N. The initial input to the mix-net is $\boxed{\Delta_{\pi_0}} = \boxed{\Delta}$. For every $S_{i\in\bar{h}}$, $\mathcal{A}$ creates $\boxed{\Delta_{\pi_i}}$ by shuffling and re-encrypting $\boxed{\Delta_{\pi_{i-1}}}$ (π_{i-1} and π_i are the permutations of S_{i-1} and S_i, respectively). $\mathcal{A}$ calls F_{VS}^R to prove the correctness of its shuffle to every other servers including S_h. As such, $\mathcal{A}$ sends (Prove, S_i, S_j, (pk, $\boxed{\Delta_{\pi_{i-1}}}$, $\boxed{\Delta_{\pi_i}}$)) for $j \neq i$, and the witness ω_i as the proof of correct shuffle to F_{VS-R}. B acting as F_{VS-R} hands over (Prove, S_i, S_j, (pk, $\boxed{\Delta_{\pi_{i-1}}}$, $\boxed{\Delta_{\pi_i}}$)) to the adversary $\mathcal{A}$. If $\mathcal{A}$ responds with *continue*, B sends $(proof, S_i, S_j, (pk, \boxed{\Delta_{\pi_{i-1}}}, \boxed{\Delta_{\pi_i}}))$ to each $S_{j\in\bar{h},i\neq j}$, otherwise sends *abort* to $S_{j\in\bar{h},i\neq j}$ and terminates. A part of ω_i includes the permutation π_i (to indicate that $\boxed{\Delta_{\pi_i}}$ is the correct shuffle of $\boxed{\Delta_{\pi_{i-1}}}$). This means that B while simulating F_{VS}^R will learn this permutation. However, we remark that B will not use this information. B simulates F_{VS}^R and verifies the proof correctness. When the honest server S_h takes the turn, B simulates on behalf of it as follows. B populates $\boxed{\Delta_{\pi_h}}$ with k encryptions of a junk value δ' i.e., $\boxed{\Delta_{\pi_h}} = \{\boxed{\delta'}, ..., \boxed{\delta'}\}$. B delivers $\boxed{\Delta_{\pi_h}}$ to S_{h+1}

(that is controlled by $\mathcal{A}$). B acting as F_{VS-R} hands over (Prove, S_h, S_i, (pk, $\boxed{\Delta_{\pi_{h-1}}}$, $\boxed{\Delta_{\pi_h}}$)) for $S_i \in \bar{h}$ to the adversary $\mathcal{A}$. If $\mathcal{A}$ responds with *continue*, B sends ($proof, S_h, S_i, (pk, \boxed{\Delta_{\pi_{h-1}}}, \boxed{\Delta_{\pi_h}})$) to each $S_{i\in\bar{h}}$ playing as F^R_{VS}. Otherwise, sends *abort* to $S_{i\in\bar{h}}$ and terminates. If $h = N$ then the honest server's (B's) output $\boxed{\Delta_{\pi_h}}$ constitutes the final output of the mix-net.

(b) $\mathcal{A}$ creates $k-2$ profiles on behalf of corrupted members. As such, $\mathcal{A}$ communicates $\boxed{BF_j}$, and $\boxed{Pf_j}$ with S_h (j indicates the index of corrupted member). Also, $\mathcal{A}$ proves the correctness of profiles to S_h by invoking F^R_{POPR} and F^R_{POCM} for every element of $\boxed{BF_j}$ and $\boxed{Pf_j}$, respectively. B shall act as F^R_{POPR} and F^R_{POCM} and verify the correctness of profiles. Note that according to Equation 3.10, B while simulating F^R_{POPR}, learns the profiles of corrupted members. That is, for each encrypted element $\boxed{b_{j,i=1:p}}$ of $\boxed{BF_j}$ (for each corrupted member j controlled by $\mathcal{A}$), $\mathcal{A}$ submits a witness ω to B which contains the corresponding plaintext $b_{j,i}$ (together with the encryption randomness). This enables B to learn all the bit value of the Bloom filter BF_j.

3. $\mathcal{A}$ outputs two attribute sets Att_0, Att_1 to be used for the honest users.

4. B generates two Bloom filters out of Att_0 and Att_1 namely BF_0 and BF_1. B sends $M_0 = \{BF_0, BF_1\} = \{b_{0,1}, ..., b_{0,p}, b_{1,1}, ..., b_{1,p}\}$ and $M_1 = \{BF_1, BF_0\} = \{b_{1,1}, ..., b_{1,p}, b_{0,1}, ..., b_{0,p}\}$ to the IND-CPA challenger and obtains $C = \{\boxed{b_{b,1}}, ..., \boxed{b_{b,p}}, \boxed{b_{\hat{b},1}}, ..., \boxed{b_{\hat{b},p}}\} = \{\boxed{BF_b}, \boxed{BF_{\hat{b}}}\}$. B homomorphically multiplies δ' into each encrypted Bloom filter $\boxed{BF_b}$ and $\boxed{BF_{\hat{b}}}$ and constructs $\boxed{Pf_b}$ and $\boxed{Pf_{\hat{b}}}$, respectively. B registers $\boxed{BF_b}$, $\boxed{Pf_b}$ under $UName_0$ and $\boxed{BF_{\hat{b}}}$, $\boxed{Pf_{\hat{b}}}$ for $UName_1$. B acts as F^R_{POPR} and F^R_{POCM} to prove the correctness of profiles to each server $S_{i\in\bar{h}}$. That is, for the j^{th} element of BF_b i.e., $\boxed{b_{0,j}}$ where $j \in [1, p]$, B sends (proof, $UName_0$, S_i, (pk, $\boxed{BF_{b,j}}$, $[0, 1]$)) to each $S_{i\in\bar{h}}$. Also, for the j^{th} element of $\boxed{Pf_b}$ B sends (proof, $UName_0$, S_i, (pk, $\boxed{BF_{b,j}}$, C_b, $\boxed{Pf_{b,j}}$)) to each

$S_{i\in\bar{h}}$. B acts similarly for $UName_1$, $\boxed{BF_{\hat{b}}}$ and $\boxed{Pf_{\hat{b}}}$

5. $\mathcal{A}$ and B start registering arbitrary advertising requests to the system.

6. $\mathcal{A}$ invokes the advertising protocol for an arbitrary advertising request Req. This step can be repeated polynomially many times. B computes the aggregation as $\boxed{\phi}$ based on the registered profile. Next, B, simulating F_{THRES}, waits for all the other servers $S_{i\in\bar{h}}$ to request decryption of $\boxed{\phi}$ to compute the decryption result. Observe that B does not own the decryption power so cannot execute decryption. However, in step 2.b, B acting as F^R_{POPR} could learn the content of corrupted users' profiles. Also, B knows the content of honest users profiles due to step 3. Thus, B can craft the decryption result i.e., ϕ on its own. As such, B first computes the matching results of corrupted members i.e., α_3,...α_k using the Bloom filters extracted in step 2 (α_is is defined in Equation 3.16). Then, B calculates the individual matching results of the honest members i.e., α_1 and α_2 based on BF_0 and BF_1, respectively. B needs to associate each matching result with a membership identifier (as given in Equation 3.18). Therefore, B shuffles Δ under a random permutation π and associates j^{th} element of Δ_π i.e., $\delta_{\pi(j)}$ with the j^{th} group member. B performs shuffling only once and then keeps using the same assignment for the rest of advertising requests. B outputs the aggregate value ϕ as $\sum_{j=1:k} \delta_{\pi(j)} \cdot \alpha_j$.

   Note that B acting as F_{THRES} is supposed to perform decryption of a ciphertext (i.e., encrypted aggregation) upon the request of all the N servers. Thus, B would only attempt decryption of ciphertexts that are agree with its own (i.e., the honest server's) local computations for which B knows the corresponding plaintext. Thus, B is always able to make a correct decryption indistinguishable from F_{THRES}. If $\mathcal{A}$ sends a decryption request for an arbitrary ciphertext which does not correspond to any aggregation computed by B, then B would never decrypt it.

7. $\mathcal{A}$ outputs a bit value b'. B delivers the same value to the IND-CPA challenger.

B carries polynomial operations at each step hence runs in polynomial time. Also, B simulates user privacy game to $\mathcal{A}$ indistinguishable from the original game as discussed next. Steps 1 is done as instructed in the real protocol. In step 2, B replaces all the membership IDs i.e., $\delta_1, ..., \delta_k$ with δ' which remains unnoticed to $\mathcal{A}$ due to the IND-CPA security of the encryption scheme (this is formally proven in in Theorem 2). Step 3 is run as expected. In step 4, B creates the content of the two honest users flawlessly. Though, B does not know the real content of $\boxed{BF_b}$ and $\boxed{BF_{\hat{b}}}$ to generate the profile correctness proof, B itself simulates F^R_{POPR} and F^R_{POCM} which enable him to approve to the adversary $\mathcal{A}$ that the profiles of honest members are constructed appropriately. Step 5 is executed as expected. In step 6, the decrypted aggregate value ϕ constructed by B is certainly admissible to $\mathcal{A}$ since the decryption is modeled by the ideal functionality F_{THRES}.

Notice that the execution of the protocol at steps 1-3 and 5 convey no useful information regarding the attribute assignment of the honest members. This is the case since the simulator B uses identical membership identifiers δ' for all the group members including the honest ones. Thus, the only way to learn which attribute is assigned to which honest member is through the content of registered profiles. This should not be possible as the utilized encryption scheme is IND-CPA secure i.e., the profiles reveal no information about the underlying attributes. Hence, if $\mathcal{A}$ guesses bit b' with non-negligible advantage, the IND-CPA security of the underlying encryption is broken. In particular, the output bit b' asserts that $Att_{b'}$ is assigned to $UName_0$ i.e., $BF_{b'}$ is used in $UName_0$ registration. Recall that the profile of $UName_0$ is constructed based on the first part of IND-CPA challenger's output. Therefore, the IND-CPA challenger's output must be the encryption of $M_{b'}$ i.e., $\{\boxed{BF_{b'}}, \boxed{BF_{\hat{b'}}}\}$. Assuming that $\mathcal{A}$ has non-negligible advantage $\epsilon(\lambda)$ in winning the user privacy game, then B by outputting the same b' also breaks the IND-CPA game with non-negligible advantage $\epsilon(\lambda)$. This is a contradiction with the initial assumption stated in Theorem 1. Thus, $\epsilon(\lambda)$ is negligible. ■

As for the indistinguishability of $\{\delta', ..., \delta'\}$ from real membership IDs i.e., $\{\delta_1, ..., \delta_k\}$ in the mix-net execution, we construct a modified simulator B' which runs identical to B except that at step 3.a, during the mix-net execution, it uses the real membership identifiers. In particular, B' executes step 2.(a) honestly (not with the junk identifiers).

Next we prove that $\mathcal{A}$ cannot distinguish its interaction with B and B' unless the underlying encryption scheme is not IND-CPA secure. Let $UPrivacy_{A,B'}(\lambda)$ indicate the user privacy experiment run between $\mathcal{A}$ and B'.

Theorem 2 *In F^R_{POPR}, F^R_{POCM}, F^R_{VS}, F_{THRES} hybrid model and assuming that the $TEnc$ encryption scheme is IND-CPA secure then*

$$|Pr[UPrivacy_{\mathcal{A},B}(\lambda) = 1] - Pr[UPrivacy_{\mathcal{A},B'}(\lambda) = 1]| < negl(\lambda) \tag{4.5}$$

Proof: If $\mathcal{A}$ can distinguish between its interaction with B and B' with non-negligible advantage ϵ', i.e.,

$$|Pr[UPrivacy_{\mathcal{A},B'}(\lambda) = 1] - Pr[UPrivacy_{\mathcal{A},B}(\lambda) = 1]| = \epsilon' \tag{4.6}$$

then we can construct an adversary D who can break the IND-CPA security of the encryption scheme with ϵ' advantage. D interacts with $\mathcal{A}$ as below.

1. D runs identical to B.

2. (a) D and $\mathcal{A}$ run the mix-net. Shuffling starts from S_1 and ends with S_N. For every $S_{i\in\bar{h}}$, $\mathcal{A}$ sends (Prove, S_i, S_j, (pk, $\boxed{\Delta_{\pi_{i-1}}}$, $\boxed{\Delta_{\pi_i}}$), ω_i) for $j \neq i$ to F_{VS-R}. π_{i-1} and π_i are the permutations of S_{i-1} and S_i, respectively. ω_i is the witness of correct shuffle. D acting as F_{VS-R} hands over (Prove, S_i, S_j, (pk, $\boxed{\Delta_{\pi_{i-1}}}$, $\boxed{\Delta_{\pi_i}}$)) to the adversary $\mathcal{A}$. If $\mathcal{A}$ responds with *continue*, D verifies the correctness of the proof and sends (Proof, S_i, S_j, (pk, $\boxed{\Delta_{\pi_{i-1}}}$, $\boxed{\Delta_{\pi_i}}$)) to $S_{j\in\bar{h},i\neq j}$, otherwise sends *abort* to $S_{i\in\bar{h},i\neq j}$ and terminates. When S_h takes the turn, D simulates on behalf of it as follows. D selects a random permutation π_h and permutes the plaintext Δ as $\Delta_{\pi_h} = \{\delta_{\pi(1)}, ..., \delta_{\pi(k)}\}$.

D constructs $M_0 = \{\delta_{\pi(1)}, ..., \delta_{\pi(k)}\}$ and $M_1 = \{\delta', ..., \delta'\}$ and hands over to the IND-CPA challenger. δ' is a junk identifier. As the result, D receives ciphertext $C = \{C_1, ..., C_k\}$. He sends $\boxed{\Delta_{\pi_h}} = \{C_1, ..., C_k\}$ to S_{h+1} (that is controlled by $\mathcal{A}$). D acting as F_{VS-R} hands over (Prove, S_h, S_i, (pk, $\boxed{\Delta_{\pi_{h-1}}}$, $\boxed{\Delta_{\pi_h}}$)) for $S_i \in \bar{h}$ to the adversary $\mathcal{A}$. If $\mathcal{A}$ responds with *continue*, D sends $(proof, S_h, S_i, (pk, \boxed{\Delta_{\pi_{h-1}}}, \boxed{\Delta_{\pi_h}}))$ to each $S_{i\in\bar{h}}$ playing as F^R_{VS}. Otherwise, sends *abort* to $S_{i\in\bar{h}}$ and terminates.

(b) $\mathcal{A}$ registers $k-2$ users into the system. D acts identical to B. Recall that in this step, B will learn the content of registered profiles i.e., $BF_3, ..., BF_k$.

3. $\mathcal{A}$ outputs two attribute sets Att_0 and Att_1.

4. D generates two Bloom filters out of Att_0 and Att_1 namely $BF_0 = \{b_{0,1}, ..., b_{0,p}\}$ and $BF_1 = \{b_{1,1}, ..., b_{1,p}\}$. He constructs $\boxed{Pf_0}$ using BF_0 and the membership ID C_1, and $\boxed{Pf_1}$ using BF_1 and the membership ID C_2. Next, D flips a coin $b \leftarrow \{0, 1\}$ and registers $\boxed{BF_b}$, $\boxed{Pf_b}$ under $UName_0$ and $\boxed{BF_{\hat{b}}}$, $\boxed{Pf_{\hat{b}}}$ for $UName_1$. Then, D acts as F^R_{POPR} and F^R_{POCM} to prove the correctness of profiles to each server $S_{i\in\bar{h}}$. That is, for the j^{th} element of BF_b i.e., $\boxed{b_{0,j}}$ where $j \in [1, p]$, D sends (proof, $UName_0$, S_i, (pk, $\boxed{BF_{b,j}}$, $[0, 1]$)) to each $S_{i\in\bar{h}}$. Also, for the j^{th} element of $\boxed{Pf_0}$ D sends (proof, $UName_0$, S_i, (pk, $\boxed{BF_{b,j}}$, C_b, $\boxed{Pf_{b,j}}$)) to each $S_{i\in\bar{h}}$. D acts similarily for $UName_1$, $\boxed{BF_{\hat{b}}}$ and $\boxed{Pf_{\hat{b}}}$.

   Observe that in the view of adversary $\mathcal{A}$, D runs this step the same as B (since similar to B, D also ends up with a random assignment of profiles to the honest users).

5. D and $\mathcal{A}$ register their own advertising requests.

6. $\mathcal{A}$ invokes the advertising protocol for an arbitrary advertising request Req. D acts the same as B.

7. $\mathcal{A}$ outputs a bit value b'. If $b == b'$ then D outputs 0, otherwise 1.

Clearly, D operates in polynomial time. D simulates indistinguishable from the real challenger (as well as indistinguishable from B) in user privacy game. All the steps are run identical except the shuffle operation. If the bit choice of IND-CPA challenger is 0 then D receives $C = \{\boxed{\delta_1}, ..., \boxed{\delta_k}\}$. In such case, the view of $\mathcal{A}$ is identical to its interaction with B'. If $C = \{\boxed{\delta'}, ..., \boxed{\delta'}\}$ then $\mathcal{A}$ experiences an interaction with B. According to Equation 4.6 we have:

$$\begin{aligned}
\epsilon'(\lambda) =& |Pr[UPrivacy_{\mathcal{A},B'}(\lambda) = 1] - Pr[UPrivacy_{\mathcal{A},B}(\lambda) = 1]| \\
& |Pr[b == b' | C = \{\boxed{\delta_{\pi(1)}}, ..., \boxed{\delta_{\pi(k)}}\}] - Pr[b == b' | C = \{\boxed{\delta'}, ..., \boxed{\delta'}\}]| = \\
& |Pr[\text{D outputs } 0 | M = M_0] - Pr[\text{D outputs } 0 | M = M_1]| = \\
& |Pr[output(PubK^{CPA}_{D,TEnc}(\lambda, 0)) = 0] - Pr[output(PubK^{CPA}_{D,TEnc}(\lambda, 1)) = 0]|
\end{aligned}$$

The last equality holds due to IND-CPA security of $TEnc$ (Equation 3.6). If $\epsilon'(\lambda)$ is non-negligible, then D can break the IND-CPA security of $TEnc$ with the non-negligible advantage. This yields to a contradiction with Theorem 2. Thus, $\epsilon'(\lambda)$ is negligible which implies that $\mathcal{A}$ cannot distinguish its interaction with B and B'. ■

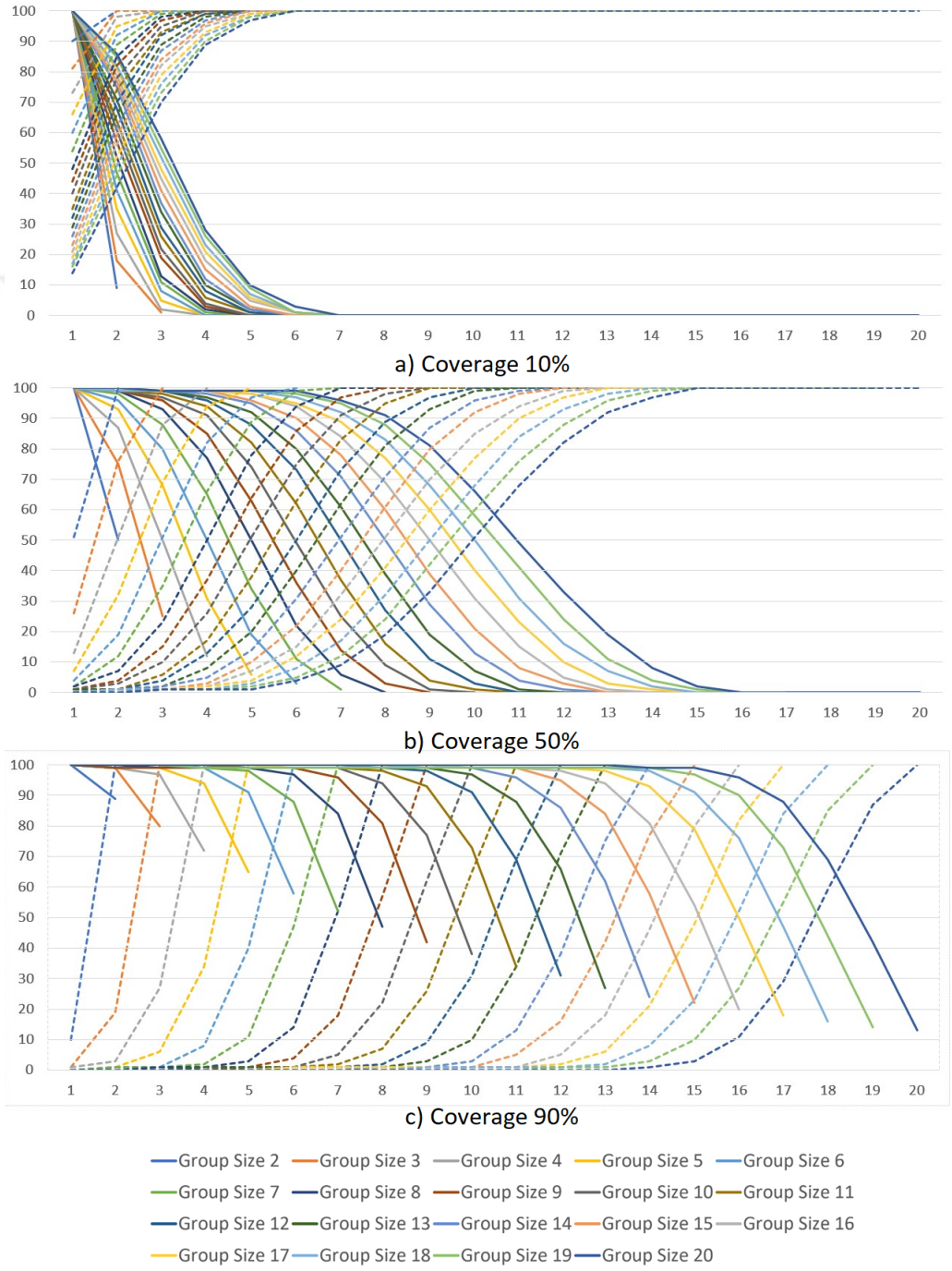

Figure 4.3: *Target accuracy* and *Non-Target accuracy* vs threshold for group sizes 2-20. Dashed curves represent *Non-Target accuracy* and solid ones the *Target accuracy*. X axis: threshold. Y axis: accuracy

Notes to Chapter 4

1 https://blog.sqlizer.io/posts/facebook-on-aws/

2 https://arstechnica.com/information-technology/2013/02/who-needs-hp-and-dell-facebook-now-designs-all-its-own-servers/

3 We deployed the code provided by "https://github.com/yhassanzadeh13/GC-Delay-Study"

Chapter 5

ANONYMA: ANONYMOUS INVITATION-ONLY REGISTRATION IN MALICIOUS ADVERSARIAL MODEL

5.1 Introduction

Motivation: Invitation-based systems (or invitation-only registration systems, interchangeably) are services in which registration is possible only through getting invitations from the current members of the system. Many reasons encourage invitation-only registration policy e.g., the lack of sufficient resources to cover arbitrary many users, improving the quality of service by constraining the number of members, and to protect the system against spammers or undesirable users [109].

In a nutshell, an invitation-only system is comprised of an *administrator/server*, a set of members, who will act as *inviters*, and a new user, called *invitee*, who wants to join the system. Figure 5.1 illustrates the parties and their interaction. The new user (i.e., invitee) can register to the system by being invited from the existing members. Successful registration relies on having a certain number (i.e., t) of invitations from t distinct members. The invitee collects the invitations and hands over them to the administrator who checks whether the invitations are issued by legitimate members. If so, it accepts the registration request and allows the invitee in. Additionally, the administrator may issue credentials for the invitee to be able to start inviting others.

Google initiated this invitation-only policy when deploying services such as Google Inbox, Orkut, and Google Wave[1]. Another successful system with an invitation-only registration is *Spotify*[2]. *Facebook* also has secret groups in which new users can participate upon getting invitations from other group members[3]. Similarly, messengers

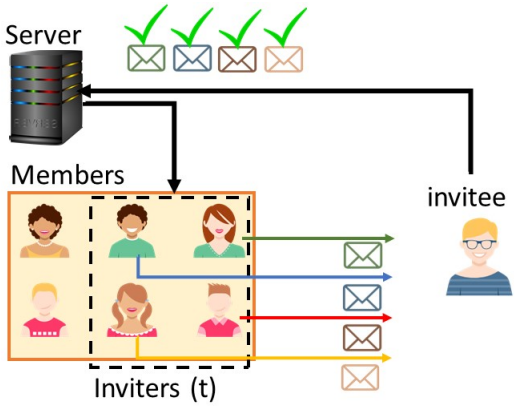

Figure 5.1: The sample workflow of an invitation-only registration system. The server issues credentials for the members to generate invitations. The invitee collects t invitations and sends them to the server. Then, the server accepts or rejects the invitee's request for registration by checking whether each invitation is issued by a valid current member (resulting in knowing who is invited by whom).

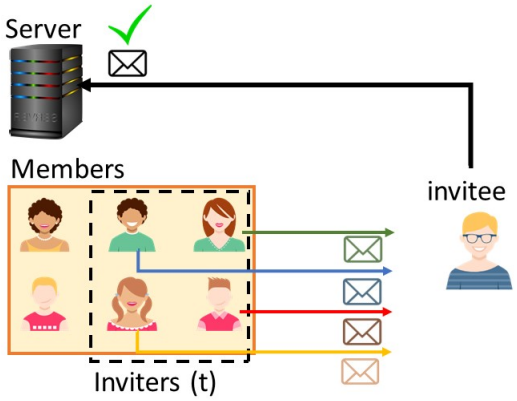

Figure 5.2: Anonyma overview. The invitee receives the individual invitations and aggregates them into a unified letter to be sent to the server. The server authenticates the letter without knowing the identity of the inviters.

such as *WhatsApp*[4] and *Telegram*[5] offer private groups running on the invite-only basis. The similar approach is sought in *PIE Register*[6] where it enables users to set up exclusive websites whose contents are only visible to the visitors invited by the website administrator or authorized members. Another closely relevant example is the trustee-based social authentication deployed by Facebook as a backup authentication method [16, 17, 110]. A backup authentication method is used when the user fails to pass the primary authentication, e.g., forgetting the password [16]. The account holder determines a set of trustees and informs the server in advance. When the user loses access to his account, the server sends recovery codes to the trustees (equivalent to inviters in the invitation-only registration scenario). Upon collection of enough number (recovery threshold) of codes from the trustees and handing the codes to the server, the user regains access to his account.

As we discussed in chapter 1, the invitee-inviter relation in invitation-only systems are prone to inference attacks where information like location, religious beliefs, sexual orientation, and political views can be extracted about an individual by analyzing the common features among his inviters. Due to this issue, inviter-invitee relation counts as privacy-sensitive information.

Related Work: Hiding the inviter-invitee relationship has similarly been addressed by researchers in the context of electronic voting systems and threshold ring-based signature schemes. However, those proposals suit their unique settings and become inefficient when utilized for invitation-only registration scenario. The issue in threshold ring-based signature schemes is that the computation complexity for invitation creation (at the user side) as well as invitation verification (at the server-side) is $O(N)$ where N is the total number of members in the system [111, 112, 113, 114]. The similar issue applies to the e-voting systems [115, 116] where all the existing members are required to get involved in every single registration. Added to this is the size of the invitation which grows with the number of registered members (more precisely, each invitation contains N group elements, where N is the total number of members).

Anonyma: In our prior work, Inonymous [117], we address the problem of inference

attack in invitation-based systems by proposing a protocol by which an invitee is able to authenticate herself to the group administrator needless to disclosing the identity of her inviters thus protecting the inviter-invitee relationship (or inviter anonymity). Additionally, Inonymous ensures that an invitee with an insufficient number of inviters would not be able to convince the administrator and register into the system. This feature is called invitation unforgeability. Inonymous guarantees the above features under an honest but curious adversarial model, namely, all the parties shall follow the protocol descriptions honestly. However, a corrupted administrator, by disregarding the protocol description, can damage inviter anonymity and learn who is invited by whom. To cope with this attack, we propose Anonyma [24] which extends Inonymous to withstand *malicious adversarial model* where parties may deviate from protocols' descriptions.

An overview of Anonyma is depicted in Figure 5.2. Anonyma consists of three entities: A server (administrator), existing members (inviters) and a newcomer (invitee). At the beginning of the system lifetime, the server registers an initial set of members who shall start inviting outsiders. For instance, In the Google Inbox example, the employees of Google can be the initial members. The invitee receives invitations from a subset of existing members i.e., inviters. The invitee knows the inviters beforehand via some other means outside the network to be joined to (e.g., Google employees invite their families/friends). The invitee combines the invitations into a single invitation letter and submits to the server. If the invitation is verified by the server, the invitee's registration request is accepted. The verification does not rely on any interaction between the inviters and the server. Upon successful registration, the server can decide to issue credentials for the new user to enable him to invite others. In particular, the ability to make invitations is up to the administrator who can either reserve this right to himself or share it with the members of his choice. In Anonyma, the confidentiality of inviter-invitee relationship (i.e., inviter-anonymity) is protected against both the server and the other members including inviters of the same invitee. Together with this confidentiality, Anonyma guarantees

invitation unforgeability where a malicious invitee cannot convince the server unless with holding threshold many legitimate invitations. We provide formal definition for *inviter anonymity* and *invitation unforgeability* in Section 6.2. Our definitions are slightly different from Inonymous in the sense that we allow the adversary to spoof the communication channels between inviters and invitee hence gain more information. A formal game-based proof of security against an *active/malicious adversary* (who disregards the protocol instructions and acts arbitrarily) follows in the same section 6.2.

Anonyma preserves the inviter anonymity i.e., hides the invitee-inviter relationship only for the registration. However, it does not cover the case that the inviter and invitee may reveal their relation through their interaction inside the service. For instance, in the Google Inbox example, Bob can get invited to the service by Alice using an anonymous invitation-only registration (i.e., no one knows Alice has invited Bob). However, later on, Bob may exchange an email with Alice which would imply a relationship between Alice and Bob. We emphasize that any interaction of this type which occurs after the registration phase must be considered out of the scope of this paper.

Anonyma imposes only $O(t)$ overhead on the invitee, and constant number of group exponentiations on the inviters (for the invitation creation) and the system administrator (for the authentication of new registration). As such, Anonyma provides better efficiency compared to prior studies whose computation overheads for the invitee and the server are linear in the total number of existing members. Furthermore, in Anonyma, the server is able to efficiently, and without re-keying the existing members, generate credentials for a newcomer to empower him for inviting others. This is significantly better than the prior proposals where the server has to carry out $O(N)$ communication overhead to submit some information about the newly registered user to each one of the current members.

Additionally, we propose AnonymaX, an anonymous cross-network invitation-based system on top of Anonyma which can be of independent interest. In the cross-

network design, a user joins one system e.g., Twitter, by obtaining invitations from members of another network e.g., Facebook. The cross-network design is beneficial especially to bootstrap a system, for example in the case where a research group wants to hire qualified researchers from another group, where a qualified researcher is the one with enough recommendations i.e., invitations from his group. We also prove that AnonymaX preserves inviter anonymity and invitation unforgeability against a malicious adversary.

5.2 Related Works

In this section, we investigate related studies under two main categories: Electronic Voting (e-voting) systems and Ring-based signature schemes. These two topics show the most similarity to the invitation-only registration systems and address the confidentiality of the inviter-invitee relationship. However, the research done in both categories suit their unique settings and suffer from the efficiency issues when deployed for the invitation-only registration scenario. More details are provided below.

5.2.1 Electronic Voting (e-voting)

Electronic voting systems consist of a set of voters, some candidates to be voted, and one/multiple authorities which handle tallying. An e-voting system must ensure that only the authorized users participate in the voting, and each voter casts only one vote. More importantly, the content of the individual votes must be kept private, i.e., no vote can be traced back to its voter. In the literature, this property is known as vote privacy, anonymity, and untraceability. E-voting techniques are similar to the anonymous invitation-only systems in many aspects. The role of voters is analogous to the inviters. Each round of the election with the Yes/No votes for a candidate can be treated as inviters casting their invitations for the registration of a newcomer. A Yes vote indicates inviting the candidate/newcomer and a No vote implies not inviting. Preserving the privacy of the vote is equivalent to the inviter anonymity. Likewise, the prevention of double-voting resembles the invitation unforgeability.

Despite the aforementioned similarities, e-voting proposals fall short in satisfying inviter anonymity, invitation unforgeability, and scalability simultaneously. To illustrate this incompatibility, we first classify the e-voting techniques into two main categories: 1- explicit vote casting, 2-anonymous vote casting. Under each category, we identify the subtleties to transplant the e-voting solution into the invitation-only systems.

1. Explicit vote casting: The voter authenticates himself to the authorities explicitly and immediately casts his private ballot. The ballot is shielded using either a threshold encryption scheme whose decryption key is divided between multiple authorities [118, 119], or secret sharing schemes where multiple authorities obtain one share of the ballot [120]. Before tallying the votes, the identifiable information shall be removed from the individual votes either by shuffling them through mix-net [121] or by homomorphically aggregating them [120]. In the context of the invitation-only system, this type of proposal has performance problems. That is, to preserve the inviter's anonymity (namely, hiding the identity of voters with the Yes vote), all the members should participate in the voting (including those who will cast a No vote). Otherwise, the real inviters will be revealed to the voting authorities. This imposes an unnecessary load to the non-inviters (voters with the No votes). In contrast, in Anonyma, the entire invitation procedure is carried out only by the invitee and his inviters.

2. Anonymous vote casting: This technique relies on one-time pseudonyms together with an anonymous communication channel. A voter hands over its credential (e.g., social security number – SSN) to the voting authority. Then, through a blind signature scheme, the voting authority issues a signature on the voter's pseudonym (that is also bond to the voter's SSN). The pseudonym is a one-time value and untraceable to the real identity, i.e., SSN. Later on, a voter casts a vote under his pseudonym and via an anonymous communication channel to the voting authority. Voters attempting voting twice will have to

risk the disclosure of their real identities (i.e., SSN) [115, 116]. When we integrate this solution to the invitation-only system, the main problem is that the pseudonyms are one-time hence a user cannot use the same pseudonym for multiple elections (i.e., to invite different users). Otherwise, his identity and will be disclosed. To cope with this issue, upon the arrival of each newcomer, the authority has to issue new pseudonyms for all the existing members to enable them to act as the inviter for the newcomer (regardless of being the inviters of the newcomer or not). This is certainly not an efficient solution as the load of the authority scales linearly with the number of joining members. Moreover, all the existing members also have to work linearly in the number of joining users. Alternatively, the authority should issue multiple pseudonyms (instead of one) for each member. However, this is not clear how to ensure that an inviter will only be able to use one pseudonym for each newcomer. In other words, the inviter should not be able to use all of his pseudonyms to make t valid invitations for just a single invitee since otherwise it would violate the invitation unforgeability (in which the invitations must be issued by t *distinct* inviters).

5.2.2 (t,N) Threshold Ring Signature

A ring signature specifies a group of N signers together with a proof that shall convince any verifier that a message is signed by t members of the group [122]. A ring signature scheme must satisfy three properties: 1-correctness which denotes that every group of t signers must be able to create a valid signature and proof, 2-unforgeability that means making valid signature is not feasible for non-signers, and 3-anonymity which indicates that given a signature and its proof, the identity of the signers should not be predictable with probability negligibly better than $\frac{1}{N}$.

An invitation-based system can be instantiated from a threshold ring signature, assuming that the signers are the inviters and a valid signature constitutes a valid invitation for a newcomer. The important shortcoming of such schemes is that their running time complexity for the generation of an invitation is at least linearly dependent on

the size of the system, i.e., the total number of existing members [111, 112, 113, 114]. In some other cases, the dependency is exponential [122]. The same issue applies to the length of the signature (invitation letter) as it is comprised of $O(N)$ group elements where N is the total number of existing members. This considerably degrades the system's performance. In contrast, the performance of Anonyma is only influenced by the threshold of t and is independent of the size of the system. That is, the invitation generation complexity is $O(t)$, and the invitation verification is done in $O(1)$. Also, the invitation length is $O(1)$.

5.3 System Model

5.3.1 Model

Anonyma is composed of three types of entities: a server, a set of existing members (inviters), and a new user (invitee) who is willing to join. The server sets up the system parameters, generates and distributes some secret values among users, and administers user registrations. For successful registration, each newcomer needs to obtain a threshold many (denoted by t) invitations from the existing members. The inviters exchange the invitations with the invitee out of band, e.g., via a messaging application. After the collection of t invitations, the invitee removes the identifiable information from the individual invitations (through aggregation) and submits a final invitation letter to the server. The server authenticates the letter. Upon successful authentication, the server lets the new user register and can issue credentials to empower him to act like an inviter and make invitations. Note that the system shall start by having at least t initial members who begin inviting outsiders. These initial members are given credentials directly from the server. In the Google Inbox example, the initial members (account holders) can be the employees of Google. In our system, we assume all the entities communicate through a secure and authenticated channel.

5.3.2 Security Goal and Adversarial Model

Our security objective is two-fold: *Inviter Anonymity* and *Invitation Unforgeability*, which are explained below. In Anonyma, we aim to satisfy both objectives in the *malicious adversarial model*, where the entities may deviate from protocol specifications.

1. **Inviter anonymity**: As we discussed in Section 5.1, due to the inference attack possibility, the inviter-invitee relationship must be treated as privacy-sensitive information. Thus, by inviter anonymity, we aim to hide who is invited by whom. This relation should be protected against both the server and other inviters of the same invitee (as they might be also curious to learn the identity of other inviters). Putting these together, we assume that the adversary against the inviter anonymity may get to control the server and $t-1$ inviters of an invitee and aims at determining the identity of the remaining non-colluding inviter. Please note that the invitee is concerned about his privacy, and hence has no incentive to expose the identity of his inviters to others. We formally propose a security definition for inviter anonymity in Section 6.2.2. Our definition also implies *between-inviter anonymity*, which refers to the fact that the anonymity of the invitee-inviter relationship holds even against the inviters of the same invitee.

2. **Invitation unforgeability**: The invitation unforgeability indicates that an invitee whose number of inviters ($t^{'}$) is less than the threshold t (i.e. $t^{'} < t$) should not be able to register to the system. Trivially, if the invitee already has t inviters, i.e. $t^{'} = t$, then he is an eligible person and can make a valid registration. We propose a security definition for invitation unforgeability in Section 6.2.3. That definition embodies the following security properties:

    - **Non-exchangability**: This means that invitations issued for a particular invitee are not reusable for another user. Otherwise, the current registered users can exchange their past invitations (by which they got invited to the system) with others and cause ineligible outsiders to join the system.

- **Preventing double invitation:** This feature indicates that an inviter cannot issue more than one valid invitation for a single invitee. This is essential since otherwise an invitee with insufficient inviters (i.e., $t^{'} < t$) can obtain multiple invitations (e.g., $t - t^{'}$) from one of her inviters and successfully register.

5.4 Notations, Definitions, and Preliminaries

5.4.1 Notations

We refer to a Probabilistic Polynomial Time entity as PPT. $x \in_R X$ and $x \leftarrow X$ both mean x is randomly selected from set X. $\perp$ indicates an empty string. $\equiv_c$ stands for computational indistinguishability. We use $DL_g(y)$ to indicate the discrete logarithm of y in base g. TTP stands for Trusted Third Party. The description of the notations used in Anonyma is provided in Table 5.1.

1^λ: The security parameter
$\perp$: Empty string
$\equiv_c$: Computational indistinguishability
t: The threshold for the number of inviters
N: The total number of existing members
ek, dk: Encryption key and decryption key
h: The public key of El Gamal Encryption
sk, vk: Signature key and verification key
S: Master value
s_i: A share of the master value for the i^{th} member
B_i: Lagrange coefficient computed for the i^{th} user w.r.t. a given set of points
G: A cyclic group
p, q: Prime numbers of length λ
g: The generator of a subgroup of G of order q
PRG: A pseudo random number generator

f: Polynomial of degree $t-1$
$a_0, \cdots, a_{t-1}$: The coefficients of polynomial f
$F_0, ..., F_{t-1}$: The commitments over the coefficients of f where $F_i = g^{a_i}$
γ_i: The commitment of the master share of the i^{th} user i.e., g^{s_i}
ω: The randomness inside the $Token$
η: Server generated signature
$Token$: The token issued by the server
δ_i: The randomness used by i^{th} user during $Igen$
τ_i: The first half of the invitation issued by the i^{th} user ($\tau_i = \omega^{s_i+\delta_i}$)
$e\delta_i$: The encryption of δ_i
Inv_i: The invitation letter issued by the i^{th} user i.e., $(\tau_i, e\delta_i)$
Δ: The aggregate of randomnesses in invitations of t inviters i.e., $\sum_{i=1}^{t} \delta_i$
$e\Delta$: The encryption of Δ
T: $\omega^{S+\Delta}$
$InvLet$: The final invitation letter $(T, e\Delta)$ given to the server
S_j: The j^{th} host server
sk_{S_j}, vk_{S_j}: Signature and verification key of the j^{th} host server
ek_{S_j}, dk_{S_j}: Encryption and decryption key of the j^{th} host server
$Param_{S_j}$: The public parameters generated by S_j
$e(.,.)$: Bi-linear map

Table 5.1: Notations used in Anonyma

5.4.2 Definitions

Negligible Function Function f is negligible if for $\forall p(.)$ where $p(.)$ is polynomial, there exists integer N s.t. for every $n > N$, $f(n) < \frac{1}{p(n)}$.

Computational Indistinguishability: Let $X = \{(in, \lambda)\}_{in \in \{0,1\}^*, \lambda \in \mathbb{N}}$ and $Y = \{(a, \lambda)\}_{in \in \{0,1\}^*, \lambda \in \mathbb{N}}$ be two series of random variables which are indexed with in and λ where in is the input and λ is the security parameter. The two distributions are com-

putationally indistinguishable i.e., $X \equiv_c Y$ if the following holds: $\forall$ D (a non-uniform polynomial-time distinguisher), $\exists$ a negligible function $negl(.)$ s.t. $\forall in \in \{0,1\}^*$ and $\forall \lambda \in \mathbb{N}$ [97]:

$$|Pr[D(X(in,\lambda)) = 1] - Pr[D(Y(in,\lambda)) = 1]| \leq negl(\lambda) \tag{5.1}$$

Secure Multi-Party Computation: Consider function $F(in_1, ..., in_N)$ $= (f_1(in_1, ..., in_N), \cdots, f_N(in_1, ..., in_N))$ that receives inputs in_i from i^{th} party to whom delivers $f_i(in_1, ..., in_N)$. F shall be run by a trusted third party. We refer to such execution as the *IDEAL* world. Assume γ^F is a multi-party protocol that computes F. The execution of γ^F by the interaction of parties constitutes the *REAL* world. γ^F is said to securely realize F if the following holds. That is, for every PPT adversary A in protocol γ^F with auxiliary input $aux \in \{0,1\}^*$ and controlling parties specified in P_c, there exists a PPT simulator Sim for the ideal functionality F, that $\forall$ security parameter λ:

$$\{IDEAL_{F,Sim(aux),P_c}(in_1, ..., in_N, \lambda)\}\} \equiv_c \{REAL_{\gamma^F,A(aux),P_c}(in_1, ..., in_N, \lambda)\} \tag{5.2}$$

$IDEAL_{F,Sim(aux),P_c}(in_1, ..., in_N, \lambda)$ represents the output of parties in interaction with ideal functionality F while Sim is controlling parties specified in set P_c. Similarly, $REAL_{\gamma^F,A(aux),P_c}(in_1, ..., in_N, \lambda)$ asserts the output of the parties interacting in protocol γ^F.

Hybrid Model: Assume θ is a multiparty protocol that makes use of a sub-protocol γ^F. γ^F in turn securely realizes the ideal functionality F. The hybrid model allows proving the security of θ by replacing γ^F with F. As such, for any execution of γ^F in the proof, parties contact a rusted third party running the ideal functionality F. This would be called F-hybrid model [97].

Sigma protocol: A Σ protocol is a three rounds proof system (P, V) for a relation R which satisfies the following properties [97]:

- **Completeness:** An honest prover P holding a private input w, where $(x, w) \in R$, can always convince an honest verifier V.

- **Special soundness:** There exists a polynomial time machine A that for every pair of accepting transcripts (a, e, z) and (a, e', z') (where $e \neq e'$) of an statement x, A extracts witness w s.t. $(x, w) \in R$

- **Special honest verifier zero knowledge:** There exists a PPT machine S which given statement x and e can generate an accepting transcript (a, e, z) whose distribution is the same as the transcript of the real interaction of P and V. More formally, $\forall (x, w) \in R$ and $e \in \{0,1\}^t$

$$\{S(x, e)\} \equiv_c \{(P(x, w), V(x, e))\} \tag{5.3}$$

  The output of simulator S is denoted by $\{S(x, e)\}$. $\{(P(x, w), V(x, e))\}$ indicates the output transcript of an execution between P (holding inputs x and w) and V (with inputs x and random tape e).

Zero-knowledge proof of knowledge from Σ protocols: Following the method given in [97], it is proven that one can efficiently construct a zero-knowledge proof of knowledge (ZKPOK) system from any sigma protocol. We refer to [97] for more details of such construction. Applying this method on a Σ protocol e.g., Π (defined for the relation R) will result to construction that securely realizes the ideal functionality F_{Π}^{R} (defined in Equation 5.4) in the presence of malicious prover and verifier. Namely, dislike Σ protocols, the prover and the verifier are not obliged to act honestly hence may deviate from protocol descriptions.

$$F_{\Pi}^{R}((x, w), x) = (\perp, R(x, w)) \tag{5.4}$$

x refers to the statement whose correctness is to be proven and w indicates the witness. The ideal functionality F_{Π}^{R} receives a common input x from the prover and the verifier as well as the private input w from the prover. F_{Π}^{R} outputs to the verifier whether x and w fit into the relation R.

5.4.3 Preliminaries

Pseudo Random Generator A deterministic polynomial time function $P : \{0,1\}^m \rightarrow \{0,1\}^{l(m)}$ (where $l(.)$ is a polynomial) is called Pseudo Random Generator (PRG) if $m < l(m)$ and for any probabilistic polynomial-time distinguisher D there exists a negligible function $negl(.)$ such that:

$$|Pr[x \leftarrow \{0,1\}^m : D(P(x)) = 1] - Pr[y \leftarrow \{0,1\}^{l(m)} : D(y) = 1]| = negl(m) \quad (5.5)$$

(t,n)-Shamir Secret Sharing Scheme The (t,n)-Shamir secret sharing scheme [123, 124] is a tool by which one can split a secret value into n pieces such that any subset of t shares can reconstruct the secret. The scheme works based on polynomial evaluations. Let F_q be a finite field of order q. The secret holder/dealer picks a reandom polynomial f of degree $t-1$ with coefficients from Z_q:

$$f(x) = \sum_{i=0}^{t-1} a_i \cdot x^i \quad (5.6)$$

The dealer sets the secret data S as the evaluation of that function at point 0 i.e., $f(0) = a_0 = S$. The share of each participant shall be one point on f e.g., $f(j)$ is the share of j^{th} shareholder. As such, a dealer can generate arbitrary many shares from its secret (i.e., by evaluating function f on a new point). Since each polynomial of degree $t-1$ can be uniquely reconstructed by having t distinct points of that function, t Shamir shareholders are able to reconstruct the secret. Given any t shares $\{(i, s_i)\}_{i=1}^{t}$, the secret reconstruction algorithm works as below.

$$S = f(0) = \sum_{i=1}^{t} s_i \cdot B_i \quad (5.7)$$

where B_is are Lagrange coefficients defined as

$$B_i = \sum_{i=1}^{t} s_i \prod_{j \neq i}^{j=1:t} \frac{j}{j-i} (\text{mod q}) \quad (5.8)$$

Shamir secret sharing scheme satisfies the following properties: 1) Given t or more than t shares, it can reconstruct the secret S easily; and 2) with knowledge of fewer

than t shares, it cannot reconstruct the secret S. Shamir's scheme is *information theoretically secure* relying on no computational assumption.

Shamir shares are homomorphic under addition operation i.e., let $[s_1]$ and $[s_2]$ be shares of S_1 and S_2 (using (t, n)-Shamir secret sharing scheme), then $[s_1] + [s_2]$ constitutes a share of $S_1 + S_2$.

Multiplicative Homomorphic Encryption Scheme A public key encryption scheme π consists of three algorithms key generation, encryption, and decryption, denoted by $\pi = (KeyGen, Enc, Dec)$. Using $KeyGen$, q pair of keys is generated called encryption key ek and decryption key dk. π is called multiplicative homomorphic encryption if for every a and b, $Enc_{ek}(a) \otimes Enc_{ek}(b) = Enc_{ek}(a \cdot b)$ where a and b belong to the encryption message space and $\otimes$ is an operation over ciphertexts. As an example, in El Gamal encryption [125], $\otimes$ corresponds to group multiplication. Additionally, we have $Enc_{ek}(a)^c = Enc_{ek}(a^c)$ where a is a plain message and c is any integer. Throughout the paper, we consider El Gamal scheme as our underlying encryption scheme.

Signature Scheme A signature scheme [126] Sig consists of three algorithms key generation, sign and verify denoted by $Sig = (SGen, Sign, SVrfy)$. A pair of keys (sk, vk) is generated via $SGen$ where sk is the signature key and vk is the verification key. The signer signs a message m using sk by computing $\eta = Sign_{sk}(m)$. Given the verification key vk, a receiver of signature runs $SVrfy_{vk}(\eta, m)$ to verify.

A signature scheme $Sig = (SGen, Sign, SVrfy)$ is said to be existentially unforgeable under adaptive chosen message attack if $\forall$ probabilistic polynomial time adversary A, there exists a negligible function $negl(.)$ s.t. the following holds [100]:

$$Pr[(sk, vk) \leftarrow SGen(1^\lambda); (m, \sigma) \leftarrow A^{Sign_{sk}(.)}(vk) \\ \text{s.t. } m \notin Q \text{ and } SVrfy_{vk}(m, \sigma) = accept] = negl(\lambda) \tag{5.9}$$

$A^{Sign_{sk}(.)}$ indicates that adversary has oracle access to the signature algorithm. Q indicates the set of adversary's queries to the signature oracle.

Computational Diffie-Hellman Assumption Given a cyclic group G of prime order q with a generator g, and two randomly selected group elements $h_1 = g^{r_1}, h_2 = g^{r_2}$, the Computational Diffie-Hellman (CDH) assumption [100] is hard relative to G if for every PPT adversary A there exists a negligible function $negl(\lambda)$ where λ is the security parameter, such that:

$$\Pr[\ A(G, q, g, h_1, h_2) = g^{r_1 \cdot r_2}] = negl(\lambda)$$

Zero-knowledge Proof of Knowledge of Discrete Logarithm (PODL) This proof system was initially introduced by Schnorr [100] for proving the knowledge of a discrete logarithm in the group G of prime order q with generator g. That is, for a given $\omega, g \in G$, one can prove the knowledge of $x \in Z_q$ s.t. $x = DL_g(\omega)$ (DL stands for discrete logarithm). We apply the method given in [97] to the Schnorr protocol to convert it to a zero-knowledge proof system. We refer to the resultant protocol by $ZKPODL((G, q, g, \omega), r)$. Let F^R_{PODL} (given in Equation 5.10) demonstrate the security guarantees of the $ZKPODL$ protocol over the relation R that is given in Equation 5.11. F^R_{PODL} shall be run by a trusted third party. X refers to the statement whose correctness is to be proven i.e., $X = (G, q, g, \omega)$ and the witness $W = x$, which is only known to the prover. The ideal functionality F^R_{PODL}, that is run by a TTP, receives a common input X from the prover and the verifier as well as the private input W from the prover. F^R_{PODL} outputs to the verifier whether X and W fit into the relation R.

$$F^R_{POIC}((X, W), X) = (\perp, R(X, W)) \tag{5.10}$$

$$R = \{((G, q, g, \omega), x) |\ g^x = \omega\ (mod\ p)\} \tag{5.11}$$

Zero-Knowledge Proof of Plaintext Knowledge This proof system is used to prove the plaintext knowledge of a given ciphertext. That is, given ciphertext C that is encrypted under public key pk, a prover proves the knowledge of x and r s.t.

$C = Enc_{pk}(x, r)$. r is the randomness used while encryption. We instantiate such proof system using the proposal of [127] for El Gamal encryption scheme.

Zero-Knowledge Proof of Discrete Logarithm Equality For a group G of prime order q and generators $g_1, g_2, h_1, h_2 \in G$, the ZKP of discrete logarithm equality is a protocol to prove that $h_1 = g_1^{\alpha}$ and $h_2 = g_2^{\alpha}$ where $\alpha \in Z_q$ [128].

Bilinear Map Consider G_1 and G_2 as multiplicative groups of prime order q. Let g_1 be the generator of G_1. We employ an efficiently computable bilinear map $e : G_1 \times G_1 \rightarrow G_2$ with the following properties [129]

- Bilinearity: $\forall u, v \in G_1$ and $\forall a, b \in \mathbb{Z}_q : e(u^a, v^b) = e(u, v)^{a \cdot b}$.
- Non-degeneracy: $e(g_1, g_1) \neq 1$.

5.5 Construction

Anonyma is comprised of three main entities, namely a *server*, a set of existing members who shall act as *inviters*, and newcomers/*invitee*s wishing to become a member of the system. The general interaction between the parties is illustrated in Figure 5.3. Anonyma consists of six algorithms: *SetUp, Token generation (Tgen), Invitation generation (Igen), Invitation collection (Icoll), Invitation Verification (Ivrfy)* and *Registration (Reg)*. The summary of each algorithm is explained in Section 5.5.1 followed by the full construction in Section 5.5.2. Throughout the paper, we assume that all the parties communicate via secure and authenticated channels.

5.5.1 Construction Overview:

- *SetUp*: The server invokes the *SetUp* algorithm with the input of the security parameter 1^{λ} to initialize the system parameters: a cyclic group G, a master value $S \in_R G$, as well as key pairs for a signature scheme (denoted by sk, vk) and ElGamal encryption (denoted by ek, dk). At the beginning of the system lifetime, the server needs to register at least t initial users so that they can start inviting others. These initial members are given credentials by the server to be

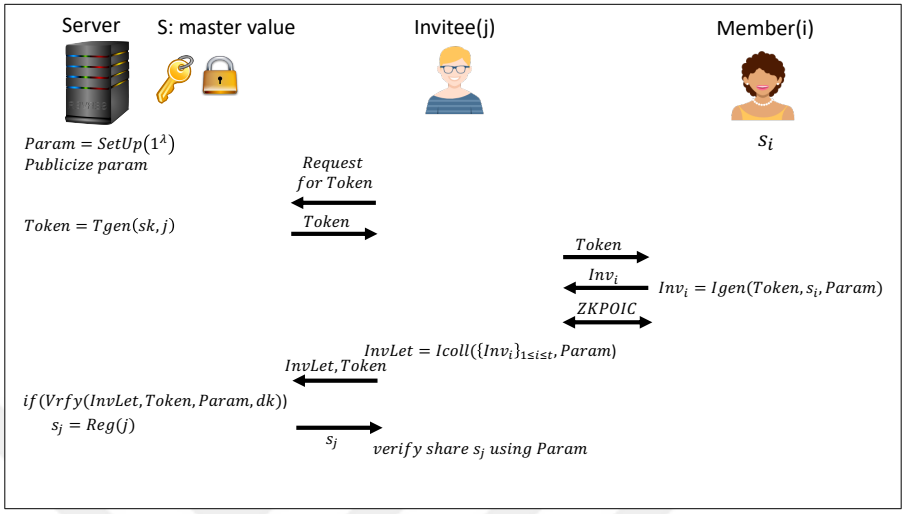

Figure 5.3: Parties' interaction in Anonyma.

able to make invitations. Each credential is indeed a share s_i of the server's master value S that is generated using (t, n)-Shamir secret sharing scheme. For the shares to be verifiable (the member can verify whether or not his piece is valid), the server publicizes the commitment to the selected polynomial function.

- *Tgen*: Each newcomer (i.e., invitee) contacts the server to get a token. The server runs the $Tgen$ algorithm to generate a $Token$ and hands it to the invitee. The $Token$ is a server signed certificate that embodies the index of the newcomer (each user is associated with a unique index) as well as a random element from the group G. Tokens shall be used by the inviters to issue an invitation for their intended invitee. Invitations issued for a particular token cannot be used for another token. This way, we guarantee the non-exchangeability of the invitations.

- *Igen*: The Invitee contacts each of his t inviters (this communication cannot be observed by the server/administrator) and communicates his $Token$ with them. Provided a valid token, each inviter generates an invitation by executing *Igen*. The invitation consists of two parts: a masked version of the inviter's

share s_i, and the masking value encrypted using the server's ek. The token is integrated into both parts of the invitation. As a part of $Igen$, the inviter has to prove in zero-knowledge that his invitation is well structured. For this sake, we devise a zero-knowledge proof protocol (i.e., Zero-Knowledge Proof of Invitation Correctness (ZKPOIC)). This proof helps in protecting between-inviter anonymity, i.e., inviters who collude with the server do not learn the identity of other inviters. Next, the inviter hands his invitation Inv_i to the invitee.

- $Icoll$: Upon the receipt of t invitations $\{Inv_i\}_{i=1}^{t}$, the invitee invokes the $Icoll$ algorithm through which he aggregates and blinds the invitations into a unified invitation letter $InvLet$. Aggregation and blinding remove any identifiable information about the identity of the inviters and helps in providing inviter anonymity (especially against a corrupted server). Additionally, through aggregation, the masking version of the master value S homomorphically gets reconstructed. We utilize the homomorphic property of both Shamir shares and the ElGamal encryption scheme to enable aggregation. At last, the invitee submits the final invitation letter $InvLet$ together with his $Token$ to the server.

- *Ivrfy*: The server authenticates the invitation letter by running *Ivrfy* and accepts or rejects accordingly. In a nutshell, the $InvLet$ is valid if and only if it contains the master value S.

- *Reg*: If the verification passes successfully (i.e., $Ivrfy$ outputs accept), the server runs the *Reg* algorithm to issue credentials for the newcomer to enable him to act as an inviter. This credential is a Shamir share of the server's master value S. The newcomer verifies the validity of his share using the parameters output by the server in the *SetUp* phase, and then stores his share for inviting others.

5.5.2 Full Construction:

Full construction of algorithms is explained in the followings.

SetUp:

This algorithm is run by the server who inputs the security parameter 1^λ and generates system parameters *Param* as follows.

- Two primes p and q of length λ such that $q|p-1$.
- g is a generator of a cyclic subgroup G of order q in Z_p^*.
- ElGamal encryption scheme $\pi = (EGen, Enc, Dec)$ with the key pair $(ek = h = g^a, dk = a)$ denoting encryption key and decryption key, respectively. dk remains at the server while ek is publicized.
- A signature scheme $Sig = (SGen, Sign, SVrfy)$. The signature and verification keys (sk, vk) are generated according to $SGen$. vk is publicized.
- A pseudo random generator $PRG{:}\{0,1\}^\lambda \rightarrow Z_q$
- A master value $S \leftarrow Z_q$
- A randomly chosen polynomial function $f(y) = a_{t-1}y^{t-1} + ... + a_1 y + a_0$ of degree $t-1$ whose coefficients $a_1, ..., a_{t-1}$ belong to Z_q and $a_0 = S$.
- The server initially registers t users into the system so that they can start inviting outsiders. Each user is associated with a unique index i and shall receive the evaluation of function f on that index, i.e. $s_i = f(i)$. We refer to s_i as the master share of the ith user.
- The server publicizes $F_0 = g^{a_0}, F_1 = g^{a_1}, \cdots, F_{t-1} = g^{a_{t-1}}$ as the commitment to the selected function f. Given $F_0, \cdots, F_{t-1}$, the computation of commitment

Algorithm 4 Tgen [Server]

Input: sk, j

Output: $Token$

1: $r \leftarrow Z_q$

2: $\omega = g^r$

3: $\eta = Sign_{sk}(j||\omega)$

4: $Token = (\eta, j, \omega)$

on $f(i)$ for any i is immediate as given in Equation 5.12. We will use γ_i to indicate g^{s_i}.

$$\gamma_i = \prod_{j=0}^{t-1} F_j^{i^j} = g^{a_0} \cdot g^{a_1 \cdot i} \cdots g^{a_{t-1} \cdot i^{t-1}} = g^{a_0 + a_1 \cdot i + \cdots + a_{t-1} \cdot i^{t-1}} = g^{f(i)} = g^{s_i} \quad (5.12)$$

- The server publicizes $Param = (G, p, q, g, ek, vk, (F_0, \cdots, F_{t-1}))$.

Token Generation:

Users wishing to register to the system first need to contact the server and obtain a token. The server generates a token through the token generation algorithm shown in Algorithm 4. In this procedure, the server initially assigns the user a unique index j. Indices can simply be assigned based on the arrival order of users, as long as no two users are assigned the same index. Hence, the j^{th} coming user receives the index value of j. Next, the server generates a random group element ω (lines 1-2) and certifies $j||\omega$ using his signing key sk (line 3). Let η be the signature outcome. The tuple (η, j, ω) constitutes the Token (line 4). We remark that the server is not required to record any information regarding the issued tokens. Thus, the generated tokens can simply be discarded and only the last value of j (the number of token requests) needs to be remembered. Therefore, we do *not* incur any storage load on the server per token.

Invitation Generation:

Invitation generation is run by the inviter to generate an invitation for a token given by the invitee. The procedure is shown in Algorithm 5. We assume that invitee and inviter communicate out of band (cannot be observed by the server/administrator), e.g., using a messaging application. Firstly, the inviter checks the authenticity of the token against the server verification key vk (line 1). Then, he samples a random value δ_i from Z_q by applying PRG on the random seed v (lines 2-3). Then, he blinds his master share using δ_i, i.e., $s_i + \delta_i$, and then ties this value to the provided token as $\tau_i = \omega^{s_i+\delta_i}$ (line 4). He also encrypts the masking value ω^{δ_i} as $e\delta_i$ using the server's encryption ek (line 5). To ensure that the inviter is acting honestly (i.e., generating the invitation as instructed in the algorithm), the inviter must prove the correctness of the invitation in zero-knowledge. To enable this, we propose a zero-knowledge proof system for the Proof Of Invitation Correctness, or for short $ZKPOIC$. The inviter and invitee engage in $ZKPOIC$ (line 7) through which the inviter proves the correctness of his invitation $Inv_i = (\tau_i, e\delta_i)$ to the invitee in zero-knowledge. In the followings, we explain our proposed proof system. We first draw a Σ protocol for $POIC$ and prove its security. Then, the zero-knowledge variant is immediate using the method proposed in [97, 130].

Σ Protocol for Proof Of Invitation Correctness (POIC): The invitation is constructed correctly if the inviter proves the following statements:

1. The inviter possesses a valid share of the master value S. That is, the inviter holding index i must prove the knowledge of the discrete log of γ_i, i.e., s_i. Note that $\gamma_i = g^{s_i}$ can be computed from $F_0, ..., F_{t-1}$ as explained in Equation 5.12.

2. The inviter knows the plaintext of $e\delta_i$, i.e., the knowledge of ω^{δ_i} and r such that $e\delta_i = (e\delta_{i,1} = \omega^{\delta_i} \cdot h^r, e\delta_{i,2} = g^r)$.

3. The randomness δ_i used in the creation of τ_i is correctly encrypted in $e\delta_i$. This can be captured by proving that $\tau_i \cdot e\delta_{i,1}^{-1} \cdot h^r$ $(= \omega^{s_i})$ and $\gamma_i = g^{s_i}$ have the same

Algorithm 5 Igen [Inviter]

Input: $Token = (\eta, j, \omega)$, s_i, $Param$

Output: $Inv_i / \perp$

1: **if** $Svrfy_{vk}(\eta, j||\omega)$=accept **then**
2: $v \leftarrow \{0,1\}^\lambda$
3: $\delta_i = PRG(v)$
4: $\tau_i = \omega^{s_i+\delta_i}$
5: $e\delta_i = Enc_{ek}(\omega^{\delta_i})$
6: $Inv_i = (\tau_i, e\delta_i)$
7: return inv_i //Inviter authenticates Inv_i through $ZKPOIC$
8: **end if**
9: return $\perp$

discrete logarithm s_i. The former is true due to Equation 5.13.

$$\tau_i \cdot e\delta_{i,1}^{-1} \cdot h^r = \omega^{\delta_i+s_i} \cdot \omega^{-\delta_i} \cdot h^{-r} \cdot h^r = \omega^{s_i} \tag{5.13}$$

To enable zero-knowledge proof of the aforementioned statements, we devise a Σ-protocol (P, V) as depicted in Figure 5.4. We refer to this proof system by Proof of Invitation Correctness, or *POIC*. POIC captures the relation R indicated in Equation 5.14. In our protocol, we incorporate the Shnorr protocol [100] for the proof of discrete logarithm knowledge (first anf third statement), proof of plaintext knowledge as proposed in [127] (for the second statement), and the proof of discrete logarithm equality [128] (for the fourth statement).

$$\begin{aligned}
&R = \{((\tau_i, e\delta_i = (e\delta_{i,1}, e\delta_{i,2}), \gamma_i, \omega), (s_i, r, \delta_i))| \\
&DL_g(\gamma_i) = s_i \ \wedge \\
&e\delta_{i,1} = \omega^{\delta_i} \cdot h^r \ \wedge \\
&DL_g(e\delta_{i,2}) = r \ \wedge \\
&DL_\omega(\tau_i \cdot e\delta_{i,1}^{-1} \cdot h^r) = DL_g(\gamma_i) = s_i\}
\end{aligned} \tag{5.14}$$

Prover (s_i, r, δ_i)		Verifier
$s', r', \delta' \leftarrow Z_q$		
$A = g^{s'}$		
$B_1 = \omega^{\delta'} \cdot h^{r'}$		
$B_2 = g^{r'}$		
$C = \omega^{s'+\delta'}$	$\xrightarrow{A,B=(B_1,B_2),C}$	
	$\xleftarrow{e}$	
$Z_1 = s' + e \cdot s_i$		
$Z_2 = \delta' + e \cdot \delta_i$		
$Z_3 = r' + e \cdot r$		
	$\xrightarrow{Z_1,Z_2,Z_3}$	
		if $(A \cdot \gamma_i^e == g^{Z_1}$
		$\wedge\, B_1 \cdot e\delta_{i,1}^e == \omega^{Z_2} \cdot h^{Z_3}$
		$\wedge\, B_2 \cdot e\delta_{i,2}^e == g^{Z_3}$
		$\wedge\, C \cdot \tau_i^e \cdot B^{-1} \cdot e\delta_{i,1}^{-e} \cdot h^{Z_3} == \omega^{Z_1})$
		Accept

Figure 5.4: Σ protocol of Proof of Invitation Correctness for the common input $\tau_i, e\delta_i = (e\delta_{i,1}, e\delta_{i,2}), \gamma_i, \omega$. The prover has the private input (s_i, r, δ_i).

Completeness: To prove that completeness holds, observe that if the prover P follows the protocol honestly, then due to the Equations 5.15, 5.16, 5.17, and 5.18, the verifier V accepts.

$$A \cdot \gamma_i^e = g^{s'} \cdot (g^{s_i})^e = g^{s'+e \cdot s_i} = g^{Z_1} \tag{5.15}$$

$$B_1 \cdot e\delta_{i,1}^e = (\omega^{\delta'} \cdot h^{r'}) \cdot (\omega^{\delta} \cdot h^{r})^e = \omega^{\delta'+e \cdot \delta_i} \cdot h^{r'+e \cdot r} = \omega^{Z_2} \cdot h^{Z_3} \tag{5.16}$$

$$B_2 \cdot e\delta_{i,2}^e = (g^{r'}) \cdot (g^{r})^e = g^{r'+e \cdot r} = g^{Z_3} \tag{5.17}$$

$$\begin{aligned} &C \cdot \tau_i^e \cdot B^{-1} \cdot e\delta_{i,1}^{-e} \cdot h^{Z_3} = \\ &(\omega^{s'+\delta'}) \cdot (\omega^{e \cdot s_i + e \cdot \delta_i}) \cdot (\omega^{-\delta'} \cdot h^{-r'}) \cdot (\omega^{-e \cdot \delta_i} \cdot h^{-e \cdot r}) \cdot h^{r'+e \cdot r} = \\ &\omega^{(s'+e \cdot s_i)} = \omega^{Z_1} \end{aligned} \tag{5.18}$$

We prove the special soundness and special honest verifier zero knowledge properties in Section 6.2.1. We additionally present the security properties of a zero-knowledge proof system for POIC that is achieved using the method given in [97, 130].

Invitation Collection:

Upon receipt of t invitations, the invitee runs the Invitation Collection ($Icoll$) procedure as indicated in Algorithm 6. The invitee aggregates τ_i values as $\prod_{i=1}^{t} \tau_i^{B_i}$ (line 3). He operates similarly for $e\delta_i$ values as $\prod_{i=1}^{t} e\delta_i^{B_i}$ (line 4). B_i values are the Lagrange coefficients (as defined in Equation 5.8) used for the reconstruction of the master value S from the Shamir shares. Next, the invitee randomizes both aggregates T and $e\Delta$ by adding a random value of his own choice, i.e., δ^*. The randomization cancels out the effect of the Lagrange coefficients and makes the final aggregates, i.e., T and $e\Delta$, independent of the B_i values. Recall that the Lagrange coefficients are dependent on the inviters' indices and by hiding them we aim at protecting inviter anonymity. The final invitation letter $InvLet$ shall be the pair $(T, e\Delta)$. The invitee submits the invitation letter and the token to the server.

Algorithm 6 Icoll [Invitee]

Input: $\{Inv_i = (\tau_i, e\delta_i) | 1 \leq i \leq t\}$, $Param$

Output: $InvLet$

1: $r \leftarrow \{0,1\}^\lambda$

2: $\delta^* = PRG(r)$

3: $T = \omega^{\delta^*} \cdot \prod_{i=1}^{t} \tau_i^{B_i}$

4: $e\Delta = Enc_{ek}(\omega^{\delta^*}) \cdot \prod_{i=1}^{t} e\delta_i^{B_i}$

5: $InvLet = (T, e\Delta)$

In Equations 5.19 and 5.20, we expand the result of T and $e\Delta$, which leads to the following observations.

$$\begin{aligned} T = \omega^{\delta^*} \cdot \prod_{i=1}^{t} \tau_i^{B_i} = \omega^{\delta^*} \cdot \prod_{i=1}^{t} \omega^{B_i \cdot s_i + B_i \cdot \delta_i} &= \omega^{\delta^* + \sum_{i=1}^{t} B_i \cdot s_i + \sum_{i=1}^{t} B_i \cdot \delta_i} \\ &= \omega^{S + \delta^* + \sum_{i=1}^{t} B_i \cdot \delta_i} = \omega^{S+\Delta} \end{aligned} \tag{5.19}$$

$$\begin{aligned} e\Delta = Enc_{ek}(\omega^{\delta^*}). \prod_{i=1}^{t} e\delta_i^{B_i} &= Enc_{ek}(\omega^{\delta^*}). \prod_{i=1}^{t} Enc_{ek}(\omega^{B_i \cdot \delta_i}) \\ &= Enc_{ek}(\omega^{\delta^* + \sum_{i=1}^{t} B_i \cdot \delta_i}) = Enc_{ek}(\omega^{\Delta}) \end{aligned} \tag{5.20}$$

The first observation is that T has the master value S embedded in its exponent. Intuitively, the presence of S in the exponent is a proof that the invitee has t distinct invitations. Since otherwise, the reconstruction of S would be impossible (we elaborate on this in Section 6.2 and formally prove the unforgability of invitations). Another observation is that the computation of both T and $e\Delta$ depends on the token ω. Hence, as desired, the resultant $InvLet$ is now bound to the given token. This would help for the non-exchangeability of the invitations. At last, T contains a masked version of master value, i.e., $S + \Delta$, in the exponent whereas $e\Delta$ embodies the corresponding masking value Δ. The encryption $e\Delta$ of the masking value shall be used at the server for the verification purpose (see invitation verification below).

Invitation Verification: Once the invitee hands his invitation letter $InvLet$ together with the corresponding token $Token$ to the server, the server executes the

Algorithm 7 Ivrfy [Server]

Input: $InvLet = (T, e\Delta), Token = (\eta, j, \omega)$, $Param$, dk

Output: $reject/accept$

1: **if** $Svrfy_{vk}(\eta, j||\omega)$=accept **then**
2: $\omega^{\Delta} = Dec_{dk}(e\Delta)$
3: **if** $\omega^S \cdot \omega^{\Delta} = T$ **then**
4: return accept
5: **end if**
6: **end if**

Algorithm 8 Reg [Server]

Input: j

Output: s_j

1: $s_j = f(j)$

invitation verification procedure shown in Algorithm 7. As the first step, the server authenticates the $Token$, i.e., whether it is signed under server's signature key sk (line 1). Next, the validity of the invitation letter $InvLet$ must be checked. For that, the server decrypts $e\Delta$ using its decryption key dk and obtains ω^{Δ} (line 2). Recall that Δ was used to mask the master value S in $T = \omega^{S+\Delta}$. Thus, if T and $e\Delta$ are constructed correctly, we expect that $\omega^S \cdot \omega^{\Delta} = T$ (line 3). If all the verification steps are passed successfully, then the server accepts the user's membership request.

Registration: The server invokes the registration procedure (Algorithm 8) for users who pass the verification phase (Algorithm 7). The input to Algorithm 8 is the index j of the newcomer, and the output is a Shamir share s_j of the master value S, where s_j is the evaluation of polynomial f at point j (line 1). Note that the index j is the index included in the user's $Token = (\eta, j, \omega)$. The server delivers s_j to the user who can then start making invitations as an inviter. The user authenticates his share by comparing the commitment γ_j (as given in Equation 5.12) against its own share, i.e., g^{s_j}. If they are equal, the user accepts and stores the share.

5.6 AnonymaX: Anonymous Cross-Network Invitation-Based System

Consider the situation where one system, e.g. Twitter, offers a special service for the users of another system, e.g. Facebook. We name Twitter as the *registration* network, i.e. the network serving a special service, whereas Facebook is called the *inviter network* whose users will benefit from the services offered by the *registration* network. A user of the *inviter* network is served by the *registration* network upon convincing the *registration* server on being invited by an adequate number of inviters from the *inviter* network.

Failed Approaches: One simple but cumbersome solution to empower a cross-network invitation-based system is to follow the regular invitation-based system, i.e. each time a *inviter* user wants to join the *registration* network, the *inviter* server authenticates that particular user and communicates the authentication result to the *registration* server. However, this solution requires the two servers to keep in contact with each other and imposes unnecessary overhead on the *inviter* server.

An alternative approach proposed by Inonymous [117] (our prior work), is that the *inviter* server would publicize the commitment over the master value S as g^S to the *registration* servers. Subsequently, *registration* servers would follow a different verification method (relying on bilinear maps) to authenticate invitations on their own. While this solution works for the honest but curious adversarial model, it fails in providing invitation unforgeability against a malicious adversary, which is explained next. Consider $registration_1$ and $registration_2$ as two *registration* servers. The corrupted $registration_1$ wants to join $registration_2$ as an invitee without enough inviters. $registration_1$ receives a token with the random value ω from $registration_2$ and then issues the same token to the users who want to join its own service. As such, $registration_1$ can reuse the invitation letters of his own users to craft a valid invitation letter to join $registration_2$. We address this issue in AnonymaX by making the *registration* servers prove in zero-knowledge (using an interactive proof) that they know the discrete logarithm $DL(w)$ of their issued tokens during the $Tgen$ protocol. As such, no *registration* server can issue tokens that are not generated by itself.

Algorithm 9 XTgen [*registration* Server S_j]

Input: $Param_{guest} = (G, q, g, ek_{guest}, vk_{guest}, (F_0, \cdots, F_{t-1})), sk_{S_j}, i$

Output: $Token$

1: $r \leftarrow Z_q$

2: $\omega = g^r$

3: $\eta = Sign_{sk_{S_j}}(i||\omega)$

4: $Token = (\eta, i, \omega)$

5: *Run* $ZKPODL((G, q, g, \omega), r)$

AnonymaX Overview: The *inviter* network with the master value S publicizes g^S as a part of its set of parameters $Param_{guest}$. Note that the description of group G is only generated by the *inviter* server and is used by other *registration* servers. On the other side, the *registration* networks denoted by S_j, $1 \leq j \leq N$, announce their $Param_{S_j}$ to be the pair of encryption keys ek_{S_j} and signature verification keys vk_{S_j}, i.e. $Param_{S_j} = (ek_{S_j}, vk_{S_j})$. The corresponding decryption key dk_{S_j} as well as the signature signing key sk_{S_j} remain private at the server side. Each invitee willing to join S_j shall obtain a token from S_j. During the token generation, the *registration* server follows Algorithm 4 and additionally must prove in zero-knowledge that it knows the discrete logarithm of the ω embodied in the token. As such, after the issuance of a token, the *registration* server runs an instance of *ZKPODL* protocol (given in section 5.4.3) with the invitee. The modified procedure is provided in Algorithm 9.

Upon a successful proof, the invitee accepts the token. The invitee needs to collect invitations from the members of the *inviter* network to be used in the registration of a particular *registration* network. Inviters issue invitation as in the regular invitation procedure given in Algorithm 5. However, the inviters should verify the tokens against the *registration* server verification key who has issued it. Also, the inviters shall use the encryption key of the *registration* network to encrypt their masking values. Indeed, in Algorithms 5 and 6, the inviter uses ek_j and vk_{S_j}, i.e. $Param_{S_j}$ as input. Therefore, the invitation letters received by the S_j are of the form $InvLet = (T, e\Delta)$ where $e\Delta$

Algorithm 10 XIVerify [Host Server S_j]

Input: $InvLet = (T, e\Delta), Token = (\eta, j, \omega), Param_{inviter}, Param_{S_j}, dk_{S_j}$

Output: *reject*/*accept*

1: **if** $Svrfy_{vk_{S_j}}(\eta, j||\omega)$=accept **then**
2: $\omega^{\Delta} = Dec_{dk_{S_j}}(e\Delta)$
3: **if** $e(\omega, g^S) \cdot e(\omega^{\Delta}, g) = e(T, g)$ **then**
4: return accept
5: **end if**
6: **end if**

is an encrypted masking value under ek_j. The *registration* server runs a different verification routine, which is given in Algorithm 10. We assume the existence of a bilinear map e: $G \times G \rightarrow G_2$ where G and G_2 are multiplicative groups of prime order q. The only difference between Algorithm 10 and Algorithm 7 is at the second verification step i.e., line 3. The correctness holds by the bilinearity of the bilinear map e, as in Equation 5.21.

$$\begin{aligned} e(\omega, g^S) \cdot e(\omega^{\Delta}, g) &= e(\omega, g)^S \cdot e(\omega, g)^{\Delta} = e(w, g)^{S+\Delta} = e(w^{S+\Delta}, g) \\ &= e(T, g) \end{aligned} \tag{5.21}$$

Notes to Chapter 5

1 http://www.macworld.com/article/1055383/gmail.html

2 https://community.spotify.com/t5/Accounts/Spotify-Family-Q-amp-A/td-p/988520

3 https://blog.hootsuite.com/facebook-secret-groups/

4 ttps://faq.whatsapp.com/en/android/26000123/?category=5245251

5 https://telegram.org/tour/groups

6 https://pieregister.com/features/invitation-based-registrations.

Chapter 6

ANONYMA: PERFORMANCE AND SECURITY

6.1 Performance

6.1.1 Running Time

In this section, we analyze the running time of each algorithm of Anonyma.

Simulation setting: The running time is measured on a standard laptop with 8 GB 1600 MHz DDR3 memory and 1.6 GHz Intel Core i5 CPU. The simulation setup consists of 100 registered members and 100 invitees. Each invitee collects threshold many invitations from randomly chosen inviters, i.e., the 100 initially registered members. The running time of all the algorithms is recorded over threshold values 1-10. The DSA signature scheme [131] is instantiated with the key length of 1024 bits.

Under the aforementioned setting, the running time of the parties are as follows, and the results are summarized in Table 6.1.

Server: The server spends 1.6 seconds in order to run the *SetUp* phase. This phase should be executed only once for the entire system lifetime. The *Token Generation* algorithm requires 4.24 milliseconds. The *Invitation Verification* procedure incurs 6.5 milliseconds. The *Registration* of each newcomer requires 0.08 milliseconds.

Invitee: The invitee performs *Invitation Collection* ($Icoll$) procedure, whose running time is linearly dependent on the number of required invitations, i.e., t. As such, the invitee's running time for $Icoll$ is shown in Figure 6.1 for the threshold values of 1-10. In this diagram, we included the time for the verification of invitations' correctness proof as part of the $Icoll$ procedure as well. In particular, the running time of $Icoll$ is dominated by $t \cdot t_{POIC_{verify}} + (t-1) \cdot t_{agg}$, where t is the threshold, $t_{POIC_{verify}}$ the time to authenticate each invitation, and t_{agg} is the time required for

	SetUp	Tgen	Igen	Ivrfy	Reg
Server	1.6 s	4.24 ms	-	6.5 ms	0.08 ms
Inviter	-	-	27.5 ms	-	-

Table 6.1: Running time of the server and the inviter. (s: seconds, ms: milliseconds)

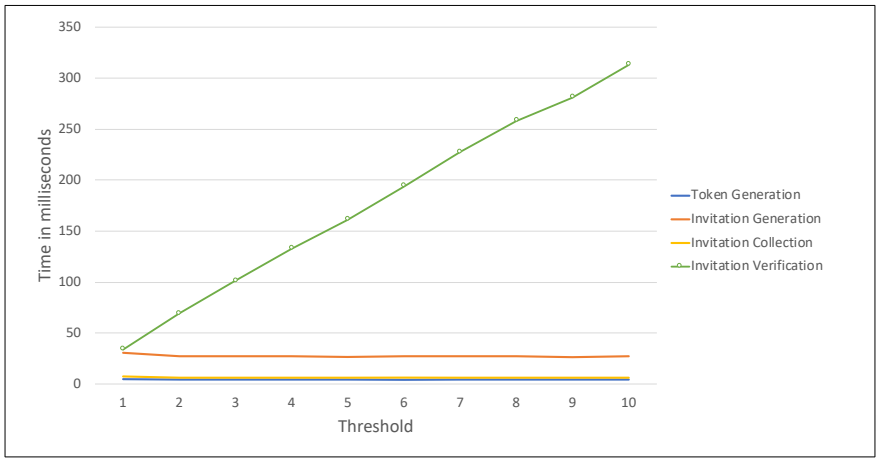

Figure 6.1: The invitee's running time.

homomorphic aggregation of two individual invitations.

Inviter: The inviter is only involved in the execution of the *Invitation Generation* algorithm for which he spends 27.5 milliseconds.

6.1.2 Communication Complexity

The asymptotic communication complexity among the parties is constant in the security parameter. However, we additionally measure the concrete communication complexity as the number of bits exchanged between parties. The communication complexity for the $Tgen$ protocol to transfer a Token with two group elements of size 2048 bits is 4096 bits (= 0.512 KB). For the invitation generation protocol, the inviter exchanges the invitation (consisting of 3 group elements) together with the ZKPOIC (with 8 group elements). Hence the total communication complexity of $Igen$ is 22528 bits (= 2.81 KB). The invitee submits the invitation letter (with 3 group elements)

together with the token (of size 4096 bits) to the server for the sake of registration which results in 10240 bits (= 1.28 KB) communication complexity.

6.1.3 Storage

The storage requirement of each entity is measured based on the number of bits that the party needs to retain locally. The server holds a signature and encryption key pairs, hence requires 20896 bits (2.612 KB) of storage. Moreover, the server saves the description of the polynomial of degree $t-1$ with t coefficients and their corresponding commitments which approximately results in $2t \cdot 2048$ bits of storage requirement at the server. The invitee only needs to keep its share of the master value which is of size 2046 bits (0.25 KB). For the invitee, no local storage is required.

6.2 Security

In this section, we first prove the special soundness and honest verifier zero-knowledge property of our proposed Σ protocol for proof of invitation correctness (POIC), followed by the ideal functionality F^R_{POIC} corresponding to the zero-knowledge version of POIC (Section 6.2.1). Next, we provide security definitions for inviter anonymity and invitation unforgeability, and then prove the security of Anonyma (Sections 6.2.2 and 6.2.3). In Section 6.2.4, we prove the security of AnonymaX for which we supply a new security definition capturing invitation unforgeability in the cross-network invitation based systems.

6.2.1 Proof of Invitation Correctness

Soundness:

Consider two valid transcripts (A,$B = (B_1, B_2)$,C,e,Z_1,Z_2,Z_3) and (A,$B = (B_1, B_2)$,C, e^*,Z_1^*,Z_2^*,Z_3^*), where $e \neq e^*$, $Z_1 \neq Z_1^*$, $Z_2 \neq Z_2^*$, and $Z_3 \neq Z_3^*$, then we extract δ_i, r and s_i as explained below. Since both transcripts are accepting we

have $A \cdot \gamma_i^e = g^{Z_1}$ and $A \cdot \gamma_i^{e^*} = g^{Z_1^*}$. We divide both sides of equalities and obtain

$$g^{Z_1 - Z_1^*} = \gamma_i^{e-e^*} = g^{s_i \cdot (e-e^*)} (\text{mod } p) \tag{6.1}$$

Thus, $Z_1 - Z_1^* \equiv s_i \cdot (e - e^*)$ *mod* q. It follows that $s_i = \frac{Z_1 - Z_1^*}{e-e^*}$. To extract δ_i and r we proceed as follows. We know that $B_1 \cdot e\delta_{i,1}^e = \omega^{Z_2} \cdot h^{Z_3}$ as well as $B_1 \cdot e\delta_{i,1}^{e^*} = \omega^{Z_2^*} \cdot h^{Z_3^*}$. Dividing both sides of equalities results in

$$\begin{aligned} e\delta_{i,1}^{e-e^*} &= \omega^{Z_2 - Z_2^*} \cdot h^{Z_3 - Z_3^*} (\text{mod } p) \\ \omega^{\delta_i (e-e^*)} \cdot h^{r(e-e^*)} &= \omega^{Z_2 - Z_2^*} \cdot h^{Z_3 - Z_3^*} (\text{mod } p) \end{aligned} \tag{6.2}$$

As such, it follows that $\delta_i = \frac{Z_2 - Z_2^*}{e-e^*} \text{ mod } q$ and $r = \frac{Z_3 - Z_3^*}{e-e^*} \text{ mod } q$.

Special honest verifier zero knowledge:

We construct a PPT simulator *Sim* which is given $\tau_i, e\delta_i = (e\delta_{i,1}, e\delta_{i,2}), \gamma_i, \omega$ and e and generates an accepting transcript. It selects Z_1, Z_2, Z_3 at random and constructs $A = \frac{g^{Z_1}}{\gamma_i^e} \text{ mod } p$ and $B_1 = \frac{\omega^{Z_2} \cdot h^{Z_3}}{e\delta_{i,1}^e} \text{ mod } p$ and $B_2 = \frac{g^{Z_3}}{e\delta_{i,2}^e} \text{ mod } p$ and $C = \tau_i^{-e} \cdot B \cdot e\delta_{i,1}^e \cdot h^{-Z_3} \cdot \omega^{Z_1} \text{ mod } p$. *Sim* outputs $(A, B = (B_1, B_2), C, e, Z_1, Z_2, Z_3)$. It is immediate that the probability distribution of $(A, B, C, e, Z_1, Z_2, Z_3)$ and a real conversation between honest prover and honest verifier are identical.

Zero-knowledge POIC (ZKPOIC):

We apply the method given in [97] to our Σ protocol to convert it to a zero-knowledge proof system. Let F_{POIC}^R (given in Equation 6.3) demonstrate the security guarantees of the resultant ZKPOIC over the relation R that we defined in Equation 5.14.

$$F_{POIC}^R((X, W), X) = (\perp, R(X, W)) \tag{6.3}$$

F_{POIC}^R shall be run by a trusted third party. X refers to the statement whose correctness is to be proven, i.e., $X = (\tau_i, e\delta_i, \gamma_i, \omega)$ contains the content of an individual invitation letter $(\tau_i, e\delta_i)$ as well as the commitment to the inviter's master share, i.e., γ_i, and the token ω. The witness W, which is only known to the prover, is (s_i, r, δ_i).

The ideal functionality F^R_{POIC} receives a common input X from the prover and the verifier as well as the private input W from the prover. F^R_{POIC} outputs to the verifier whether X and W fit into the relation R.

6.2.2 Inviter Anonymity

An invitation-based system protects inviter anonymity if an invitee with t inviters can authenticate himself to the server without disclosing the identity of his inviters to the server. In the extreme situation where a corrupted server also manages to control $t-1$ inviters of an invitee, the inviter anonymity should guarantee that the identity of the remaining non-colluding inviter remains protected against the server and other inviters. The coalition of the server and $t-1$ inviters is the most powerful adversary against inviter anonymity. In the following, we present the formal definition of inviter anonymity as well as a formal security proof of inviter anonymity of Anonyma.

Security Definition:

We model inviter anonymity as a game denoted by $InvAnonym_A(\lambda)$ played between a challenger and an adversary. The challenger acts as the invitee as well as the uncorrupted members of the system. On the other hand, the adversary plays as the server as well as arbitrary many corrupted members. The challenger is to register the invitee into the system while $t-1$ inviters of the invitee are controlled by the adversary and the remaining inviter is under the control of the challenger. As such, the adversary issues $t-1$ invitations on behalf of the corrupted inviters. Then, the adversary selects two indexes u_0, u_1 corresponding to two uncorrupted members. The challenger selects one of them randomly as u_b, where $b \in \{0,1\}$, to be the other inviter. The challenger issues an invitation from u_b for the invitee and combines it with the $t-1$ invitations issued by the adversary. The final invitation letter is submitted to the adversary (who also plays the role of the server). The challenge of the adversary is to guess a bit b indicating the index of the uncorrupted inviter. If the adversary cannot guess that index with more than a negligible advantage, then the system provides

inviter anonymity. The formal definition follows.

Inviter Anonymity Experiment $InvAnonym_A(\lambda)$

1. The adversary is given the security parameter 1^λ. It acts as the server and hands over $Param$ to the challenger.

2. The adversary registers arbitrary many users to the system. The adversary instructs the challenger to register honest users through the Reg protocol. U_h and U_c contain the indices of the honest and corrupted members, respectively.

3. (a) The adversary outputs the index of two honest inviters $u_0, u_1 \in U_h$.

   (b) The adversary, acting as the server, generates a token $Token$ for the invitee with index $j^* \in U_h$.

   (c) The adversary specifies a set of $t-1$ indices $I_c \subset U_c$ to be the corrupted inviters. For every $i \in I_c$, the adversary engages with the challenger in the execution of the $Igen$ protocol using $Token$ as the input. As the result, the invitee (i.e., the challenger) obtains a set of $t-1$ invitations denoted by $\{Inv_i\}_{i \in I_c}$.

4. (a) The challenger selects a bit value $b \leftarrow \{0,1\}$. The challenger runs the $Igen$ protocol over $Token$ to issue an invitation from u_b for the invitee. Let Inv_b be the result.

   (b) The challenger runs $Icoll$ using $\{Inv_i\}_{i \in I_c} \cup Inv_b$ and $Param$ and generates an invitation letter $InvLet$. The challenger attempts to register to the system by sending $InvLet$ to the adversary.

5. The adversary guesses a bit b'.

6. The output of the game is 1 if $b == b'$, 0 otherwise.

Definition 2 *An invitation-based system has inviter anonymity if for every probabilistic polynomial time adversary A there exists a negligible function negl(.) such that:*

$$Pr[InvAnonym_A(\lambda) \ = 1] = \tfrac{1}{2} + negl(\lambda)$$

At a high level, in Anonyma, the anonymity of the inviter holds due to the soundness of the proposed ZKPOIC (zero-knowledge proof of invitation correctness) and the security of the pseudo-random number generator (i.e., PRG). Below, to give an insight into how ZKPOIC can protect inviter anonymity, we draw a situation where the lack of ZKPOIC would immediately break inviter anonymity. Then, by relying on the F^R_{POIC} hybrid model for our proof, we relate the inviter anonymity of Anonyma to the security of the deployed PRG.

Recall that, as defined in the game, the adversary controls the server and $t-1$ inviters of the invitee. Due to the employed ZKPOIC, the invitee is assured that the inviters are not able to deviate from the protocol descriptions and hence would have to use their real master shares for the invitation generation. This implies that the master value S shall be reconstructed correctly as the output of $Icoll$. Therefore, as the result of the registration of the invitee (step 4.b from $InvAnonym_A(\lambda)$ experiment), the server obtains $InvLet = (T = \omega^{S+\Delta}, e\Delta)$ out of which the adversary can learn S and Δ. According to the Shamir secret sharing scheme, although the adversary knows $t-1$ shares that are used for the reconstruction of S, the remaining contributing shareholder can be any of the existing members, and hence the inviter anonymity is guaranteed. Now, consider that the inviters are not required to prove the correctness of their invitations. The $t-1$ corrupted inviters use zeros instead of their real master shares for invitation generation, i.e., $s_i = 0$ for $i \in I_c$. Then, the server obtains $w^{S'}$ with the following value: $S' = \sum_{i \in I_c} s_i.B_i + s_{u_b}.B_{u_b} = s_{u_b}.B_{u_b}$. The adversary can simply try the combinations of master shares s_{u_0} and s_{u_1} with B_0 and B_1, respectively and figure out the remaining inviter's index (in practice, the possible number of values is linear in the number of non-colluding inviters, which is the number of registered users). This trivially breaks inviter anonymity, which follows from

the lack of ZKPOIC.

As we discussed before, due to ZKPOIC all the invitations issued for the invitee are guaranteed to be well-structured (and their correctness are proven during $Igen$). Thus, the execution of $Icoll$ by the invitee would lead to a valid invitation letter of the form $InvLet = (\omega^{S+\Delta}, e\Delta)$ where S is the server's master value, $e\Delta$ is the encryption of ω^{Δ} and $\Delta = \delta^* + \sum_{i \in I_c} B_i \cdot \delta_i + B_{u_b} \cdot \delta'$ (δ^* is the masking value added by the invitee, δ' is the non-colluding inviter's masking value resulted from a PRG and B_{u_b} is the Lagrange coefficient computed based on the index of the non-colluding inviter). The adversary may get some idea about the identity of the non-colluding inviter by extracting the Lagrange coefficients from the Δ value (Lagrange coefficients are the function of inviters' indices). Two cases may occur. If the random values δ' and δ^* are selected truly at random, then we know that Δ is also a random value and conveys nothing about the Lagrange coefficient B_{u_b}. Though, if δ' and δ^* are the output of a PRG then the adversary may have advantages to extract the Lagrange coefficients. We denote the adversary's advantage by ϵ. If ϵ is non-negligible, it implies that we can distinguish between a PRG and a random number generator hence we break the security of the PRG. In the following, we provide a formal proof.

Theorem 3 *Anonyma provides inviter anonymity in F_{POIC}^{R} hybrid model (as defined in Equation 6.3), assuming that PRG is a secure pseudo-random number generator.*

Proof: We reduce the inviter anonymity of Anonyma to the security of the employed PRG. If there exists a PPT adversary A who breaks the inviter anonymity of Anonyma with non-negligible advantage, then we can construct a PPT adversary B who distinguishes between a random number generator and a pseudo-random number generator with the same advantage of A. Assume A's success probability is

$$Pr[InvAnonym_A(\lambda) \ = 1] = \frac{1}{2} + \epsilon(\lambda) \tag{6.4}$$

B runs A as its subroutine to distinguish the pseudo-random number generator from the truly random number generator. B is given a vector of values in Z_q denoted by $\vec{\delta} = (\delta', \delta'')$ and aims at specifying whether $\vec{\delta}$ is selected truly at random or is the

output of a *PRG*. B invokes A as its subroutine and emulates the game of inviter anonymity for A as follows. If A succeeds then B realizes that $\vec{\delta}$ is pseudo-random, otherwise random.

1. B is given the security parameter 1^λ and a vector of two values denoted by $\vec{\delta} = \{\delta^{'}, \delta^{''}\}$ s.t. $\delta', \delta'' \in Z_q$. Adversary A outputs $Param$ including ek, vk, and $(F_0, \cdots, F_{t-1})$.

2. A registers its own users. U_c contains the indices of corrupted members. Also, A instructs B to register users into the system. Let U_h indicate the set of indices registered by B. For each user $i \in U_h$, B obtains a share s_i and verifies its correctness by checking whether g^{s_i} is equal to $\prod_{j=0}^{t-1} F_j^{i^j}$.

3. (a) A outputs two indices $u_0, u_1 \in U_h$.

   (b) A outputs a token $Token = (\eta, j^*, \omega)$. j^* is the index of the invitee in U_h.

   (c) A specifies a set of $t-1$ incides $I_c \subset U_c$ to be the corrupted inviters. For every $i \in I_c$, and $Token$, A engages in $Igen$ with the callenger. A outputs $Inv_i = (\tau_i, e\delta_i)$ and contacts F_{POIC}^R with the input of $(\tau_i, e\delta_i, \gamma_i, \omega)(s_i, r_i, \delta_i)$. B acting as F_{POIC}^R, accepts or rejects A's proof by verifying whether $(\tau_i, e\delta_i, \gamma_i, \omega)$ and (s_i, r_i, δ_i) fit into the relation R as defined in Equation 5.14. Note that at this step B can learn all the $t-1$ master shares of corrupted inviters, i.e., $\{s_i\}$ for $i \in I_c$.

4. (a) B selects a random bit b. B uses the $Token$ and runs $Igen$ to create an invitation letter from u_b as $Inv_{u_b} = (\tau_{u_b}, e\delta_{u_b}) = (\omega^{s_{u_b}+\delta^{'}}, Enc_{ek}(\omega^{\delta^{'}}))$ (δ' is given from the distinguish-ability game of PRG)).

   (b) B runs $Icoll$ over $\{Inv_i\}_{i \in I_c} \cup Inv_b$, sets $\delta^* = \delta^{''}$ and computes
   $T = \omega^{\delta^*} \cdot \tau_{u_b}^{B_{u_b}} \cdot \prod_{i \in I_c} \tau_i^{B_i}$
   and
   $e\Delta = Enc_{ek}(\omega^{\delta^*}) \cdot Enc_{ek}(\omega^{\delta^{'}})^{B_{u_b}} \cdot \prod_{i \in I_c} e\delta_i^{B_i}$.

The value of $e\Delta$ will be equal to $Enc_{ek}(\omega^{\delta^* + \delta^{'} \cdot B_{u_b} + \sum_{i \in I_c} \delta_i \cdot B_i})$. B_i and B_{u_b} denote the Lagrange coefficients as defined in Equation 5.8. B submits $InvLet = (T, e\Delta)$ to the adversary A.

5. A outputs a bit $b^{'}$.

6. If $b = b^{'}$ then B outputs 0, otherwise 1.

Note that B follows all the steps as indicated in the $InvAnonym_A(\lambda)$ game and hence is indistinguishable from a real challenger. This means that B also runs in polynomial time (as there is no rewind). B ties the $InvAnonym_A(\lambda)$ game to the security of PRG by embedding δ' and $\delta^{''}$ (the challenge of PRG game) as the randomness δ_{u_b} (used by the non-colluding inviter for the invitation generation), and the value of δ^* (used by the invitee in $Icoll$ execution), respectively. Below is the success probability analysis of the reduction.

Let $\vec{\delta}$ be a truly random vector. Once the adversary decrypts $e\Delta$ he obtains

$$\omega^{\delta^{''} + \Gamma}$$

where

$$\Gamma = \delta^{'} \cdot B_{u_b} + \textstyle\sum_{i=1}^{t-1} \delta_i \cdot B_i$$

Γ is a function of inviters indices due to the presence of Lagrange coefficients whereas $\delta^{''}$ is a random value completely independent of inviters' indices. If $\vec{\delta}$ is a random vector then $\delta^{''}$ is also a random value from $\mathbb{Z}_q$. Therefore, in $\omega^{\delta^{''} + \Gamma}$, Γ is indeed masked with $\delta^{''}$ ($\delta^{''} + \Gamma \mod q$ is a completely random element of $\mathbb{Z}_q$). By this masking, Δ (i.e., $\delta^{''} + \Gamma$) becomes completely independent of the Lagrange coefficients and A has no advantage to infer the identity of the uncorrupted inviter. Thus, A's advantage is exactly $\frac{1}{2}$ i.e.,

$$Pr[B(\vec{\delta} \leftarrow Z_q) = 1] = Pr[b = b^{'}] = \frac{1}{2} \tag{6.5}$$

but if $\vec{\delta}$ is the output of a PRG then

$$Pr[r \leftarrow \{0,1\}^{\lambda} : B(\vec{\delta} = PRG(r)) = 1] = Pr[b = b^{'}] = \frac{1}{2} + \epsilon(\lambda) \tag{6.6}$$

where $\frac{1}{2} + \epsilon(\lambda)$ is the success probability of A (as assumed in our proof in Equation 6.4). By combining Equations 6.5 and 6.6 we have

$$|Pr[r \leftarrow \{0,1\}^{\lambda} : B(\vec{\delta} = PRG(r)) = 1] - Pr[B(\vec{\delta} \leftarrow Z_q) = 1]| = \epsilon(\lambda) \tag{6.7}$$

Equation 6.7 corresponds to the security definition of PRG (see Equation 5.5). Thus, if $\epsilon(\lambda)$ is non-negligible, then the distinguisher B can distinguish a PRG from a random generator. This contradicts with the security definition of PRG. Therefore, $\epsilon(\lambda)$ must be negligible according to the PRG definition. This concludes the security proof of inviter anonymity of Anonyma. ■

6.2.3 Invitation Unforgeability

In an invitation-based system, the invitation unforgeability indicates that people who do not have enough inviters (less than t) should not be able to join the system. Hence, no adversary can forge invitations on his own. Next, we present a formal definition for invitation unforgeability together with a formal security proof of Anonyma.

Security Definition:

We define the following game denoted by $InvUnforge_A(\lambda)$ running between a challenger and an adversary. The adversary controls a set of $t-1$ members. The rest of the users denoted by I_h are controlled by the challenger. Also, the adversary has oracle access to the token generation $Tgen(sk, j)$, invitation generation $Igen(., s_i, Param)$ for $i \in I_h$, and invitation verification $Ivrfy(., ., Param, dk)$ algorithms. Finally, the adversary wins the game if it manages to register to the system successfully, using a token that was not queried from the invitation generation oracle. The success of the adversary asserts that the invitations are forgeable. Otherwise, the system provides

invitation unforgeability. We remark that by giving the adversary oracle access to the invitation generation algorithm we aim to capture the *non-exchangability* of invitations. This oracle access is equivalent to having an adversary who eavesdrops the communication of other invitees and inviters and wishes to forge an invitation over its token.

Invitation Unforgeability experiment $InvUnforge_A(\lambda)$:

1. The adversary specifies a set I_c consisting of the index of $t-1$ users to be controlled by the adversary.

2. The challenger runs the setup algorithm and outputs $Param$ to the adversary.

   The next steps (3-6) are the learning phase of the adversary and can be run in an arbitrary order.

3. (a) The adversary registers a corrupted user $i \in I_c$ to the system. The adversary can repeat this part for every user $i \in I_c$.

   (b) The adversary instructs the challenger to register an honest user to the system. I_h shall contain the index of honest members.

4. The adversary asks the challenger to issue a token. The challenger generates a token for the next available index j. This step may be repeated polynomially many times upon the adversary's request. Q^{Token} holds the set of tokens queried by the adversary.

5. The adversary queries invitation verification function on the invitations of his own choice. The challenger runs the $Ivrfy$ algorithm and responds accordingly.

6. The adversary has oracle access to the $Igen$ algorithm. That is, the adversary asks the challenger to use a particular token and generate

an invitation from an honest member. As such, the adversary specifies the index $i \in I_h$ of an honest member together with a valid $Token_j \in Q^{Token}$. Then, the challenger issues an individual invitation by running $Inv_{i,j} = Igen(Token_j, s_i, Param)$ and gives the output to the adversary. Let $Q^{Inv} = \{(Token_j, Inv_{i,j})\}$ be the set of tokens together with the individual invitations queried by the adversary.

7. The adversary outputs an invitation letter $InvLet$ for a valid token $Token^{'} \in Q^{Token}$ for which no query exists in Q^{Inv}.

8. If the output of $Ivrfy(InvLet, Token^{'}, Param, dk)$ is accepted then the game's output is 1 indicating the adversary's success, 0 otherwise.

Definition 3 *An invitation-based system has invitation unforgeability if for every probabilistic polynomial time adversary A there exists a negligible function negl(.) such that:*

$$Pr[InvUnforge_A(\lambda) \ = 1] = negl(\lambda)$$

At a high level, in order for the adversary to be able to win the game, it has to compute ω^{*S} for some token $Token' = (\tau, j, \omega^*)$ where $Token'$ does not belong to Q^{Inv}. Through the oracle accesses, the adversary learns a set of individual invitations $Q^{Inv} = \{(Token_j, Inv_{i,j})\}$ where $Inv_{i,j} = (\tau_{i,j} = \omega_j^{s_i+\delta_{i,j}}, e\delta_{i,j} = Enc_{ek}(\omega_j^{\delta_{i,j}}))$. The $Inv_{i,j}$ carries no useful information regarding the master value S to the adversary as the master share s_i is masked through a random value $\delta_{i,j}$. There is no way for the adversary to get to learn $\delta_{i,j}$ unless with decryption of $e\delta_{i,j}$ which is not possible as the adversary lacks the decryption key dk. Alternatively, the adversary may attempt to combine invitations issued under different tokens to obtain a valid invitation under a new token $Token'$. This is impossible due to the CDH problem. That is, given $\tau_{i,j}$ $(= \omega_j^{s_i+\delta_{i,j}})$ and $\omega^*(= g^x)$, the adversary must compute $\tau_{i,j}^x$ which corresponds to a solution to the CDH problem. Similarly, the knowledge of ω^* and $F_0 = g^S$ (from

$Param$) does not help in making a valid invitation letter since computing ω^{*S} is equivalent to solving the CDH problem. That is, given $\omega^* = g^x$ and $F_0 = g^S$, the adversary shall compute $g^{x \cdot S} = \omega^S$. In the theorem and proof given below, we reduce $InvUnforge_A(\lambda)$ game to the CDH problem.

Theorem 4 *Anonyma satisfies invitation unforgeability as defined in Definition 3, in F^R_{POIC} hybrid model, given that the signature scheme Sig is existentially unforgeable under chosen message attack, and Computational Diffie-Hellman problem is hard relative to group G.*

Proof: If there exists a PPT adversary A who breaks the invitation unforgeability with the non-negligible advantage then we can construct a PPT adversary B who solves the CDH problem with non-negligible advantage.

Let ϵ denote the probability of success of A. B interacts with the CDH challenger and also runs A as its subroutine. B is given $G, q, g, X = g^x, Y = g^y \in G$ for which B is supposed to compute $Z = g^{x \cdot y}$.

1. A outputs a set of $t-1$ indices as I_c to be the index of members under its control.

2. B runs the setup algorithm and generates the encryption and signature key pairs (ek, dk) (sk, vk) as normal. To set up Shamir secret sharing scheme, B performs as follows. B sets $F_0 = Y$ (recall that F_0 is the commitment to master value S thus $F_0 = g^{f(0)} = g^S$; this implies that B does not know the master value S since it is the discrete logarithm of Y (i.e., y), which is selected by the CDH challenger). B selects $t-1$ random values $s_{i \in I_c} \leftarrow Z_q$ to be the master shares of the corrupted members. Also, B computes $\gamma_i = g^{s_i}$ for $i \in I_c$. Recall that the share of the master value for the i^{th} user is $f(i)$, thus by setting the master shares of corrupted parties, B fixes $t-1$ points of polynomial f as $f(i) = s_i$ for $i \in I_c$. These $t-1$ points together with F_0, which is indeed $g^{f(0)}$, will fix polynomial f since the degree of f is $t-1$. Next, B interpolates Y i.e., $(g^{f(0)})$ and $\{(i, \gamma_i)\}_{i \in I_c}$, and computes the commitments $F_1, \cdots, F_{t-1}$ (where

$F_1 = g^{a_1}, \cdots, F_{t-1} = g^{a_{t-1}}$) over the coefficients of polynomial f [132] (where $f = S + a_1 \cdot x + ... + a_{t-1} \cdot x^{t-1}$).

Note that B does not obtain the exact coefficients of the polynomial f (i.e., a_i values) but only computes the commitments $F_i = g^{a_i}$. This is sufficient for B to simulate the role of the server since it only needs to publicize the commitments of the polynomial and not the exact coefficients.

B outputs $param = (G, q, g, ek, vk, (F_0, ..., F_{t-1}))$, as well as the security parameter 1^λ. Note that B also records the master shares of corrupted members i.e., $\{(i, f(i))\}_{i \in I_c}$ to use in the registration phase.

3. (a) A registers a corrupted user to the system (a user with the index $i \in I_c$). As such, B sends $f(i)$ (which was computed during the setup protocol) to A.

   (b) A instructs B to register an honest user to the system. Note that B cannot generate the master shares of honest users since it does not know the coefficients of the function f. However, since it is a local calculation for B, this shortage remains unnoticed to A. B records the index of honest user inside I_h.

4. A has oracle access to the token generation $Tgen$. Initially, B draws a random value $j^* \in [1, P(\lambda)]$ where $P(\lambda)$ is the upper-bound on the number of adversary's queries to $Tgen$. B answers the queries of A for $Tgen$ as follows. For the j^{*th} query, B sets $Token_{j^*} = (Sign_{sk}(j^*||X), j^*, X)$ (X was given to B from the CDH game) and inserts $(j, X, \perp)$ into Q^{Token}. Otherwise, B selects a random $r_j \in_R Z_q$, sets $\omega_j = g^{r_j}$ and outputs $Token_j = (Sign_{sk}(j||\omega_j), j, \omega_j)$. B records (j, ω_j, r_j) inside Q^{Token}.

5. The adversary queries the invitation verification function on the invitation letters and tokens of his own choice i.e., $InvLet = (T, e\Delta)$ and $Token = (\eta, j, \omega_j)$.

B first authenticates the token against the signature verification key. If not verified, B outputs $reject$ to A. Also, if $\omega_j = X$, then B aborts. Otherwise,

- If $Token == Token^*$, then B aborts the expriment.
- If $Token \neq Token^*$, B proceeds as follows. Due to the lack of master value S, B has to run different than the normal $Ivrfy$ algorithm. B decrypts $e\Delta$ as $\omega_j^{\Delta} = Dec_{dk}(e\Delta)$. Next, B retrieves the record of (j, ω_j, r_j) corresponding to ω_j from Q^{Token} and checks whether $F_0^{r_j} \cdot \omega_j^{\Delta} \stackrel{?}{=} T$ and responds to A accordingly. The right side of this equality check is the same as line 3 of $Ivrfy$ Algorithm (Algorithm 7) since

$$F_0^{r_j} \cdot \omega_j^{\Delta} = g^{S \cdot r_j} \cdot \omega_j^{\Delta} = g^{r_j \cdot S} \cdot \omega_j^{\Delta} = \omega_j^{S} \cdot \omega_j^{\Delta} \tag{6.8}$$

6. A has oracle access to $Igen$ algorithm. A outputs an index i of an honest member together with a $Token_j = (\eta, j, \omega_j)$. B first authenticates the token against the signature verification key. If successful, then, it attempts issuing an invitation. Notice that B cannot generate the invitation by following $Igen$ since it does not have the master share of honest users i.e., s_i for $i \in I_h$. B performs differently to compute a valid invitation as explained next. B retrieves the record (j, ω_j, r_j) from Q^{Token} (if the token is valid and has a correct signature from the server then it must be already queried by the adversary and hence should exist in Q^{Token}, otherwise, the signature forgery happens which is not possible due to the security of the underlying signature scheme). B computes $\gamma_i = \prod_{j=0}^{t} F_j^{i^j}$ as well as selects a random value $\delta_i \in_R Z_q$. Then, B constructs $\tau_{i,j} = \gamma_i^{r_j} \cdot g^{\delta_i \cdot r_j}$ where r_j is the discrete logarithm of ω_j in base g. It is immediate that $\tau_{i,j}$ is well-structured since

$$\tau_{i,j} = \gamma_i^{r_j} \cdot g^{\delta_i \cdot r_j} = (g^{s_i})^{r_j} \cdot (g^{\delta_i})^{\cdot r_j} = (g^{r_j})^{s_i} \cdot (g^{r_j})^{\delta_i} = \omega_j^{s_i} \cdot \omega_j^{\delta_i} = \omega_j^{s_i + \delta_i} \tag{6.9}$$

B constructs $e\delta_{i,j}$ as $Enc_{ek}(\omega_j^{\delta_i})$ and outputs $Inv_{i,j} = (\tau_{i,j}, e\delta_{i,j})$ to A.

Finally, B acts as F_{POIC}^{R} and waits for A's message asking verification of

$(\tau_{i,j}, e\delta_{i,j}, \gamma_i, \omega_j)$ for which B responds *accept* to A. B keeps the set of individual invitations and their tokens queried by A in $Q^{Inv} = \{(Token_j, Inv_{i,j})\}$.

7. The adversary outputs an invitation letter $InvLet = (T, e\Delta)$ for a token $Token^{'}$ for which no query has been made from $Igen$ i.e., $Token^{'} \notin Q^{Inv}$.

8. B verifies whether the $Token^{'}$ is correctly signed under sk. If not, B outputs $\perp$ to CDH challenger. Otherwise:

   - If $Token^{'} \neq Token^{*}$, B outputs $\perp$ to the CDH game.
   - If $Token^{'} == Token^{*}$, B outputs $T \cdot Dec_{dk}(e\Delta)^{-1}$ to the CDH challenger. In fact, if A constructs $InvLet$ correctly, we expect that $T = \omega^{*S+\Delta}$ and $e\Delta = Enc(\omega^{*\Delta})$. Given that $X = g^x = \omega^*$ and $Y = g^y = g^S$, we have

$$T \cdot Dec_{dk}(e\Delta)^{-1} = (\omega^*)^{S+\Delta} \cdot (\omega^*)^{\Delta^{-1}} = (\omega^*)^S = (g^x)^y = g^{xy} \quad (6.10)$$

   $g^{x.y}$ is the solution to the given CDH problem.

This is immediate that B runs in polynomial time. The index j^* chosen by B at step 4 represents a guess as to which $Tgen$ oracle query of A will correspond to the token of eventual invitation letter forgery output by A. If this guess is correct, then A's view while running with B is identical to $InvUnforge_A(\lambda)$ game.

When B guesses correctly and A outputs a forgery, then B can solve the given instance of CDH. Assume that A's advantage in $InvUnforge_A(\lambda)$ game is ϵ. The probability that B wins is

$$\begin{aligned} Pr[\text{B wins}] &= Pr[B(G, q, g, X = g^x, Y = g^y) = g^{x \cdot y}] \quad (6.11) \\ &= Pr[\text{A wins} \wedge (Token^{'} = Token^{*})] \\ &= Pr[\text{A wins } | Token^{'} = Token^{*}] \cdot Pr[Token^{'} = Token^{*}] \\ &\geq \epsilon \cdot \frac{1}{Poly(\lambda)} \end{aligned}$$

The last equality holds since the number of queries made by A is at most $P(\lambda)$ (P is polynomial in 1^λ), hence, the probability $Token^* = Token^{'}$ is $\frac{1}{Poly(\lambda)}$. Note that

due to the signature unforgeability, A cannot create a valid token outside of the set of queried tokens i.e., $\notin Q^{Token}$.

Assuming that ϵ is non-negligible, B also wins with non-negligible probability. This contradicts with the hardness of the CDH problem. Hence A's success probability in $InvUnforge_A(\lambda)$ must be negligible. This concludes the proof. ■

6.2.4 Security of AnonymaX

Inviter Anonymity: The inviter anonymity of AnonymaX can be defined identically to the experiment of $InvAnonym_A(\lambda)$. The challenger shall control the honest members, i.e. U_h and invitee whereas the adversary will have the control of $S_{inviter}$ and all the *registration* servers S_j together with the corrupted members I_c which shall constitute $t-1$ inviters of the invitee. AnonymaX meets inviter anonymity due to the similar proof supplied for Anonyma. Without loss of generality and for the sake of simplicity, we consider only one *registration* server to exist, though the extension of proof for multiple *registration* servers is straightforward. In particular, the following theorem holds for AnonymaX with one *inviter* server and one *registration* server.

Theorem 5 *AnonymaX provides inviter anonymity in F^R_{POIC} hybrid model (as defined in Equation 6.3), assuming that PRG is a secure pseudo-random number generator.*

Proof Sketch: Given that a PPT adversary A' can break the inviter anonymity game for AnonymaX with non-negligible advantage, we can construct an adversary B' to distinguish between a PRG and a truly random number generator. The internal code of adversary B' shall be identical to the simulator B in proof of Theorem 3. The only difference is in the $SetUp$ phase where the challenger outputs two pair of encryption keys $(ek_{reg}, ek_{inviter})$ among which only ek_{reg} will be used throughout the experiment.
Invitation Unforgeability: Recall that invitation unforgeability guarantees that a corrupted invitee with an insufficient number of inviters would not be able to join the system. In a cross-network invitation-based system, the invitation unforgeability should additionally hold for the *registration* service. That is, if Alice does not have

enough inviters from the *inviter* system, she should not be able to successfully register to the *registration* service.

Note that in a cross-network invitation-based system, invitation unforgeability cannot be defined for the case that $S_{inviter}$ acts against S_{reg}, i.e. $S_{inviter}$ wants to generate valid invitations for the invitee of its choice to join the *registration* service. This is trivial since $S_{inviter}$ is able to register arbitrary many users into its own system (i.e., *inviter* system). Then, every subset of t registered users of *inviter* service will consequently be able to issue invitations and register arbitrary many users into the *registration* service. Note that this is not a limitation imposed by our design and rather is implicit in any cross-network invitation-based system.

In the invitation unforgeability game as defined in $XInvUnforge_A(\lambda)$, we consider N *registration* servers which all accept invitations from the members of one *inviter* server. The adversary plays on behalf of $t-1$ corrupted users of the *guest* service and a subset of *registration* servers. The challenger controls the honest users, i.e. I_h of the *inviter* service together with $S_{inviter}$ and some of the uncorrupted *registration* servers. At the end of the game, the mission of adversary as a corrupted invitee with an insufficient number of inviters is to successfully register to one of the honest *registration* servers S_{j^*} controlled by the challenger.

In $XInvUnforge_A(\lambda)$, we index *registration* servers as S_j, where $1 \leq j \leq N$, and the *inviter* server as S_0. The set of servers controlled by the adversary are denoted as set C. We assume H indicates the set of un-corrupted *registration* servers. The set of members of *inviter* service is denoted by $U_{inviter}$. I_c represents the set of $t-1$ corrupted members in the *inviter service* whereas I_h contains the indices of the honest members. We have $U_{inviter} = I_h \cup I_c$. We prefix the algorithms with its executing entity, e.g. we write $S_j.Tgen$ to show the invocation of the token generation algorithm at the server j.

Invitation Unforgeability experiment $XInvUnforge_A(\lambda)$ for cross-network invitation based system:

1. The adversary specifies a set $I_c \subset U_{inviter}$ consisting of the index of $t-1$ users to be under his control.

2. The challenger runs $Setup$ for all $S_j \in H$ and outputs $Param_j$ to the adversary. The adversary outputs $Param_j$ for $j \in C$. The next steps (3-6) are the learning phase of the adversary and can be run in an arbitrary order.

3. (a) The adversary registers a corrupted user $i \in I_c$ to the *inviter* system. The adversary repeats this part for every user $i \in I_c$.

   (b) The adversary instructs the challenger to register an honest user i to the *inviter* system where $i \in I_h$.

4. The adversary has Oracle access to the $S_j.Tgen$ for $j \in H$. Also, the adversary generates a $Token$ for a user $i \in I_h$ from S_j where $j \in C$ and hands over to the challenger.

5. The adversary has oracle access to $S_j.XIvrfy$ for $j \in H$.

6. The adversary has oracle access to the $Igen$ algorithm. That is, the adversary specifies the index l of an honest member i.e., $l \in I_h$ and a server index $j \in H \cup C$ together with a $Token$ issued by S_j.

   The challenger generates an individual invitation by running $Inv_l = Igen(Token, s_l, Param_j)$ and gives the output Inv_l to the adversary. Let $Q_j^{Inv} = \{(Token, Inv_l)\}$ be the set of tokens together with the individual invitations queried by the adversary to be generated by the l^{th} user for the j^{th} service.

7. The adversary outputs an invitation letter $InvLet$ together with token $Token'$ for the registration in the j^{*th} server where $j^* \in H$. There should not be any issued invitation using $Token'$ in $Q_{j^*}^{Inv}$.

8. If the output of $XIvrfy(InvLet, Token^{'}, Param_{inviter}, Param_{S_{j^*}}, dk_{S_{j^*}})$ is accepted, then the game's output is 1 indicating the adversary's success, 0 otherwise.

Definition 4 *An cross-netwrok invitation-based system has invitation unforgeability if for every probabilistic polynomial time adversary A there exists a negligible function negl(.) such that:*

$$Pr[XInvUnforge_A(\lambda) = 1] = negl(\lambda)$$

Theorem 6 *AnonymaX satisfies invitation unforgeability as defined in Definition 4, in F^R_{POIC} and F^R_{PODL} hybrid model, given that the signature scheme Sig is existentially unforgeable under chosen message attack, and Computational Diffie-Hellman problem is hard relative to group G.*

At a high level, The reduction idea between CDH problem and $XInvUnforge_A(\lambda)$ of AnonymaX is similar to $InvUnforge_A(\lambda)$. However, in AnonymaX, the simulator B additionally is able to extract the CDH solution during the $Tgen$ and $XIvrfy$ which we explain below. B is given $X = g^x$ and $Y = g^y$ from the CDH challenger and sets Y as the commitment to the master value S. B also guesses at which query of $Tgen$ A will succeed to forge a valid $InvLet$. B sets the value ω of that token to X. B can solve the CDH challenge if

- A creates a token with the value of X for which A must prove the knowledge of the discrete logarithm x. Then B outputs Y^x as the CDH solution.

- The adversary A queries $XIvrfy$ with a valid invitation letter $InvLet$ over the token with $\omega = X$, then B extracts the CDH solution. The $InvLet$ is of the form $\omega^S = g^{xy}$ which is the solution to the CDH problem.

- A submits a valid invitation letter using the token X. That is of the form $\omega^S = g^{xy}$ which is the solution to the CDH problem.

A may also win by forging a token (i.e., a signature) on behalf of an honest *registration* server $\in H$. However, since the signature scheme is secure, the probability of signature forgery is negligible.

Proof: If there exists a PPT adversary A who breaks the invitation unforgeability of AnonymaX with non-negligible advantage, then we can construct a PPT adversary B who solves the CDH problem with non-negligible advantage.

Let ϵ denote the probability of success of A. B interacts with the CDH challenger and also runs A as its subroutine. B is given the security parameter 1^λ, $G, q, g, X = g^x, Y = g^y \in G$ for which B is supposed to compute Z s.t. $Z = g^{x \cdot y}$.

1. A outputs a set of $t-1$ indices as I_c to be the index of members under its control.

2. For every S_j $j \in H$, B runs the setup algorithm and generates the encryption and signature key pairs (ek_{S_j}, dk_{S_j}) (sk_{S_j}, vk_{S_j}) as normal. $Param_{S_j}$ will be (ek_{sk_j}, vk_{S_j}).

   Similarly, B sets up an encryption and signature key pairs for $S_{inviter}$ as $(ek_{inviter}, dk_{inviter})$ and
   $(sk_{inviter}, vk_{inviter})$, respectively. As for the initialization of Shamir secret sharing scheme, B performs as follows. B sets $F_0 = Y$ (recall that F_0 is the commitment to master value S thus $F_0 = g^{f(0)} = g^S$; this implies that B does not know the master value S since it is the discrete logarithm of Y (i.e., y), which is selected by the CDH challenger). B selects $t-1$ random values $s_{i \in I_c} \leftarrow Z_q$ to be the master shares of the corrupted members. Also, B computes $\gamma_i = g^{s_i}$ for $i \in I_c$. Recall that the share of the master value for the i^{th} user is $f(i)$, thus by setting the master shares of corrupted parties, B fixes $t-1$ points of polynomial f as $f(i) = s_i$ for $i \in I_c$. These $t-1$ points together with F_0, which is indeed $g^{f(0)}$, will fix polynomial f since the degree of f is $t-1$. Next, B interpolates Y i.e., $(g^{f(0)})$ and $\{(i, \gamma_i)\}_{i \in I_c}$, and computes the commitments $F_1, \cdots, F_{t-1}$ (where $F_1 = g^{a_1}, \cdots, F_{t-1} = g^{a_{t-1}}$) over the coefficients of polynomial f [132] (where

$f = S + a_1 \cdot x + ... + a_{t-1} \cdot x^{t-1}$). Note that B does not obtain the exact coefficients of the polynomial f (i.e., a_i values) but only computes the commitments $F_i = g^{a_i}$. This is sufficient for B to simulate the role of the *inviter* server since it only needs to publicize the commitments of the polynomial and not the exact coefficients.

B outputs $param = (G, q, g, ek_{inviter}, vk_{inviter}, (F_0, ..., F_{t-1}))$, as well as the security parameter 1^λ to the adversary. Note that B also records the master shares of corrupted members i.e., $\{(i, f(i))\}_{i \in I_c}$ to use in the registration phase.

3. (a) A registers a corrupted user to the system (a user with the index $i \in I_c$). As such, B sends $f(i)$ (which was computed during the *SetUp* protocol) to A.

   (b) A instructs B to register an honest user to the system. Note that B cannot generate the master shares of honest users since it does not know the coefficients of the function f. However, since it is a local calculation for B, this shortage remains unnoticed to A. B records the index of the honest user inside I_h.

4. A has oracle access to token generation i.e., $S_{inviter}.Tgen$ and $S_j.Tgen$ for all $j \in H$. B keeps the set of tokens queried by A for each server S_j inside Q_j^{Token}. Initially, B draws two random values $j^* \in [1, N]$ (to be the guess over the index of the honest *registration* server for which the adversary comes up with the invitation letter forgery) and $l^* \in [1, P(\lambda)]$ where $P(\lambda)$ is the upper-bound on the number of adversary's queries to $Tgen$ for each of the servers.

   - If j is equal to j^*, and if this is the l^* query to $S_{j^*}.Tgen$ then B returns

     $$Token^* = (Sign_{sk_{S_{j^*}}}(l^*||X), l^*, X)$$

     X was given to B from the CDH game. B plays the role of F_{PODL}^R, receives the verification request of (G, q, g, X) from the adversary and outputs *accept* to the adversary. B inserts $(X, \perp)$ into $Q_{j^*}^{Token}$.

- If $j \neq j^*$, and assuming this is the l^{th} query of adversary to $S_j.Tgen$, B selects a random $r \in_R Z_q$, sets $\omega = g^r$ and outputs $Token = (Sign_{sk_{S_j}}(l||\omega), l, \omega)$. B plays the role of F^R_{PODL}, receives the verification request of (G, q, g, ω) from the adversary and outputs *accept* to the adversary. B inserts (ω, r) to Q^{Token}_j.

The adversary may generate a token $Token = (\eta, l, \omega)$ for a user $l \in I_h$ from S_j where $j \in C$ and hands over to the challenger. The adversary contacts F^R_{PODL} i.e., the challenger B and hands over $((G, q, g, \omega), r)$. B checks whether $g^r = \omega$ and accepts or rejects the token accordingly. Also, B verifies the signature η against the verification key of S_j and accepts or rejects the token accordingly. If the verification passed successfully, B stores (ω, r) in Q^{Token}_j. If $\omega == X$ (the CDH challenge), and the token is accepted, then B outputs Y^r to the CDH challenger.

5. The adversary queries $S_j.XIvrfy(InvLet, Token, Param_{inviter}, dk_{S_j})$ for $j \in H$ on the invitation letters and tokens of his own choice i.e., $InvLet = (T, e\Delta)$ and $Token = (\eta, l, \omega)$. B runs $XIvrfy$ algorithm and responds accordingly. If the output of $XIvrfy$ is not reject and if $j = j^*$ and $Token = Token^*$ (i.e., $\omega = X$), then B outputs $T \cdot Dec_{dk_{S_{j^*}}}(e\Delta)^{-1}$ to the CDH game.

$$T \cdot Dec_{dk_{S_{j^*}}}(e\Delta)^{-1} = X^{S+\Delta} \cdot X^{-\Delta} = X^S = g^{xS} = g^{xy} \tag{6.12}$$

6. A has oracle access to $Igen$ algorithm. A asks the challenger to generate an invitation from the honest member i for the *registration* server $j \in H \cup C$ using a $Token = (\eta, l, \omega)$. B first authenticates the token against the signature verification key of S_j. If not verified, B outputs $reject$ to A. Also, if $\omega = X$, then B aborts. Otherwise, B attempts issuing an invitation. Notice that B cannot generate the invitation by following $Igen$ since it does not have the master share of honest users i.e., s_i for $i \in I_h$. B performs differently to compute a valid invitation as explained next. B computes $\gamma_i = \prod_{v=0}^{t} F_v^{i^v} = g^{s_i}$ (the second

equality holds due to Equation 5.12) as well as selects a random value $\delta_i \in_R Z_q$. Then, B constructs $\tau_i = \gamma_i^r \cdot g^{\delta_i \cdot r}$ where r is the discrete logarithm of ω in base g. It is immediate that τ_i (to be the first component of the invitation letter) is well-structured since

$$\tau_i = \gamma_i^r \cdot g^{\delta_i \cdot r} = (g^{s_i})^r \cdot (g^{\delta_i})^{\cdot r} = (g^r)^{s_i} \cdot (g^r)^{\delta_i} = \omega^{s_i} \cdot \omega^{\delta_i} = \omega^{s_i + \delta_i} \quad (6.13)$$

B constructs $e\delta_i$ as $Enc_{ek}(\omega^{\delta_i})$ and outputs $Inv_i = (\tau_i, e\delta_i)$ to A.

Finally, B acts as F^R_{POIC} and waits for A's message asking verification of $(\tau_i, e\delta_i, \gamma_i, \omega)$ for which B responds *accept* to A. B keeps the set of individual invitations and their tokens queried by A for each server S_j in $Q_j^{Inv} = \{(Inv_i, Token)\}$.

7. The adversary outputs an invitation letter $InvLet = (T, e\Delta)$ for a valid token $Token'$ issued by $S_{j' \in H}$ i.e., $Token' \in Q_{j'}^{Token}$ for which no query has been made from $S_{j'}.Igen$ i.e., $Token' \notin Q_{j'}^{Inv}$.

8. B verifies whether the $Token'$ is correctly signed under $sk_{j'}$. If not, B outputs $\perp$ to CDH challenger. Otherwise:

   - If $j' \neq j^*$ or $Token' \neq Token^*$ B outputs $\perp$ to the CDH game.
   - If $j' = j^*$ and $Token' = Token^*$, B outputs $T \cdot Dec_{dk_{S_{j^*}}}(e\Delta)^{-1}$ to the CDH challenger. In fact, if A constructs $InvLet$ correctly, we expect that $T = X^{S+\Delta}$ and $e\Delta = Enc_{ek)S_{j^*}}(X^{\Delta})$. Given that $X = g^x$ and $Y = g^y = g^S$, we have

   $$T \cdot Dec_{dk_{S_{j^*}}}(e\Delta)^{-1} = (X)^{S+\Delta} \cdot (X)^{\Delta^{-1}} = (X)^S = (g^x)^y = g^{xy} \quad (6.14)$$

   $g^{x.y}$ is the solution to the given CDH problem.

This is immediate that B runs in polynomial time. The index j^* and l^* chosen by B at step 4 represents a guess as for which server S_{j^*} and to which $S_{j^*}.Tgen$ oracle

query of A will correspond to the token of eventual invitation letter forgery output by A. If this guess is correct, then A's view while running with B is identical to $XInvUnforge_A(\lambda)$ game.

When B guesses correctly and A outputs a forgery, then B can solve the given instance of CDH. Assume that A's advantage in $XInvUnforge_A(\lambda)$ game is ϵ. The probability that B wins is

$$\begin{aligned} Pr[\text{B wins}] &= Pr[B(G,q,g,X=g^x,Y=g^y)=g^{x\cdot y}] \\ &= Pr[\text{A wins} \wedge (Token^{'} = Token^{*} \text{ AND } j' = j^{*})] \\ &= Pr[\text{A wins } | Token^{'} = Token^{*} \text{ AND } j' = j^{*}] \\ &\cdot Pr[Token^{'} = Token^{*} \text{ AND } j' = j^{*}] \\ &\geq \epsilon \cdot \frac{1}{Poly(\lambda)} \cdot \frac{1}{N} \end{aligned} \tag{6.15}$$

The last equality holds since the number of queries made by A is at most $Poly(\lambda)$ (i.e., polynomial in λ), and there are N *registration* servers (honest and corrupted), hence, the probability $Token^{*} = Token^{'}$ and $j' = j^{*}$ is at least $\frac{1}{Poly(\lambda)} \cdot \frac{1}{N}$.

A may attempt to forge a token on behalf of $S_{j'}$ for which it has obtained an individual invitation from an honest user for the registration in one of the corrupted *registration* servers. However, due to the signature unforgeability, A cannot create a valid token outside of the set of queried tokens i.e., $\notin Q_j^{Token}$ for all $j \in H$. Also, all the queries to $Igen(Token = (\eta, i, \omega), s_l, Param_{S_j})$ where $l \in I_h$ and $j \in C$ are answered if the given $Token$ is generated by the corrupted server S_j correctly i.e., $Token \in Q_j^{Tgen}$ which means that the adversary has passed ZKPODL successfully (knows the DL of the ω). The presence of zero-knowledge proof will prevent the adversary from using a token of an honest server since the adversary does not know the DL of ω due to the hardness of discrete logarithm assumption. Without ZKPODL, the adversary can win the $XInvUnforge_A(\lambda)$ (A takes ω from one of the tokens $Token = (\eta, l, \omega)$ in $Q_{j^*}^{Tgen}$ and then generates a valid token $Token'' = (Sign_{sk_j}(\omega), i, \omega)$ at step 4 from a corrupted server $S_{j \in C}$. Next, A queries $Igen(Token'', s_l, Param_{S_j})$ for $l \in I_h$ and obtains a valid invitation $Inv_l = (\tau_l, e\delta_l)$. Given Inv_l and dk_{sk_j}, the adversary would

be able to open $e\delta_l$ to ω^{δ_l} hence can construct its t^{th} valid individual invitation as $Inv_t = (\tau_l, Enc_{ek_{S_{j^*}}}(\omega^{\delta_l}))$ for a $Token = (\eta, l, \omega) \in Q_{j^*}^{Tgen}$. The adversary combines Inv_t with $t-1$ invitations issued by the $t-1$ corrupted inviters under its control and hands over an intact $InvLet$ to B).

Assuming that ϵ is non-negligible, B also wins with non-negligible probability. This contradicts with the hardness of the CDH problem. Hence A's success probability in $InvUnforge_A(\lambda)$ must be negligible. This concludes the proof. ■

Chapter 7

INTEGRITA: PROTECTING VIEW-CONSISTENCY IN ONLINE SOCIAL NETWORK WITH FEDERATED SERVERS

7.1 Introduction

In the centralized architecture of OSNs, the collaborative data sharing environments (e.g., Facebook group pages or users walls) are prone to the view inconsistency problem. The content of the shard data gets updated by multiple users independently. Since users are not aware of each other's updates, a central corrupted server may hide user updates from each other and serve users with divergence views of the data. To formalize the problem of *view consistency*, we will use the term shared *object* to indicate a collaborative data-sharing environment (such as a Facebook-like wall or a group-page) on which a set of users are authorized to perform read and write operation. We denote the shared object by $\mathcal{D}$. Each object is comprised of smaller units called *post* which have content and an author. Similar to Frientegrity [51], we assume posts are uploaded to the object one after another hence no concurrency will happen in users write operations. We denote the k^{th} version of $\mathcal{D}$ by $\mathcal{D}_k = \{post_1, \cdots, post_k\}$ namely, an ordered sequence of k posts. Likewise, the view of a user u toward the i^{th} version of $\mathcal{D}$ is comprised of a sequence of i posts seen by that user i.e., $View_i^u = \{post'_1. \cdots post'_i\}$. The view consistency concerns two aspects of users' views. First, to ensure that the corrupted storage provider cannot forge any post i.e., all the $post'_j \in View_i^u$ are issued by authorized users. This can be immediately addressed by deploying digital signatures. The second aspect regards the *history integrity* of an *object* $\mathcal{D}$ which is less recognized and studied in the literature. This second property assures that the view

of all the users (obtained through their interaction with the corrupted provider) contains an identical and intact sequence of posts i.e., no post is dropped or misplaced. More formally, for the i^{th} version of object, for every authorized user u and for all $j \in [1, i]$, $post'_j$ is equal to $post_j$ where $post'_j \in View_i^u$, and $post_j \in \mathcal{D}_i$.

Related Work: The view consistency problem is addressed in the literature by two types of solutions: communication-based solutions which are sought by the centralized architectures, and replication-based solutions deployed by the distributed designs. We elaborate on each solution type next.

In a centralized architecture with non-communicating users, the best achievable level of view consistency is fork consistency [133], first defined by [134]. The fork-consistency is a weaker form of view-consistency in which a corrupted provider is able to split users into disjoint sets (to fork them) and serve each set with a distinct view though the provider is forced to serve each set with a consistent view of the operations performed by the users of the same set. Identification of the forked views can only happen through users' communication. That is users must regularly communicate their views of the *object* (e.g., a wall) with all the other authorized users (e.g., friends) to catch any view inconsistency. This approach would not be practical considering that a user of an OSN like Facebook has 338 friends on the average[1]. Hence, each user needs to communicate with almost 338*338= 114244 other users to monitor the view consistency of her wall and her friends' walls. Addressing view consistency using communication-based solution is sought in the context of secure OSNs [51, 26], and cloud computing [135, 136, 137, 138].

The replication-based solutions are deployed in peer-to-peer OSNs [139] (as a distributed OSN), Authenticated data structures (ADS) [140, 141, 142, 143, 144, 145] as well as Byzantine fault-tolerant protocols [146, 147]. The idea is to designate multiple entities for the storage of the *object* and let all the read and write operations happen through all of them. In specific, the shared object (or some authenticated-metadata associated with it) must be replicated on $f+1$ entities considering f of them may act maliciously. Having only one honest repository suffices to always retrieve the

intact content of the object. Replication based solutions are not efficient concerning the storage overhead since f extra copies of the object must be stored in the OSN.

Integrita: In Integrita [25], we aim to achieve the best of both aforementioned solutions: a method to achieve view consistency which is replication-free as well as communication-free, and an approach where users do not have to communicate their views out-of-band. In particular, N federated servers run by multiple authorities are utilized, and the storage of shared object is split among them (rather than replicated). Each server gets to serve only a part of the shared object which has no overlap with the parts stored by other servers. This way, we cope with the storage overhead imposed by replication-based proposals as only one instance of the shared object (and its associated meta-data) is maintained in the entire design. We let servers be malicious/Byzantine entities who may act arbitrarily, collude, compromise the view consistency by dropping, tampering with, and forging posts. Nonetheless, our approach guarantees that as long as one server does not collude with the other servers, the view consistency is preserved. We assume that users shall act honestly and tend to achieve a consistent view. A similar assumption is sought in prior studies [148] as well. Note that in Integrita, we are not concerned about the privacy of the posts; one can address it using the well-practiced techniques like encryption [45, 149].

Integrita provides the following features.

- **q-Detectable-Consistency:** In Integrita, we introduce a new level of view consistency called *q-Detectable-Consistency* in which the views of users toward the *object* (i.e., wall) cannot diverge for more than q sequence of posts without detection. That is, if a user uploads a post, either her post correctly becomes a part of the shared object and being seen consistently by all the other users, or she can catch any inconsistency within the next q posts. The value of q depends on the total number of servers and the number of posts on the shared object at the time of write operation. A thorough analysis of this relation is provided in Section 7.5.3. Moreover, we provide a formal definition of q-detectable consistency together with a security proof in Section 8.1.1.

- **Communication-free:** In contrast to the fork-based systems, our fork detection mechanism relies neither on the users' collaboration in sharing their views in each operation nor an out-of-band communication. Every user is able to verify any server-side equivocation regarding her performed operation, alone (without relying on the presence of other users).

- **Replication-free:** Our solution for view consistency is storage efficient as we do not replicate the shared object over all the servers. That is, one copy of the *object* is present in the entire system and each server retains only (an identical) portion of it. Our numerical analysis asserts that by using Integrita, an OSN like Facebook with 2.3 billion monthly active users [2] saves up to 2344 Terabyte storage per year (deploying 20 servers) compared to the replication-based approach.

  Note that each of the N storage providers is modeled as a data-center that would take care of a portion of the data assigned to it. While Integrita does not depend on data replication to achieve view consistency, this does not contradict with the replication of data for the sake of *availability*. Namely, each data-center shall deploy its replication mechanism to maintain the availability of the data assigned to it. However, due to Integrita, the amount of data assigned to each data-center is $\frac{1}{N}$ of the data that would be otherwise assigned by using a replication-based solution.

- **Efficient verification:** In Integrita, each read and write operation is associated with a proof of correctness which must be verified by the user. While the creation and transmission of proof in Integrita are handled in a distributed fashion, resultant overhead for users and the servers concerning the amount of data transmission, the communication and computation complexity is identical to the centralized fork-consistent counterparts [9, 26] (while Integrita enforces a higher level of consistency). In Section 8.1, we further discuss that distributing the storage among multiple servers not only does not degrade the experience

of user's interaction with the system concerning the performance but also lowers the computational and storage overhead on each server (compared to both centralized design and the replication-based proposals).

- **Cross-server Communication-free**: While the storage of the shared object is distributed among N servers, servers do not need to communicate or to coordinate to resolve the users' read and write operations. Instead, all the communication happens solely between the users and the servers.

7.2 Related Works

The concern of view consistency similarly is investigated in the context of centralized OSN, peer-to-peer (p2p) OSN, cloud storage, Byzantine fault-tolerant protocols, and authenticated data structures. In the following, we elaborate on these proposals and compare them with Integrita.

Centralized OSN: In centralized architecture of OSNs, frientegrity [9] and SPORC [26] address the view consistency by achieving fork-consistency on a shared data. In the fork*-consistent system, while a corrupted provider is able to fork the users into disjoint sets, he is forced to serve each set with a consistent view of operations performed by the users of the same set. This is enabled since users embed their views of the object history in each post they insert. Thus, as soon as the server forks view of two users, he cannot show their operations to each other without risking detection. Users can also detect the inconsistency of their views by exchanging them out of the band. The main shortcoming of fork*-consistent systems is that the server's equivocation remains undetected till users happen to contact out of the band. Thus, to ensure view consistency, users must regularly communicate their views of the shared object. This approach would not be practical. For example, on Facebook, each user has on the average 338 friends [3] which would all have access to her wall as a shared object. Hence, each user needs to communicate with almost 338*338= 114244 other users to monitor the view consistency of her wall and her friends' walls.

Peer-to-Peer OSN: In a p2p OSN, there is no central server running the system and instead, the individual users called peers contribute a part of their computation and storage power to the system. The social networking services are enabled in a distributed manner relying on shared resources. As such, the storage of users' data is also distributed among the existing peers. The view consistency in p2p OSNs is usually addressed through replication or by leveraging users' trust. In the latter case, the object owner (e.g., owner of a wall) stores and serves her wall by herself or replicates on some trusted peers like her friends. Subsequently, the view consistency is guaranteed due to the trustworthiness of storage peers [150, 151, 152, 15, 6, 153]. However, if the storage responsibility is spread over the p2p network and the storing peers are untrusted, then view consistency is met through replication [139]. In particular, suppose f as the fraction of potential dishonest peers, the object (or some units of the object like a post) should be replicated on $f + 1$ peers to ensure that at least one honest peer is among the replicas. Each requester reads each post from all the $f + 1$ replicas and identifies the latest content (e.g., using a version number). However, such a solution results in storage overhead and communication complexity which grow linearly by f. Other studies in the context of p2p OSNs also utilize replication but for the sake of data availability [5, 154, 153]. Namely, the storing nodes are supposed to be trusted and always serve the intact contents when available.

Byzantine Fault-Tolerant Protocols and Cloud Storages: In BFT protocols, a service is to be given to a set of clients while the execution of the service concerning the sequence of requests appears identical to all clients (and this sequence preserves the temporal order of non-concurrent operations). Enabling consistency, BFT protocols also seek the replication-based solution [146, 147] where they deploy several servers each keeping a replica of the state of the intended service. Byzantine fault-tolerant systems behave correctly when no more than f out of $3f + 1$ replicas fail [146].

Similar to the centralized OSNs, the best level of view consistency in the context of cloud storage is fork-consistency [135, 136, 137, 138, 133] which is due to the presence

of a corrupted service provider and non-communicating users. Addressing the fork issue, cloud storage platforms utilize replication over multiple servers [155, 146].

Authenticated Data Structure: In the Authenticated data structures, a data owner outsources her data to multiple untrusted repositories. The outsourced data is modeled by a data structure that enables performing queries on the data in a verifiable and authenticated manner. Repositories, on behalf of the data owner, are responsible to answer queries of users on the data structure and hand them with a proof of the validity of the answer. The same data structure is replicated over all the repositories and repositories need to keep themselves updated with the data owner in the case of update [140, 141, 142, 143, 145]. As such, one can assume view consistency of ADSs is guaranteed through replication which is not storage efficient.

7.3 System Model

7.3.1 Model

Integrita is comprised of N *OSN servers* denoted by S_1,...,S_N (each operated by a distinct authority), a set of users $U_1, ..., U_T$ with read/write access to a shared *object* that is stored at the servers side. We assume all the users have an identical read/write access, though one can enforce more fine-grained access control using the technique proposed by [9]. The N servers are responsible to store the *object*s, serve the users' read and write requests.

The shared *object* is comprised of an ordered sequence of posts as well as is associated with an authenticated data structure that is to be kept at the servers' side. The storage of ADS and the object are divided among the servers where each server only holds a portion. This way, we avoid the space inefficiency of the replication by trading the strong consistency with the q-detectable consistency (in which the inconsistency may happen but would not last for more than q posts i.e., the inconsistency is detectable). Hence only one copy of the object and the associated ADS exist in the entire system.

To monitor the trustworthiness of servers in serving all the object posts con-

sistently to every authorized user, namely, preserving view consistency, each user maintains a local data structure that mirrors the state of that object at the time of reading/writing. Each time that the user reads the object, she has to check whether her local state is consistent with the current state of the object. To ensure q-detectable consistency, users also need to audit their updates on the profile at a certain point after their write operation to ensure that the update has become accessible to all the other users.

7.3.2 Security Goal

The security goal of Integrita is to achieve q-detectable consistency in which users are able to verify any server-side equivocation regarding their performed operation alone and without relying on the presence of other users. In other words, in a q-detectable consistent system, any inconsistency between users' views cannot remain undetected for more than q posts.

7.3.3 Adversarial Model

The servers are untrusted hence may act maliciously to compromise the integrity of profile history. This includes dropping a post, reordering posts, and showing users a different subset of posts. We assume that out of N servers, at least one server does not conspire with the rest of the servers. We treat confidentiality as an orthogonal issue to be addressed by encrypting the content of posts. Thus, our sole objective is protecting the users' view consistency.

7.4 Definitions and Preliminaries

7.4.1 Notations

We write $x \leftarrow X$ to denote picking an element x uniformly at random from set X. $a||b$ represents concatenation of a with b. $|A|$ stands for the number of elements in set A. The notations used in Integrita are provided in Table 7.1.

1^{λ}: The security parameter

$\perp$: Empty string

D: The shared object consisting of a sequence of posts

D_i: The shared object with i many posts

$post_i$: The i^{th} post of D

$View_i^u$: The view of user from the i^{th} version of D

N: The total number of servers

f: The number of malicious servers

S_i: The i^{th} server

U_i: The i^{th} user

vk_{U_i}: The signature verification key of the i^{th} user

T: The total number of users

TD_i: Tree digest, the root of the history tree constructed over D_i

H: The hash function

h_i: the hash of i^{th} post

L: The labeling function

l: The level of a node in the history tree

$N_{i,l}$: The node of history tree located at the l^{th} level of the insertion path of i^{th} post

$F(i,l)$: The storage assignment function for $N_{i,l}$

v: The history tree version

R: A set of post indices

$S_{F(p,l)}$: The storage provider of $N_{p,l}$

ek: The encryption key

$FList$: The list of authorized users

$SList$: The list of servers

$S_i.vk$: The signature verification key of the i^{th} server

$S_i.index$: The index of the i^{th} server

$S_i.Status$: The $Status$ of the i^{th} server

$Sign_{U_i}, Verify_{U_i}$: The execution of $Sign$ and $Verify$ algorithms using the signature key and verification key of the i^{th} user
σ_{U_i}: The signature generated by the i^{th} user
$Sign_{S_i}, Verify_{S_i}$: The execution of $Sign$ and $Verify$ algorithms using the signature key and verification key of i^{th} server
σ_{S_i}: The signature generated by the i^{th} server
p: The index of a post
σ: Signature
$last$: The index of the last post inserted to the object D
q: Audit threshold
$TP(.)$: The object transition point
DB: Server's database

Table 7.1: Notations used in Integrita

7.4.2 Definitions

Negligible: A function f is called negligible if for all positive polynomials p, there exists a constant C such that for every value $c > C$ it holds that $f(c) < \frac{1}{p(c)}$.

7.4.3 Preliminaries

Signature Scheme A signature scheme [126] Sig consists of three algorithms; key generation, sign and verify denoted by $Sig = (Gen, Sign, Vrfy)$. A pair of keys (sk, vk) is generated via $SGen$ where sk is the signature key and vk is the verification key. The signer signs a message m using sk by computing $\eta = Sign_{sk}(m)$. Given the verification key vk, a receiver of signature runs $Vrfy_{vk}(\eta, m)$ to verify.

A signature scheme $Sig = (Gen, Sign, Vrfy)$ is said to be existentially unforgeable under adaptive chosen message attack if $\forall$ probabilistic polynomial time adversary A,

there exists a negligible function $nelg(.)$ s.t. the following holds [100]:

$$Pr[(sk, vk) \leftarrow Gen(1^{\lambda}); (m, \sigma) \leftarrow A^{Sign_{sk}(.)}(vk) \\ \text{s.t. } m \notin Q \text{ and } Vrfy_{vk}(m, \sigma) = accept] = negl(\lambda) \quad (7.1)$$

$A^{Sign_{sk}(.)}$ indicates that adversary has oracle access to the signature algorithm. Q indicates the set of adversary's queries to the signature oracle.

7.5 Integrita System Design

In Integrita, we make use of an authenticated data structure called *history tree* to represent the shared *object* and to enable verifiable write and read operation for the users. That is, each read and write operation is associated with a proof through which the user can verify the authenticity of the operation result. The details of *object* representation is provided in Section 7.5.1. Representing a shared *object* using a *history tree* does not suffice to provide view consistency. We further discuss in Section 7.5.2 how to distribute the storage of shared *object* among N servers.

7.5.1 Shared object representation

The shared *object* $\mathcal{D}$ is treated as an ordered sequence of posts $\mathcal{D} = \{post_1, ..., post_M\}$. Each post shall be signed by its issuing user and can contain any type of data e.g., text or image. Concerning privacy, one can assume the content is encrypted and the decryption key is provided to all the other users.

The shared *object* is additionally attached to an authenticated data structure called *history tree* which is initially introduced by [148]. A history tree is an append-only data structure modeled by a variant of the Merkle hash tree. In Integrita, the leaves of the tree hold the hash of each post $post_i$. The intermediate nodes and root node store the hash of their children. In such a structure, the root essentially covers the entire content of the tree. The new posts can freely be added as the leaf nodes to the right side of the tree. For each newly added post, the value of intermediate nodes and the root shall be recalculated. A sample of history tree for a shared *object*

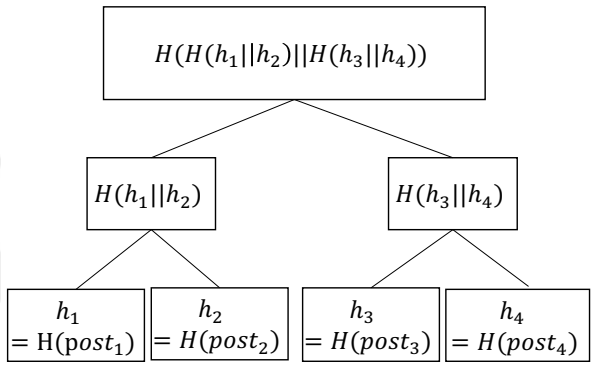

Figure 7.1: A history tree constructed for the *object* with 4 posts.

$\mathcal{D}$ with 4 posts is provided in Figure 7.1. Figure 7.2 represents the same tree after the insertion of $post_5$. We use the term of *Tree digest* or TD for short to refer to the history tree root and we write TD_i to indicate the content of the root after insertion of i^{th} post. We refer to the shared *object* with i posts as the i^{th} version of the shared *object*.

The history tree exhibits the following properties that are fundamental inefficiently preserving view consistency.

- Every tree digest TD_j uniquely defines a distinct ordered sequence of j posts. That is, for two identical sets of posts each with a different order e.g., $\mathcal{D}_3 = \{post_1, post_2, post_3\}$ and $\mathcal{D}'_3 = \{post_2, post_1, post_3\}$, their associated tree digests TD_3 and TD'_3 end up completely different. This is due to collision resistant property of the underlying hash function.

- Proof of membership: The occurrence of a particular post $post_i$ at position i in a tree digest TD_j where $i <= j$ is efficiently verifiable in $O(log(j))$. The proof of membership includes the sequence of values stored at the siblings of

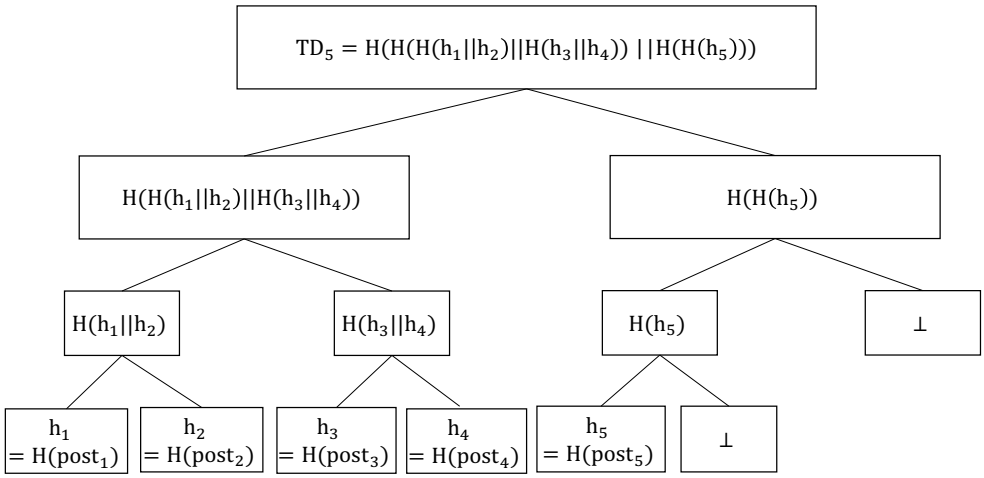

Figure 7.2: The tree of Figure 7.1 after the insertion of $post_5$.

the nodes (indicating whether it is a left or right sibling) on the path from the leaf node storing $post_i$ to the root TD_j. Given the proof, one can recompute the root as TD'_j and compare against TD_j. For example, as shown in Figure 7.3, to prove that TD_4 contains $post_3$ as its 3^{rd} post, the proof includes h_3, h_4, $H(h_1||h_2)$ and $H(H(h_5))$. The tree digest TD'_5 can be reconstructed recursively from the values included in the proof. If the computed value TD'_5 and the given tree digest TD_5 match, then $post_3$ is the 3^{rd} post of the *object*.

- Incremental Proof: Given two different tree digests TD_i and TD_j of the same shared *object*, where $i < j$, one can check whether the two tree digests make consistent claim about the past posts namely, whether TD_i and TD_j share the same history regarding $post_1, \cdots, post_i$. The incremental proof between version 2^{nd} and version 5^{th} of a shared *object* is shown in Figure 7.4. Let TD'_2 indicate the tree digests computed using the given proof. If $TD'_2 = TD_2$ then the incremental proof asserts the consistency of TD_2 and TD_5.

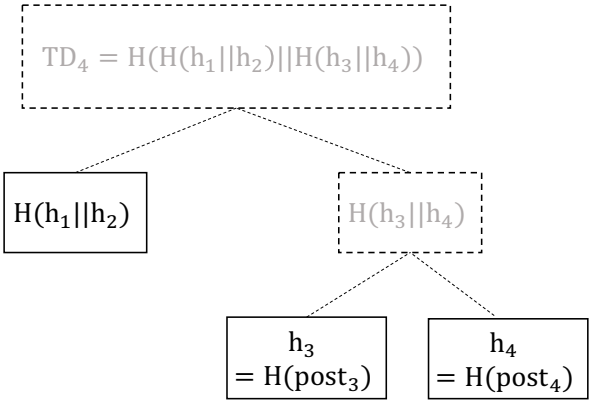

Figure 7.3: The membership proof of $post_3$ for version 4^{th} of the shared *object*.

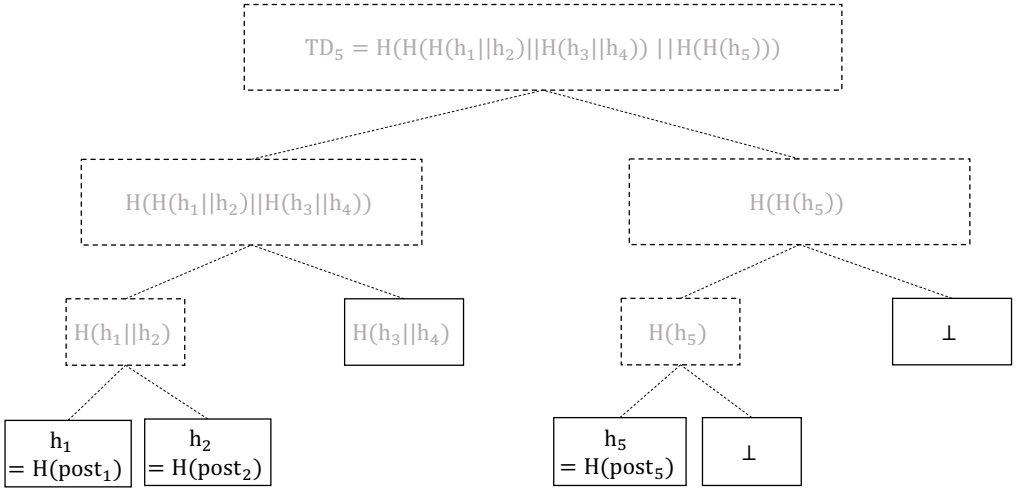

Figure 7.4: The incremental proof of the 2nd version of the shared *object* to the 5th version. The solid rectangles represent the proof. The gray parts are computable given the proof parts.

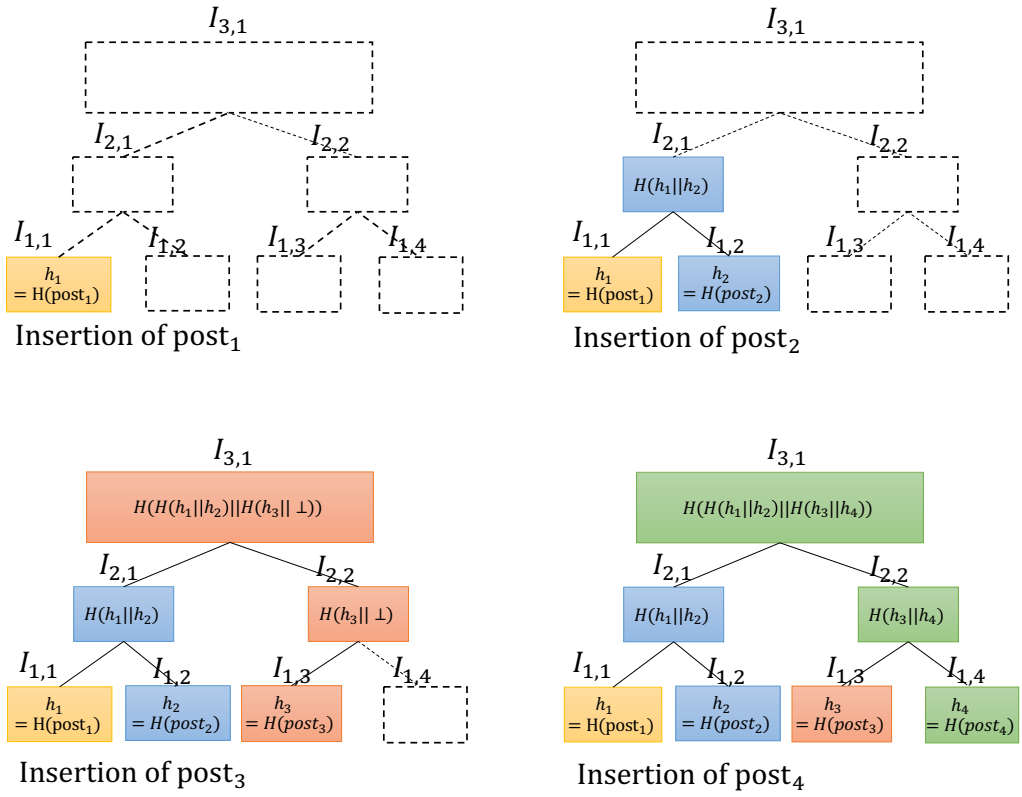

Figure 7.5: Insertion path of $post_1$, $post_2$, $post_3$, and $post_4$. Each insertion path indicated by a distinct color.

7.5.2 Distributed storage of the shared object

For the distributed storage of the shared *object* and its associated history tree, we proceed as follows. First, we define an insertion path of post i to be nodes of history tree whose hash values get altered while inserting post i to the tree. Figure 7.5 illustrates the insertion path of posts $1-4$ each with a different color. $I_{i,j}$ refers to the j^{th} node at level i. Since Merkle trees support logarithmic path lengths from the root to the leaves, the insertion path of i^{th} post will consist of $\lceil log(i) \rceil + 1$ nodes. For example, the insertion path $post_1$ is comprised of only one node $I_{1,1}$ whereas the insertion path $post_2$ consists of two nodes $I_{1,2}$ and $I_{2,1}$.

Note that insertion paths of $post_3$ and $post_4$ have two nodes in common namely, $I_{2,2}$ and $I_{3,1}$. However, the value of each of these nodes at the time of insertion of $post_3$ and $post_4$ are different e.g., $I_{2,2}$, on the insertion path of $post_3$ contains $H(h_3||\perp)$ whereas its value changes to $H(h_3||h_4)$ after insertion of $post_4$. Following this intuition, in Integrita, we treat each of these nodes separately and address them based on their location on the insertion path of each post. That is, each node is addressed with a pair of integers (i, l) as $N_{i,l}$ where i indicates post number and l stands for the level of node on the insertion path. The history tree of Figure 7.5 under the new addressing is demonstrated in Figure 7.6. Under the new addressing, node $I_{2,2}$ corresponds to $N_{3,2}$ and $N_{4,2}$ indicating its distinct values at the insertion of $post_3$ and $post_4$.

We further define a labeling function L to convert these pairs to a distinct numerical value. $L : \{1, ..., M\} \times \{1, ..., log(M)\} \rightarrow \{0, 1\}^*$ is a deterministic labeling function which receives the pair of (i, l) as defined above and returns back an integer label as given in Equation 7.2. We labeled nodes of Figure 7.6 and showed the result in Figure 7.7. We write $N_{i,l}$ and $N_{L(i,l)}$, interchangeably, e.g., $N_{4,2}$ and N_8 refer to the same node i.e., $N_{4,2} = N_8 = H(h_3||h_4)$.

$$L(i, l) = l + \sum_{j=1}^{i-1}(\lceil log(j) \rceil + 1) \tag{7.2}$$

Given the above labeling mechanism, the storage of each labeled node shall be assigned to a distinct server circularly. For this, we define a function F that receives the label of the node as *label* and returns the index of the server which is responsible for that node. N is the total number of servers.

$$F(i, l) = [L(i, l) \ mod \ N] + 1 \tag{7.3}$$

For example, in a system with 3 servers, the storage of $N_3, N_6, N_9, \cdots$ are given to the first server S_1. Nodes $N_1, N_4, N_7, \cdots$ are given to S_2 whereas S_3 gets to serve $N_2, N_5, N_8, \cdots$. Say it differently, the assignment of nodes to the servers follow a circular pattern where $\{N_1 \rightarrow S_1, N_2 \rightarrow S_2, N_3 \rightarrow S_3, N_4 \rightarrow S_1, N_5 \rightarrow S_2, N_6 \rightarrow$

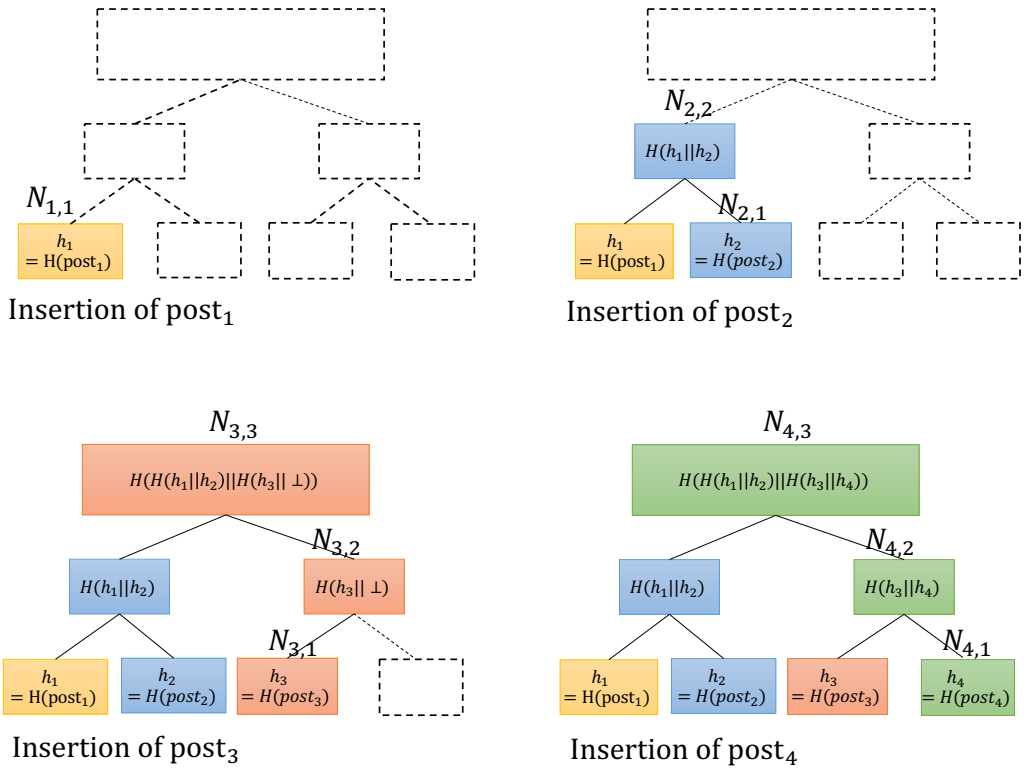

Figure 7.6: Insertion path of $post_1$, $post_2$, $post_3$, and $post_4$. Each insertion path indicated by a distinct color. Each node is addressed with a pair of integers (i, l) as $N_{i,l}$ where i indicates post number and l stands for the level of node on the insertion path.

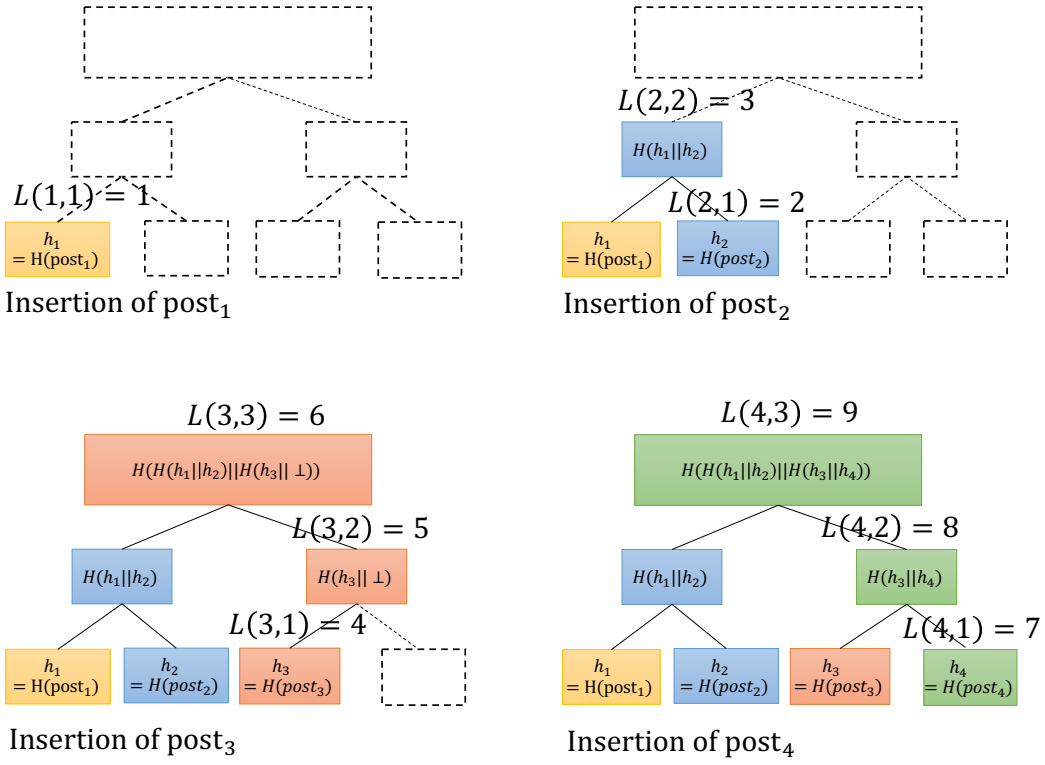

Figure 7.7: The labeling of insertion path of $post_1$, $post_2$, $post_3$, and $post_4$ using the labeling function L. Each insertion path indicated by a distinct color. The label of each node is indicated above it.

$S_3, \cdots\}$, $\rightarrow$ indicates the assignment. In section 8.2, we discuss how we enforce q-detectable-consistency using the circular distribution of storage of history tree among the servers.

Following our labeling method, we define and distinguish three types of nodes of the history tree.

- **Tree digest**: The root of tree after insertion of each post is called tree digest. In figure 7.6, nodes $N_{1,1}, N_{2,2}, N_{3,3}, N_{4,3}$ all represent the tree digests which are roots of the tree at the insertion time of $post_1, post_2, post_3$, and $post_4$, respectively. Given a post number i and the level of node on the insertion path as l, one can identify whether the node is tree digest if Equation 7.4 holds:

$$l = \lceil log(i) \rceil + 1 \tag{7.4}$$

- **Full node:** A full node is a node whose left and right sub-trees are full i.e., insertion of further posts will not alter the value of a full node. A node on the l^{th} level of insertion path of i^{th} post is full if the following relation (Equation 7.5) is met:

$$i \ mod \ 2^{l-1} = 0 \tag{7.5}$$

  In Figure 7.6, nodes $N_{1,1}, N_{2,1}, N_{3,1}, N_{4,1}, N_{4,2}$ and $N_{4,3}$ are all full.

- **Temporary node:** Nodes whose left or right sub-trees are not full are called temporary node. We call them temporary since the insertion of further posts will change their hash values. For example, nodes $N_{3,3}$ and $N_{3,2}$ in Figure 7.6 are temporary as there is an empty node in their right sub-trees corresponding to $post_4$. In general, $N_{i,l}$ is a temporary node if Equation 7.6 holds:

$$i \ mod \ 2^{l-1} \neq 0 \tag{7.6}$$

  The content of a temporary node can be reconstructed given the value of the highest full node in its left and its right sub-trees. For example, as shown in

Figure 7.7, the node N_5 as a temporary node can be reconstructed using the highest full node of its left sub-tree which is N_4 and the highest full node in its right sub-tree which is $\perp$. Likewise, the content of N_6 can be correctly recreated using N_3 and its highest full node at its right sub-tree which is N_4 i.e., $N_6 = H(N_3||N_5) = H(N_3||H(N_4|| \perp))$. Due to this property, the storage servers of temporary nodes will not save their hash values. Instead, the responsible servers just maintain some state information about the inserted post (see section 7.5.3).

Fetching proofs in a distributed manner: In Integrita, there is no central entity holding a global view of the shared *object* and its history tree. As such, dislike the centralized system where the server would create the membership and incremental proofs on its own, in Integrita, the user herself is responsible to determine the nodes on the proof path (using the labeling algorithm given in Equation 7.2) and their associated storage providers (using Equation 7.3) and fetch the hash values. Once hashes are fetched, the user can check the correctness of the proof as normal.

For the ease of explanation, we define the following function

$$\{(S_{F(p,l)}, (p,l))\} = ProofPath(v, R = \{i, \cdots, j\}) \tag{7.7}$$

that receives a version number v of the shared *object*, and a set $R = \{i, \cdots, j\}$ of a range of post indices. The function returns a set of pairs $(S_{F(p,l)}, (p,l))$ where $N_{p,l}$ is a node holding a value of the membership proof of $\{post_i, \cdots, post_j\}$ and $S_{F(p,l)}$ is the index of the corresponding storage provider. For example, one may call $ProofPath(7, \{5,6\})$ to find out the nodes located on the membership proof of $post_5$ and $post_6$ with respect to the version 7^{th} of the shared *object*. The proof, as illustrated in Figure 7.9, includes the values of N_9, N_{10}, N_{14}, and N_{18} and their corresponding storage servers S_1, S_2, S_3, and S_4. Note that since the temporary nodes are not saved in the system, the proof includes N_{18} instead of N_{19} which is a temporary node; given N_{18} the calculation of N_{19} is immediate. As we stated before, for a temporary node, its highest full node in its right and left sub-trees shall be fetched instead.

Note that an incremental proof between i^{th} and j^{th} version of the history tree involves the membership proof of $post_i$ and $post_j$. Thus, the function $ProofPath$ can be also used to find the nodes and the servers holding values of incremental proof. For example, finding the nodes on the incremental proof between 2^{nd} and 5^{th} version of the *object* requires a call to $ProofPath(5, \{2, 5\})$.

We additionally consider the existence of the following two functions that shall be run by the users:

- $True, False \leftarrow INCR.VF(TD_i, TD_j, proof)$: Given two tree digests TD_j and TD_k where $i < j$ it verifies whether $proof$ is a correct incremental proof between TD_i and TD_j. We write $TD_i \rightarrow TD_j$ to indicate that there exists an incremental $proof$ for which $INCR.VF(TD_i, TD_j, proof))$ returns $True$.

- $True, Fale \leftarrow MEMBERSHIP.VF(TD_j, i, post_i^{'}, proof))$: Given $proof$, the function checks whether the *object* with tree digest TD_j has $post_i^{'}$ as the i^{th} post. This function can further accept multiple posts $MEMBERSHIP.VF(TD_j, \{i, \cdots, k\}, \{post_i^{'}, \cdots, post_k^{'}\}, proof))$ and checks their memberships with respect to TD_j. We write $post_i \in TD_j$ to indicate that there exists a membership $proof$ for which $MEMBERSHIP.VF(TD_j, i, post_i, proof))$ returns $True$.

7.5.3 Construction

Authorized users (with read and write access to the shared *object*) are associated with a signature key pair. $FList = \{(U_j, vk_{U_j})\}_{j \in [1,T]}$ shall contain the username U_j and the verification key vk_{U_j} of each user. T is the total number of authorized users. $FList$ is publicly available to the servers and the authorized users. Also, posts on the *object* are all encrypted using an encryption key ek. The corresponding decryption key is given to the authorized users. Note that for the sake of simplicity, we assume that the exchange of $FList$ and the encryption key pair among the users happens out of band, however, one may use the method proposed by [9] to further outsource the

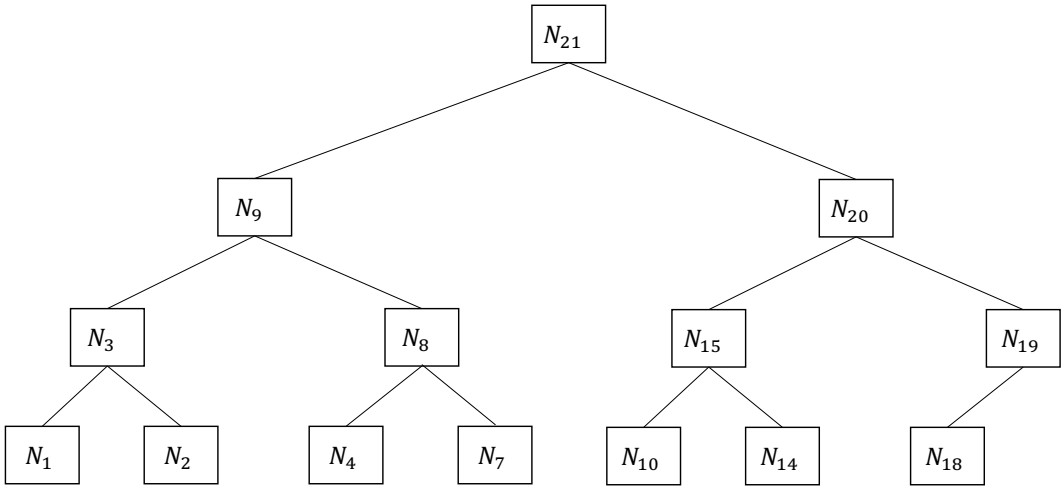

Figure 7.8: Version 7^{th} of the shared *object*. The full nodes as well as the nodes on the insertion path of the last post i.e., $N_{18}, N_{19}, N_{20}, N_{21}$ are shown.

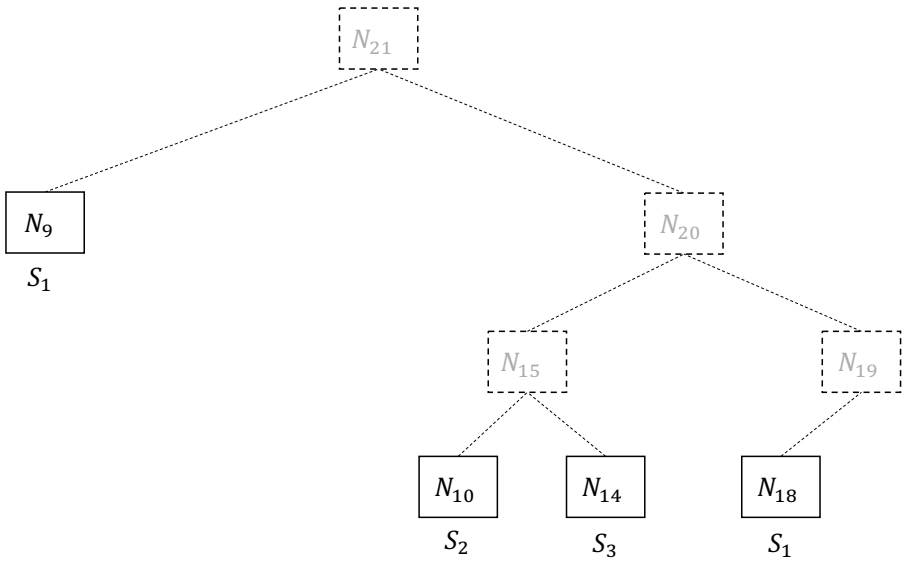

Figure 7.9: Nodes located on the membership proof of 5^{th} post concerning the 7^{th} version of the shared *object*. The solid rectangles represent the nodes included in the proof.

storage of $FList$ and management of users access to the servers side. Also, for the ease of explanation, we assume that the set of authorized users is static which can be extended to a dynamic version by deploying the proposal of [9].

Each server has a signature key pair and a unique index in the range of $[1, N]$. Servers publicize the ordered list $SList = \{S_i\}_{i=1:N}$ where each server S_i's index and verification key is accessible through $S_i.index$ and $S_i.vk$, respectively. For simplicity, we assume that the index of each server corresponds to its position in the list i.e., $S_i.index = i$. Besides, the definition of the hash function H to be used in the history tree is publicly available. Each server also sets up a database DB to store the parts of shared *object* and the history tree which is responsible for. Also, each server keeps track of the label of the last seen node as a tuple of (p, l), where p indicates the post number and l is the level of the node, in a local variable $Status$ which gets updated once a write operation takes place at that server.

Servers are accessible to the users through three different function calls $Write$, $Read$, and $GetStatus$. Users communicate with servers utilizing these function calls to handle their read and write requests. Namely, users interact with the servers through four protocols *Create object*, *Update Status*, *Read*, and *Write*. We consider an authenticated channel between users and servers. We elaborate on the function calls and protocols below.

Throughout our description, we distinguish between the data generated or operation performed by a server and a user using S and U subscript. That is, we write σ_{U_i} to indicate a signature generated by the i^{th} user. Likewise, $Sign_{U_i}(.)$, and $Verify_{U_i}(.)$ mean the execution of $Sign$ and $Verify$ algorithms using the signature key and verification key of i^{th} user, respectively. Following the same pattern, we will have σ_{S_i}, $Sign_{S_i}(.)$, and $Verify_{S_i}(.)$ for the i^{th} server.

Server-side function calls: Each server is available to the users through three function calls $Write$, $Read$, and $GetStatus$ that are explained below.

1. S_j**.Write**$(U_i, (p, l), in = (N_{p,l}, post, \sigma_{U_i}))$: User U_i calls this method to upload the tuple $in = (N_{p,l}, post, \sigma_{U_i})$ to be recorded for the node at the l^{th} level of the

insertion path of p^{th} post. $N_{p,l}$ refers to the hash value of the node in the history tree. *post* carries the content of a post and σ_{U_i} is a user-generated signature. Depending on the type of node i.e., tree digest, full or temporary, one or all of these fields might be empty. If the inserted node is a leaf node, then it is associated with the content of a post i.e., *post* and a user-side signature σ_{U_i} over $N_{p,l}||p$. Likewise, the tree digests should be associated with the user-side signature. However, for the temporary nodes, all the fields of *in* are empty.

The details of $Write$ procedure is shown in Algorithm 11. Firstly, the server needs to check whether the write operation is coming from an authorized user ($U_i \in FList$), the server is the corresponding storage provider of the intended node ($F(p,l) = S_j.index$), and if the node is the next node that the server is expected to receive (i.e., $F(p,l) - F(S_j.Status.p, S_j.Status.l)$=$N$); Note that servers hold $Status$ variable to keep track of their last seen post in the system. The last equality check is correct since the storage assignment of nodes to the servers is circular hence the labels of two consecutive nodes received by a server are N distant. If all the checks passed correctly, the server proceeds as below. If the node is a leaf node or a tree digest (line 2 and line 4), then the server must authenticate the user-side signature (lines 2-7). Upon successful verification, the server inserts the tuple *in* into the DB (line 8). Also, the server can remove all the user side signatures for the tree digests TD_i which refer to prior posts except the first post (line 9-11). This removal has a significant impact on the storage complexity where at any point in time there will be N signed tree digests saved in the system (rather than all of the signatures of the tree digests). Note that if the node is a temporary node then no data will be recorded for it (as *in* is empty)). The server updates its Status variable (line 12). If the inserted node is a tree digest, the server must sign the node (lines 13-15) and responds to the user accordingly. Otherwise, the server only acknowledges the success of the write operation (lines 16-18).

2. S_j.***Read***(p, l): Algorithm 12 demonstrates the read procedure. This function receives the index of the node in the history tree i.e., p as the post number and l as the level of the node in the insertion path of p^{th} post. The server checks whether it is responsible for the storage of the requested node i.e., $N_{p,l}$ (line 1). If not, it returns $\perp$ (line 2). Otherwise, the server retrieves the corresponding record from DB (lines 4-5). Note that as we discussed in the $S_j.Write$ algorithm, depending on the type of requested node some or all of the entries $(N_{p,l}, post, \sigma)$ might be empty. If the requested node is a tree digest, then the server generates a signature over $N_{p,l}$ (lines 6-10). Finally, the server sends the *record* and the signature (if any) to the user (line 11).

3. S_j.***GetStatus***(): The server sends its *Status* to the user (Algorithm 13).

User-side Protocols:

Create *object*: This protocol aims to initialize the share *object* by the insertion of its first post and then to communicate the first tree digest TD_1 with the authorized users. The content of the first post must be uniquely representative for the *object* e.g., the name of the group page together with its creation date. As such, given the first tree digest of the shared *object*, users can distinguish between different shared *object* s (e.g., different group pages). We assume one of the authorized users U_i e.g., the owner of a wall or the admin of a Facebook-like group page will run this protocol. The input of the user to this protocol is a *post*. The user contacts the server $S_{F(1,1)}$ who is responsible for the first post and calls its $Write$ function for $(U_i, (1,1), (H(post||1), post, Sign_{U_i}(H(post||1))$. As the result, user receives a signature $\sigma_{S_{F(1,1)}}$ over the inserted value $TD_1 = H(post||1)$ from the server. Note that, the hash value $H(post||1)$ corresponds to the $N_{1,1} = TD_1$.

The admin then communicates TD_1 with all the authorized users in $FList$. Each user initializes a local variable *Status* for the shared *object* which is the tuple of the following format $Status = (v, TD_v, \sigma)$ where v reflects the last version of shred *object* seen by the user, TD_v is the corresponding tree digest and σ is a server-side signature

of the TD_v. Each user sets the *Status* variable to $(1, TD_1, \sigma_{S_{F(1,1)}})$.

Update Status: This protocol, as demonstrated in Figure 7.10, is run between a user and the N servers through which the user aims to find the index of the latest post uploaded on the *object*, to fetch the corresponding tree digest, and check its consistency against her local Status variable. As such, the user collects the *Status* value of all the servers via their *GetStatus* function call (line 1). The largest *Status* value indicates the latest post index i.e., *last*. If the label of last nodes seen by the servers differ in more than N (N is the total number of servers), then inconsistency is detected (lines 2-3). This is because servers get to serve nodes in a circular manner, hence, the difference between the labels of the nodes seen by the servers can be at most N.

Next, the user must check whether her last seen tree digest TD_v (stored in her *Status* variable) (line 4) is consistent with the given tree digest TD_{last}. To do so, the user identifies the nodes holding the path of an incremental proof between tree digest TD_v and TD_{last} (line 5) and contacts the corresponding storage servers (lines 6-7). Next, she authenticates the retrieved signatures of leaf nodes and tree digests (lines 6-12). If the authentication succeeds, then the user checks whether the fetched proof is a valid incremental proof between TD_v and TD_{last} (line 13). If verified, the user updates her *Status* value to $(last, TD_{last}, \sigma_{S_{F(last,\lceil log(last)\rceil+1)}})$ (line 14). Note that $\sigma_{S_{F(last,\lceil log(last)\rceil+1)}}$ is fetched as a part of $proof$.

Read: During the *Read* protocol as shown in Figure 7.11, the user reads a certain range $R = [x, y]$ of posts i.e., $post_x, \cdots, post_y$ of the shared *object* (line 1). For this, the user first runs the *UpdateStatus* protocol and updates her *Status* variable (line 2). Next, she specifies the storage servers holding the nodes on the membership proof path of $post_{i \in R}$ (line 3). She contacts the servers and fetches the required data (lines 4-11). The fetched tree digest and leaf nodes should be appropriately signed by the issuing users (lines 6-9). Once the signatures are verified, the user verifies the correctness of membership proofs of the posts against TD_{last} (line 12). If the result of membership proof is false then a view inconsistency is detected (line 13).

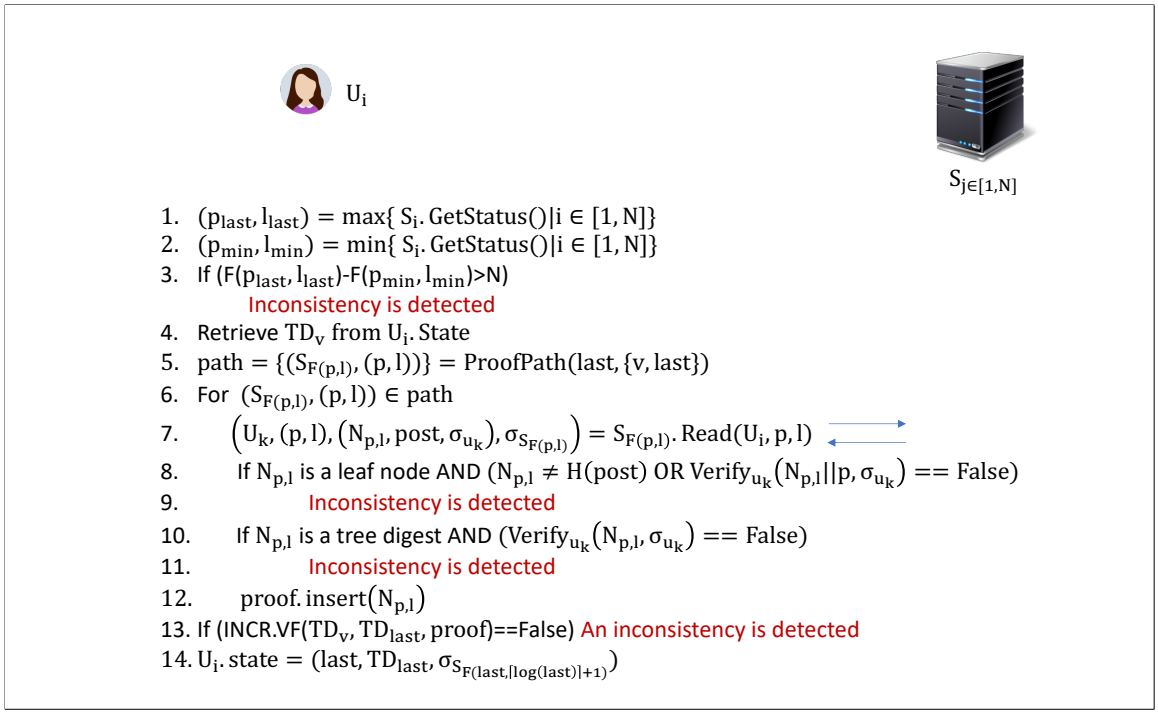

Figure 7.10: Update Status protocol. The arrows indicate users interaction with servers.

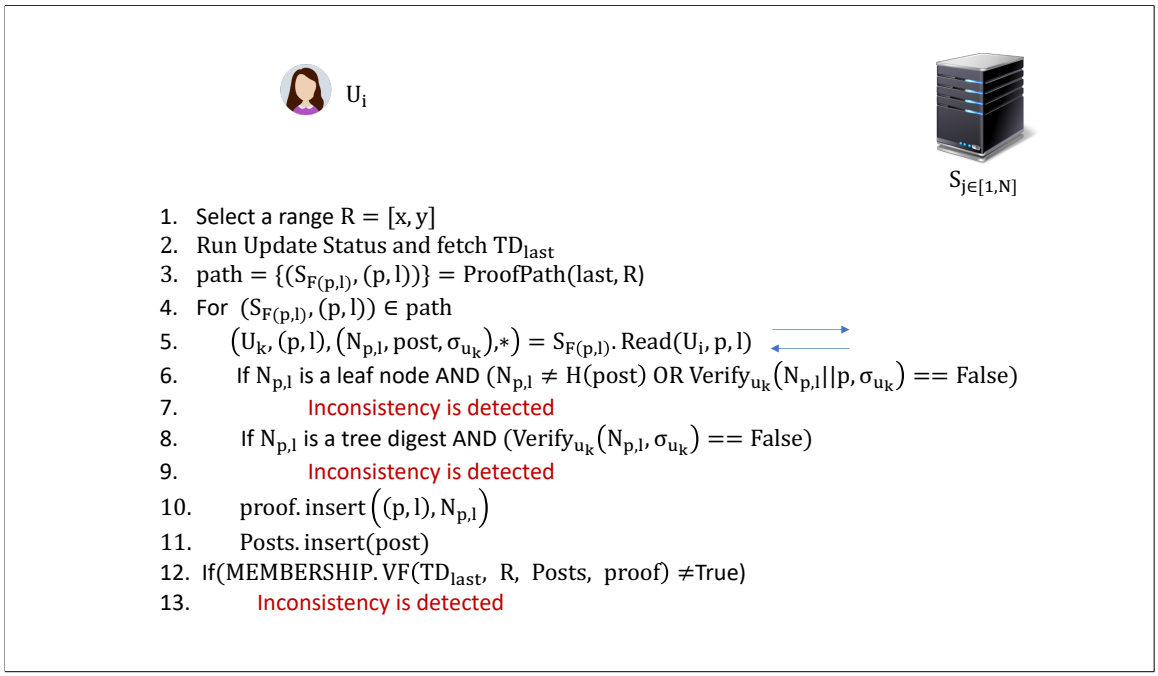

Figure 7.11: *Read* protocol. The arrows indicate the user's interaction with servers.

Write: In this protocol, illustrated in Figure 7.12, a user interacts with N servers to insert her post to the *object*. User initially runs *Update Status* protocol to fetch TD_{last} corresponding to the latest version of the *object* (line 1).

The user U_i crafts the content of her post (line 2) and signs it (line 3). She identifies the nodes on the insertion path of her post (line 5) and then fetches the values from the corresponding storage servers (lines 6-10). Next, she recomputes the hash values of intermediate nodes on the insertion path of her post (line 11). She submits the hash values to the corresponding servers by invoking their $Write$ function (lines 12-16). For the leaf node, the user submits the hash value $N_{last+1,1}$ together with the content of the post *post* and a signature σ (line 13). For the full node, the value of the node is sent to the corresponding server (line 14) whereas the storage server of the temporary node just gets informed about the insertion of the new post (line 15). If any of the servers respond by reject, then inconsistency is detected (line 17). The storage provider of the tree digest i.e., N_{p_c,l_C} returns a signature (line 18). The user authenticates the given signature and updates her Status accordingly (lines 16-17).

Audit: As we discussed before, each user is responsible to ensure that her post is correctly inserted to the *object* and is visible to all the other users. As such, every write operation of i^{th} post must be followed by a call to the $UpdateStatus$ at $i + q^{th}$ version of the *object*. That is, once the user uploads a post to the shared *object*, she must check the status of the shared *object* when q more posts are uploaded on top of her post. If the execution of $UpdateStatus$ at $i + q^{th}$ version of *object* concludes successfully, then the user ensures that her post is consistently visible to all the other users. Otherwise, an inconsistency is detected.

Audit Threshold: The value of q is a function of *object* version i (the index of inserted post) and the number of servers N as shown in Equation 7.8. We refer to q as *audit threshold* i.e., a threshold for the number of posts that a user needs to wait to be inserted on top of her post to ensure that her post is consistently visible to all the users.

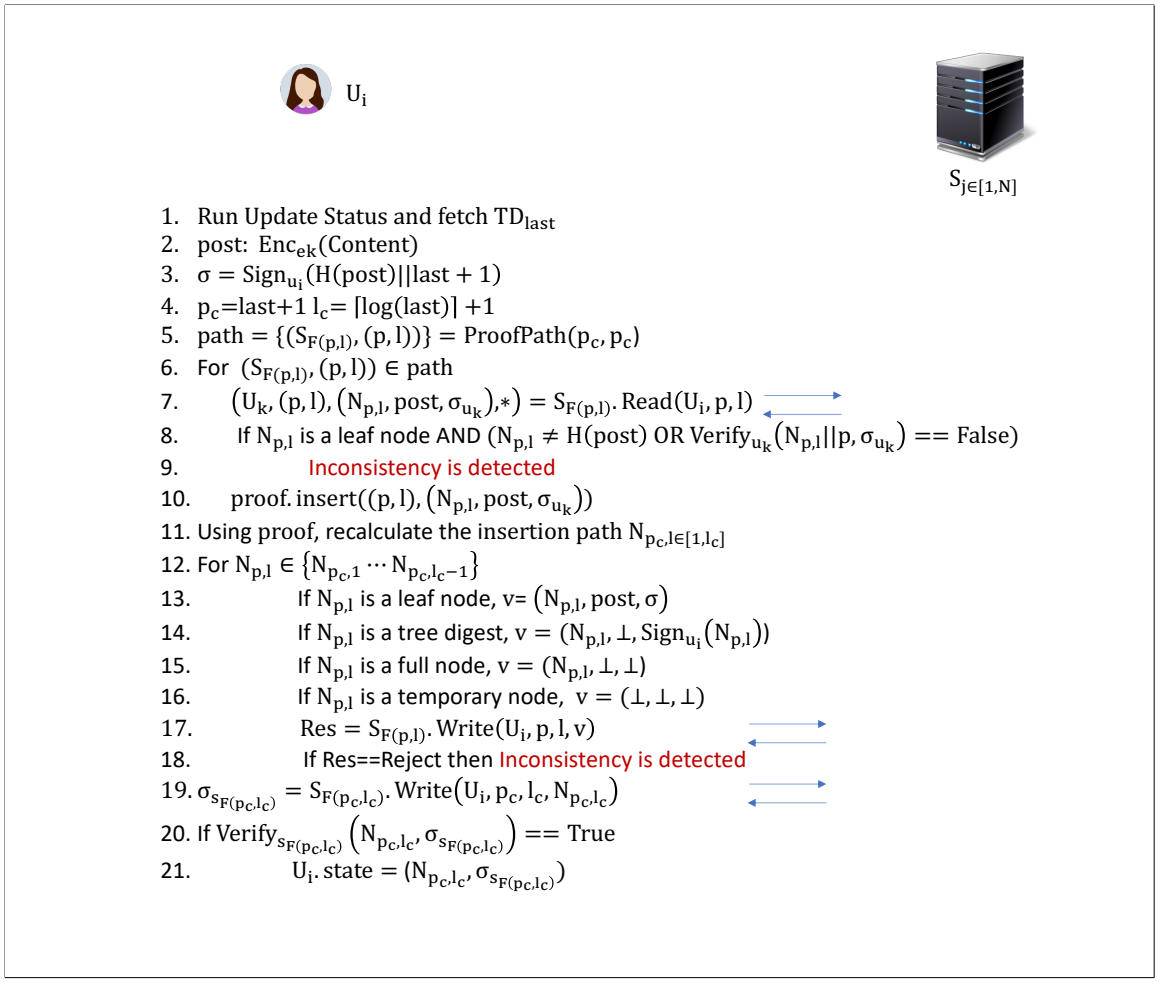

Figure 7.12: *Write* protocol. The arrows indicate the user's interaction with servers.

$$Q(i, N) = min(q) \text{ s.t. } \sum_{j=0:q} \lceil log(i+j) \rceil + 1 \geq N \tag{7.8}$$

As a concrete example, assume a system with 8 servers i.e., $N = 8$. A user who inserts the second post i.e., $i = 2$ shall execute *UpdateStatus* after the insertion of $Q(2, 8) = 2$ more posts on the *object* i.e., at the 4^{th} version of the *object*. If the *Update Status* protocol does not end successfully, then there is a view inconsistency e.g., servers attempted to drop her post or replace with another post.

We give some intuitions into why auditing the *object* after $Q(i, N)$ posts will result in achieving q-detectable consistency. In section 8.2, we will provide a formal security definition for a q-detectable consistent system together with solid proof as to how Integrita satisfies q-detectable consistency relying on our proposed auditing strategy.

For the insertion of each post i, servers that are located on the insertion path will be informed about the insertion of that new post regardless of the type of nodes they are responsible for (see Figure 7.12). Recall that we assume at least one of the servers is honest. We use the term of frozen post for a post whose part of the insertion path gets to be served by the honest server by frozen post. It is named frozen since due to the presence of the honest server, no other post with the same index as the frozen post will exist; the honest server will not accept the insertion of two posts with the same index (as indicated in line 2 of Algorithm 11). This implies that for the frozen post with the index of f, there would be only one tree digest TD_f in the system which represents a unique history (sequence of posts) of the *object*. We call a tree digest corresponding to a frozen post as frozen tree digest. All the other tree digest TD_j created as the result of further write operations $j > f$ will comply with the history that TD_f represents (this is due to the incremental proof check in the *Step 11 of the Update Status* protocol). Thus, if a post i where $i < f$ belongs to the sequence of posts that a frozen tree digest TD_f represents then it will certainly belong to all the future versions of the *object* (again due to the incremental proof check in the *Step 11 of the Update Status* protocol). Thus, to ensure view consistency, the user needs to perform a consistency check between the tree digest at the time of insertion of her

post and the very next frozen tree digest. To determine the index of the next frozen tree digest, we need to know the index of the honest server. However, there is no presumption about which server will act honestly. As such, after insertion of each post i the user shall wait for q many posts to be inserted as the result of which all the servers get contacted at least once. As the storage of nodes is assigned to the servers under a circular pattern, if the sum of the length of the insertion path of the next q posts exceeds N, it means that all the N servers, including the honest server whose index is unknown, are contacted at least once. Equation 7.8 calculates q i.e., the total number of posts (inserted after i^{th} post) whose insertion paths lengths on aggregate exceeds N. Recall that the number of nodes located on the insertion path of j^{th} post is $\lceil log(j) \rceil + 1$ which means $\lceil log(j) \rceil + 1$ distinct servers get contacted as the result of insertion of the j^{th} post.

Analysis of Audit Threshold Figure 7.13 shows the audit threshold computed based on function $Q(i, N)$ under different number of servers N and post number i. The audit threshold for a particular post number will increase with the number of servers e.g., the threshold audit for post number 65 for $N = 8, 16, 24$, and 32 are $0, 1, 2$ and 3 respectively.

After a certain version of *object*, every inserted post is a frozen one since all the servers get contacted as the insertion of each post. Indeed, the *object* enters into its strong consistent version where no fork can happen in users' views. We call that version of the *object* as *transition point*. For a given N, its transition point is computed as given in Equation 7.9.

$$TP(N) = 2^{N-2} + 1 \tag{7.9}$$

For example, with $N = 8$, strong consistency starts at version 65 whereas with $N = 16$ the transition point is 16385. Thus the higher the number of servers the later the *object* enters its strong consistency version. The transition points of the different numbers of servers (1-20) are illustrated in Figure 7.14.

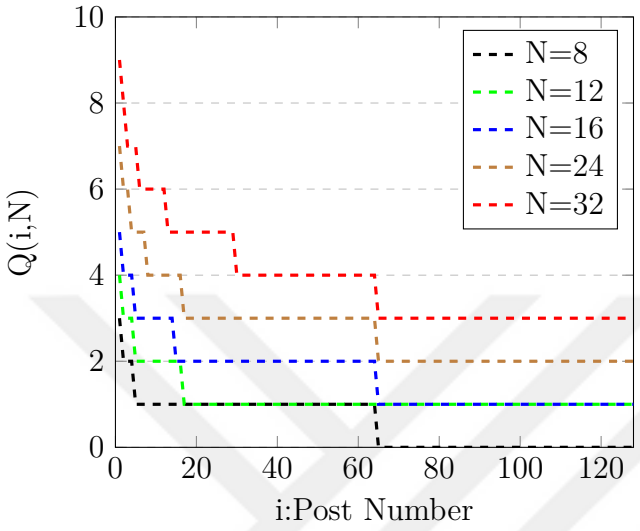

Figure 7.13: The Audit Threshold for various number of servers each demonstrated by a different diagram. The x axis represents the index of post whereas the y axis shows the audit threshold computed based on function $Q(i, N)$ given in Equation 7.8.

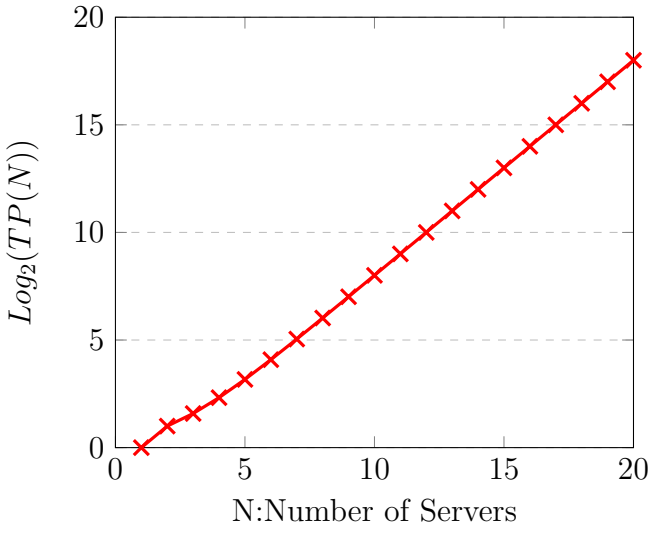

Figure 7.14: The Transition point for various number of servers as defined in Equation 7.9. The x axis represents the number of servers whereas the y axis shows the the logarithm of the transition point.

Algorithm 11 S_j.Write(U_i, (p,l) , $in = (N_{p,l}, post, \sigma_{U_i})$)

1: **if** $U_i \in FList$ AND $F(p,l) = S_j.index$ AND $F(p,l) - F(S_j.Status.p, S_j.Status.l)=N$ **then**
2: **if** $N_{p,l}$ is a leaf node AND ($H(post) \neq N_{p,l}$ OR $Verify_{U_i}(N_{p,l}||p, \sigma_{U_i}) \neq$ accept) **then**
3: Return Not Verified
4: **end if**
5: **if** $N_{p,l}$ is a tree digest AND $Verify_{U_i}(N_{p,l}, \sigma_{U_i}) \neq$ accept **then**
6: Return Not Verified
7: **end if**
8: Insert $(U_i, (p,l), in)$ into DB
9: **if** $S_j.Status.p \neq p$ **then**
10: Remove the user signature σ_{U_i} $\forall$ $TD_i \in S_j.DB \setminus TD_1$
11: **end if**
12: $S_j.Status$=(p,l);
13: **if** $N_{p,l}$ is a tree digest **then**
14: Return $Sign_{s_j}(N_{p,l})$
15: **else**
16: Return *accept*
17: **end if**
18: Return **Res**
19: **end if**

Algorithm 12 S_j.Read(U_i, p, l)

Output: $(record = (U_i, (p, l), (N_{p,l}, post, \sigma_{U_k}), \sigma_{S_j})$

1: **if** $U_i \notin FList$ OR $F(p, l) \neq S_j.index$ **then**
2: Return $\perp$
3: **end if**
4: $record = DB.get(p, l)$
5: Parse $record$ as $(*, (p, l), N_{p,l}, *, *)$
6: **if** $N_{p,l}$ is tree digest **then**
7: $\sigma_{S_j} = Sign_{S_j}(N_{p,l})$
8: **else**
9: $\sigma_{S_j} = \perp$
10: **end if**
11: Return $(record, \sigma_{S_j})$

Algorithm 13 S_j.GetStatus ()

1: Return $S_j.Status$

Notes to Chapter 7

1 https://www.brandwatch.com/blog/facebook-statistics/

2 https://www.statista.com/statistics/264810/number-of-monthly-active-Facebook-users-worldwide/

3 https://www.brandwatch.com/blog/facebook-statistics/

Chapter 8

INTEGRITA: COMPLEXITY, PERFORMANCE AND SECURITY

8.1 Complexity and Performance

In this section, we analyze the asymptotic performance of Integrita with respect to the storage overhead (section 8.1.1), communication round and communication complexity (section 8.1.2) for both servers and users.

8.1.1 Storage Overhead

User: Users have to store constant amount of data as for their Status variable.

Server: Servers are responsible to store the object's posts and its associated history tree i.e., the full nodes, the tree digests, and the leaves.

$$hSize \cdot |L| + hSize \cdot |F| + hSize \cdot |T - F| + SSize \cdot |L| + SSize \cdot |T| \tag{8.1}$$

An object with M posts consists of M leaves, and M full nodes and M tree digests. However, out of M tree digests, some of them overlap with the full nodes hence are already saved in the system. Indeed, out of M tree digests (for M posts), $log(M)$ many of them associated with post indices 2^0, 2^1, 2^2,..., $2^{log(M)}$) are full nodes, hence, servers the number of tree digests to be stored by the servers will be $M - log(M)$ (rather than M). Additionally, every leaf node is attached to a user-side signature, thus, M signatures shall be maintained by the servers. Also, for an object with M posts, $O(N)$ many signed tree digests are stored by the servers which result in $O(N)$ signatures.

As such, the total amount of storage spent by the N servers is given in Equation

Design	Asymptotic Overhead	Concrete total overhead	Concrete overhead per server	Consistency level
Centralized	$hSize \cdot (2 \cdot M) + SSize \cdot M$	124TB	124TB	Fork-consistency
Replication-based	$N \cdot [hSize \cdot (2 \cdot M) + SSize \cdot M]$	2488TB	2488TB	Strong consistency
Integrita	$hSize \cdot (3M - Log(M)) + SSize \cdot (M + N)$	143TB	7.1TB	q-detectable consistency

Table 8.1: Storage overhead of Integrita vs related work. N: number of servers. M: number of posts on the object. $SSize$: The size of each signature in bit length. $hSize$: the bit-length of hash output.

8.2 which is of $O(M)$.

$$hSize \cdot (\underbrace{M + M}_{\text{leaves and full nodes}} + \overbrace{M - log(M)}^{\text{tree digests that are not full nodes}}) + SSize \cdot (M + N) \quad (8.2)$$

Related Work: In a replication-based solution, one needs to copy the history tree (of $2M$ nodes) as well as the signed leaves of the history tree (i.e., M signatures) over N servers. Thus, the storage overhead for such design would be $N \cdot [hSize \cdot (2 \cdot M) + SSize \cdot M]$. In the centralized systems, the server stores the history tree (With $2 \cdot M$ nodes) together with its user-side authenticated leaves (i.e., M signatures) which results in $hSize \cdot (2 \cdot M) + SSize \cdot M$ overhead.

Table 8.1 summarizes the comparison of the storage overhead of Integrita with the centralized and the replication-based solutions. The concrete overhead is measured for annual storage consumption of a social network like Facebook, for walls of its 2.41 billion monthly active users[1] each wall containing 1241 posts (per year) [2]. We deploy SHA-3 as the hash function with a 512-bit output length and RSA signature scheme with a 2048-bit signature length. The number of servers N is set to 20.

8.1.2 Round Complexity and Communication Complexity

In this section, we analyze the round complexity as well as the communication complexity i.e., the number of bits communicated between parties during the protocol. We consider each communication round to be a sent and a receive operation. If multiple rounds can be done concurrently (they are independent), then we count them as one round. The results are summarized in Table 8.2.

- *Update Status:* In the first round, the user contacts all the N servers to get their latest Status. Then she performs a consistency proof check between her local state variable and the latest state of the object. To fetch values of the incremental proof path, the user contacts with N servers (at most) and downloads the necessary values concurrently. Thus, the overall round complexity of *Update Status* is 2. Likewise, as the result of *Update Status*, each server may get contacted twice, once to share its latest status and the second time when the server may be located on the proof path.

  The communication complexity of *Update Status* is to download the signed state of servers S_i for $i \in [1, N]$ as well as fetching the incremental proof. The former requires $SSzie \cdot N$ data transfer whereas the latter involves the transmission of at most $2 \cdot log(M)$ hash values (M is the number of object's posts). The user additionally downloads the user authenticated version of the last post's tree digest and leaf node as well as the server-signed version of the tree digest which adds 3 more $SSize$ to the amount of transferred data. As such, the communication complexity at the user side is at most $SSize \cdot (N + 3) + hSize \cdot 2.log(M)$. The average communication complexity for each server is $\frac{SSize \cdot (N+3) + hSize \cdot 2.log(M)}{N} \approx SSize + \frac{hSize \cdot (2 \cdot log(M))}{N}$

- *Read:* Let the object contains M posts, and a user wants to read a consecutive range $R = [i, j]$ of posts. She has to fetch the membership proof paths of all the posts from the corresponding servers. The user can connect to all the servers simultaneously, hence she can perform read in 1 round of communication. As a result, each server at most gets contacted also once which results in communication complexity of 1 for each server.

  The user downloads R many leaf nodes with their user-side signatures which requires $R \cdot (hSize + SSize)$ data transmission. The proof path includes at most $2 \cdot log(M)$ hash values. On aggregate, user communicates $hSize \cdot (2.log(M) + R) + SSize \cdot R$ bit data. Consequently, the average data transfer on each server

Entity\Overhead	Update Status	Read	Write
User	2	1	2
Servers	2	1	2

(a) Integrita Communication Complexity

Entity\Overhead	Update Status	Read	Write
User	$SSize \cdot (N+3) + hSize \cdot 2 \cdot log(M)$	$hSize \cdot (2.log(M) + R) + SSize \cdot R$	$hSize \cdot 2 \cdot log(M)$
Servers	$SSize + hSzie \cdot \frac{2 \cdot log(M)}{N}$	$\frac{hSize \cdot (2.log(M)+R)+SSize \cdot R}{N}$	$\frac{hSize \cdot 2 \cdot log(M)}{N}$

(b) Integrita Communication Complexity.

Table 8.2: Communication complexity. N: number of servers. M: number of posts on the object. $SSize$: The size of each signature in bit length. $hSize$: the bit-length of hash output. R: number of consequitive operations to be read from the object.

is $\frac{hSize \cdot (2.log(M)+R)+SSize \cdot R)}{N}$.

- *Write:* To insert a post to the object, the user needs to fetch the nodes on the insertion path of her post. This can be handled in a 1 round of communication with concurrent connections to the servers. Next, the user recomputes the values for the nodes on the insertion path of her post and upload the new values to the servers. This also counts as a 1 round of communication. Thus, in total user performs the write operation in 2 rounds of communications. Subsequently, servers may get contacted for at most 2 rounds.

  From the communication complexity perspective, downloading the insertion path of the current post requires to download at most $2 \cdot log(M)$ hash values. As such, the user-side communication complexity would lead to $hSize \cdot 2 \cdot log(M)$ bits. The data transfer at the server-side shall be $\frac{hSize \cdot 2 \cdot log(M)}{N}$.

8.2 Security

8.2.1 q-Detectable Consistency and Inconsistency Interval

In a q-detectable consistent system, views of users toward the i^{th} version of a shared *object* is guaranteed to be consistent expect for the last δ posts i.e., $post_{i-\delta}, ..., post_i$.

We use the term *inconsistency interval* to refer to the range of the posts i.e., $[i-\delta, i]$ where the inconsistency is allowed. The views of users for any history of *object* preceding $i-\delta$ version of the *object* is guaranteed to be the same. In Integrita, δ is a function of *object* version i and the number of servers N and its value is computed using function $\Delta(i, N)$ given in Equation 8.3.

$$\Delta(i, N) = max(q \in [0, i]) \text{ s.t. } \sum_{j=0:q} \lceil log(i-j) \rceil + 1 \geq N \tag{8.3}$$

As a concrete example, assume a system with 8 servers i.e., $N = 8$ and two users looking at the 5^{th} version of the
object, the inconsistency interval is 2 ($\Delta(5, 8) = 2$) that is the view consistency holds for all the posts except the 4^{th} and the 5^{th} post. As such, the following two views $View_5 = \{post_1, post_2, post_3, post_4, post_5\}$ and $View'_5 = \{post_1, post_2, post_3, post'_4, post'_5\}$ are q-consistent since the consistency holds for all the posts out of the inconsistency interval. However, the following two views $View_5 = \{post_1, post'_2, post_3, post_4, post_5\}$ and $View'_5 = \{post_1, post'_2, post_3, post'_4, post'_5\}$ do not satisfy q-consistency because there is an inconsistency at the second post which is out of the inconsistency interval.

To capture the notion of q-consistency, we define the following game to be played between an adversary $\mathcal{A}$ and a challenger $Chal$. The adversary shall control $N-1$ servers whereas the $Chal$ gets to play for authorized users U_i $i \in FList$ and the honest server. We write F to indicate the indices of corrupted servers and S_h to be the honest server. The adversary can dictate the read and write operations to be done by particular users. However, it does not have control over the Audit protocol execution. The challenge for the adversary is to make two users U and U' accept two q-inconsistent views of the i^{th} version of the *object* i.e., there is at least one index $j \notin [i - \Delta(i, N), i]$ for which U and U' read $post_j$ and $post'_j$ as the j^{th} post such that $post_j \neq post'_j$.

q-Detectable Consistency Experiment q-Det-Consistency(1^λ)

1. The challenger gives the security parameter 1^λ to the adversary. The adversary communicates a set of signature verification keys $\{vk_{S_i}\}_{i \in F}$ for the servers under its control to the challenger. The challenger runs the signature key generation algorithm for the honest server and hands the vk_{S_h} to the adversary. Also, the challenger generates the signature key pairs for the users $U_1, ..., U_T$ and outputs $FList = \{(U_1, vk_{U_1}), \cdots, (U_T, vk_{U_T})\}$ to the adversary.

2. The adversary specifies a user U_j to create the shared *object* $\mathcal{D}$ through the invocation of *Create object* protocol.

   Steps 3 and 4 can be repeated polynomial times by the adversary.

3. The adversary specifies a user U_j to $Write$ a *post* on the *object* $\mathcal{D}$. $Chal$ runs the $Write$ protocol accordingly. Note that after each write operation, the challenger shall act upon the Audit protocol.

4. The adversary specifies a range $R = [l, r]$ to be read by a particular user U_i. $Chal$ runs the *Read* protocol accordingly.

5. The adversary specifies two users U and U', a version number j and a post index i^* where $i^* < j - \Delta(j, N)$. The challenger runs *Read* protocol for j^{th} post on behalf of U and U'. $\mathcal{A}$ wins if *Read* protocol ends successfully for U and U' such that the tree digest in the Status variable of U and U' both have index j and $post_{i^*}$ and $post'_{i^*}$ be the posts the U and U' read, respectively s.t. $post_{i^*} \neq post'_{i^*}$.

Definition 5 *A storage system provides q-detectable consistency for a shared object if the success probability of adversary in q-Det-Consistency(1^λ) experiment is negligible in the security parameter λ.*

Theorem 7 *If the deployed signature scheme is existentially unforgeable under adaptive chosen message attack and the hash function is secure then Integrita satisfies q-detectable consistency.*

Proof Overview: If A wins i.e., U and U' read two different post $post_{i^*} \neq post'_{i^*}$ where $i^* < j - \Delta(j, N)$ this implies that there is a fork in the system where users are split into two groups depending on whether they are shown $post_{i^*}$ or $post'_{i^*}$. For both forks to successfully continue till the j^{th} version of the object, each fork should have a successful chain of write operations from i^{*th} to j^{th} version of the object. However, for the frozen $post_k$ where $i^* < k < i^* + Q(i^*, N) < j$, the honest server accepts only one write operation which results in one valid tree digest hence only one fork will get to grow. For the other fork (namely the second fork) to grow, the corrupted servers need to bypass the honest server. As such, the corrupted servers need to convince the users of the second fork that the last post on the object has an index higher than k so that they won't attempt insertion of the k^{th} post. However, this would only happen if the corrupted servers can generate an authenticated post and tree digest on behalf of an authorized user from the second fork. Thus, if B can guess for which authorized user this forgery takes place, B will exploit this forgery and breaks the unforgeability of the underlying signature scheme.

Proof: We base our proof over the following lemma that is due to [148].

Lemma 1 *If there is a valid incremental proof between two tree digests TD_i and TD_j, then for every operation $post_k$ where $k < i$ for which there is a valid membership proof, s.t. $True \leftarrow MEMBERSHIP.VF(k, TD_i, post_k, proof)$), and $post'_k$ s.t. there is a $proof'$ for which $True \leftarrow MEMBERSHIP.VF(k, TD_j, post'_k, proof)$) then $post_k$ must be equal to $post'_k$. Namely, if two tree digests are consistent then they both represent the same sequence of operations for their shared past [148].*

Proof: If there exists an adversary $\mathcal{A}$ who wins q-Det-Consistency(1^λ) with non-negligible probability ϵ then we construct a simulator B who breaks the underlying signature scheme. The internal code of B is given below. B is given the security

parameter 1^λ as well as a signature verification key vk' from the signature scheme challenger.

1. The challenger gives the security parameter 1^λ to the adversary. The adversary communicates a set of signature verification keys for the corrupted servers $\{vk_{S_i}\}_{i\in F}$ to the challenger. The challenger runs the signature key generation algorithm for the honest server and hands the vk_{S_h} to the adversary. B selects a random value $\beta \leftarrow [1, T]$. B sets the signature verification key of U_β to vk' and for the rest of users generates the signature key pairs as normal. B sends $FList = \{(U_1, vk_1), \cdots, (U_\beta, vk'), \cdots, (U_T, vk_T)\}$ to the adversary.

2. The adversary specifies a user U_j to create the shared *object* $\mathcal{D}$ through the invocation of *Create object* protocol. If $j = \beta$, then to generate required signatures, B queries the signing oracle of the outside challenger and stores the set of queried messages and signatures in set $QSign$. Otherwise, B acts as in *Create object* protocol.

3. The adversary specifies a user U_j to write a post on the *object* $\mathcal{D}$. B runs the $Write$ protocol accordingly.

   First, B runs the Update Status and fetch the latest tree digest TD_{last}. As the result of running Update Status, B obtains $proof = \{(N_{p,l}, post, \sigma_{U_k})\}$ for some p and l. If there exists a tree digest $N_{x,y} \in proof$ (or a leaf node) signed by U_β as σ_{U_β} s.t. $N_{x,y} \notin QSign$ (or $N_{x,y}||x \notin QSign$) then B outputs ($N_{x,y}$, σ_{U_β}) (or ($N_{x,y}||x$,σ_{U_β})) to the outside challenger.

   B fetches required nodes for the insertion of the new post as $proof = \{(N_{p,l}, post, \sigma_{U_k})\}$. If there exists a tree digest $N_{x,y} \in proof$ (or a leaf node) signed by U_β as σ_{U_β} s.t. $N_{x,y} \notin QSign$ (or $N_{x,y}||x \notin QSign$) then B outputs ($N_{x,y}$, σ_{U_β}) (or ($N_{x,y}||x$,σ_{U_β})) to the outside challenger.

   B recalculates the nodes on the insertion path of her post as well as the tree digest. B signs the leaf node i.e., $H(post)||last + 1$ and the tree digest TD_{last+1}

using the U_j signature key. If $U_j == U_\beta$ then B queries the signing oracle of the outside challenger and inserts the queried message and the obtained signature to $QSign$.

If an inconsistency is detected as the result of $Write$ protocol, B immediately aborts.

Note that after each write operation, B shall act upon the audit protocol i.e., B runs the Update Status protocol at the $last + Q(last, N)$ version of the object. As the result of running Update Status B obtains $proof = \{(N_{p,l}, post, \sigma_{U_k})\}$. If there exists a tree digest $N_{x,y} \in proof$ (or a leaf node) signed by U_β as σ_{U_β} s.t. $N_{x,y} \notin QSign$ (or $N_{x,y}||x \notin QSign$) then B outputs ($N_{x,y}$, σ_{U_β}) (or ($N_{x,y}||x$,σ_{U_β})) to the outside challenger. If any inconsistency is detected as the result of Update Status, then B aborts.

4. The adversary specifies a range $R = [l, r]$ to be read by a particular user U_i. B runs the *Read* protocol accordingly. If an inconsistency is detected as the result of *Read* protocol, then B aborts. Otherwise, during the execution of *Read* protocol, B obtains $proof = \{(N_{p,l}, post, \sigma_{U_k})\}$. If there exists a tree digest $N_{x,y} \in proof$ (or a leaf node) signed by U_β as σ_{U_β} s.t. $N_{x,y} \notin QSign$ (or $N_{x,y}||x \notin QSign$) then B outputs ($N_{x,y}$, σ_{U_β}) (or ($N_{x,y}||x$,σ_{U_β})) to the outside challenger.

5. The adversary specifies two users U and U', a version number j and a post index i^* where $i^* < j - \Delta(j, N)$. B runs *Read* protocol for U and U' separately. B acts identically to the step 4 to run the *Read* protocol. Let TD_j and TD'_j indicate the Status variable of U and U' after the *Read* protocol execution . Also let $post_{i^*}$ and $post'_{i^*}$ indicate the read posts for U and U'. If $post_{i^*} \neq post'_{i^*}$ and *Read* protocol ends without detecting any inconsistency for both U and U' then B will find a signature forgery as we discuss below.

Note that the inconsistency between $post_{i^*}$ and $post'_{i^*}$ means that there will be

two different tree digests TD_{i^*} (with $post_{i^*}$ as its i^{*th} post) and TD'_{i^*} (with $post'_{i^*}$ as its i^{*th} post). As such, from version i^* onward, users will be divided into two groups G and G' depending on whether they are shown $post_{i^*}$ (TD_{i^*}) or $post'_{i^*}$ (TD'_{i^*}). More precisely, a group G of users whose further Status variables i.e., TD_f where $f \geq i^*$ are consistent with TD_{i^*} i.e., $TD_{i^*} \rightarrow TD_f$ and the other group G' whose further Status variables i.e., TD'_f where $f \geq i^*$ are consistent with TD'_{i^*} i.e., $TD'_{i^*} \rightarrow TD'_f$.

Recall that every read and write operation requires the user to run the Update Status protocol, and to perform an incremental proof check between local Status variable and the latest state of the system i.e., TD_{last}. Since users are divided in two groups G and G', there will be two separate chains of posts (after i^{*th} post) generated by group G and G' i.e., $post_i$ $i \in [i^*, j]$ uploaded by group G and $post'_i$ $i \in [i^*, j]$ performed by users of group G'. Let assume $k \in [i^*, i^* + Q(i, N)]$ be the index of the next frozen node (the honest server is the storage server of one of the nodes on the insertion path of post k). Assume that a user from a group G attempts the insertion of $post_k$ earlier than a user from a group G'. Since the honest server appears on the insertion path of $post_k$, it gets informed about the inclusion of k^{th} post and update its *Status* accordingly. When a user from the group G' holding a state variable TD'_i wants to insert $post'_k$, it first runs the Update Status to fetch the latest version of the *object* and perform consistency check between TD'_i and the current version of the object. During the status update protocol, the adversary may try to act dishonestly which we discuss next.

1. The adversary may attempt to send an incorrect Status value to the user and make her accept a lower version $< k$ of the
   object. However, due to the presence of the honest server (who has witnessed the insertion of $post_k$), the adversary does not succeed as the honest server will communicate its intact state value i.e., k with the user.

2. The adversary may attempt sending a Status value x where $x \geq k$ for which the adversary also needs to come up with a valid tree digest TD'_x where $TD'_x \implies$

TD'_i (TD'_i is the status of user while inserting $post'_k$) in order to pass the Update Status protocol successfully. To come up with a valid TD'_x, the adversary has the following choices:

(a) The adversary may use the tree digest TD_x that is signed and generated by one of the members of the group G. However, any tree digest TD_x generated by a member of group G will be consistent with TD_{i^*} but not with TD'_{i^*} i.e., $TD_{i^*} \not\Rightarrow TD_x$. This means that there will be no valid incremental proof between TD'_{i^*} and TD_x. Thus this choice is absolute.

(b) The other choice for the adversary is to generate a $post'_x$ and forge a signature on $H(post'_x||x)$ (the leaf node) on behalf of an authorized user.

(c) The adversary uses $post_x$ generated by one of the members of G and computes the tree digest TD'_x accordingly. $\mathcal{A}$ also needs to generate a valid signature over TD'_x from one of the authorized users.

This means that for a member of group G' to accept that the latest version of *object* is $x \geq k$ and successfully pass the Update Status protocol, the adversary needs to forge a signature on behalf of an authorized user U'' either on the leaf node $H(post'_x||x)$ or the tree digest TD'_x. Thus, B shall figure out this forgery while fetching the incremental proof on behalf of a member of the group G'.

B can win the signature game if the forgery of the adversary is from U_β. Recall that the probability of $\mathcal{A}$ winning the q-Det-Consistency(1^λ) is $\epsilon(\lambda)$ and the total number of users i.e., T is a polynomial $poly(\lambda)$. Thus, we have

$$\begin{aligned} Pr[B \text{ breaks the signature}] &= Pr[\text{q-Det-Consistency}(1^\lambda) \ = 1 \text{ AND } U'' = U_\beta] \\ &= Pr[\text{q-Det-Consistency}(1^\lambda) \ = 1 | U'' = U_\beta] \cdot Pr[U'' = U_\beta] \\ &= \epsilon(\lambda) \cdot \frac{1}{T} \\ &= \epsilon(\lambda) \cdot \frac{1}{poly(\lambda)} \end{aligned} \tag{8.4}$$

if $\epsilon(\lambda)$ is non-negligible, then B also breaks the signature scheme with non-negligible probability. This concludes the proof. ■

Notes to Chapter 8

1 https://www.businessinsider.com/facebook-grew-monthly-average-users-in-q1-2019-4

2 https://blog.wishpond.com/post/115675435109/40-up-to-date-facebook-facts-and-stats

Chapter 9

CONCLUSION

9.1 Remarks

In this thesis, a framework of privacy-preserving services for distributed OSNs operated on the federated server architecture is proposed. The thesis is comprised of three modules, namely Privado, Anonyma, and Integrita.

In the first module of the framework, Privado, we address the lack of advertising service in the secure OSNs. Privado is a privacy-preserving group-based advertising system by which servers can match advertising requests to the encrypted profiles of users. We introduce and utilize the group-based advertising notion to enable user privacy, i.e., to hide the identity of the exact target customers. As such, users are divided into groups of size k at the registration time and then submit their profiles in an encrypted format. Advertisers submit their requests as plaintext. Servers find the target groups for the advertising requests. User privacy (i.e., the unlinkability of group matching result to the individual group members) is preserved against a malicious adversary who corrupts $N-1$ out of N servers, $k-2$ out of k members of each group and any number of advertisers. We formally define and prove user privacy relying on the CPA security of the deployed encryption scheme. Our advertising scheme enjoys advertising transparency where the entire process of matching groups to the advertising requests are done independent of users' and advertisers' collaboration. We perform experimental simulations and measure the advertising running time over a various number of servers and group sizes. We additionally discuss the optimum number of servers, concerning user privacy and advertising running time. We also present advertisement accuracy metrics under various system parameters providing a range of security-accuracy trade-offs.

In the second module of the framework, Anonyma, we propose an anonymous invitation-only system satisfying *inviter anonymity* and *invitation unforgeability* simultaneously. The inviter anonymity guarantees that the knowledge of who is invited by whom remains confidential against the system administrator as well as the inviters of the same invitee. By the invitation unforgeability, the system administrator is guaranteed that invitees without a sufficient number of inviters would not be able to successfully authenticate themselves to the system. Both security objectives are formally defined and proved in a *malicious adversarial model*. The anonymity of inviter relies on the security of the employed pseudo-random generator and the invitation unforgeability relies on the hardness of computational Diffie-Hellman assumption. Anonyma is efficiently scalable in terms of the number of inviters. That is, the administrator can issue credentials to the new members (without re-keying other existing members) to be immediately able to invite others. Unlike the prior studies whose running time depends on the total number of system members, in Anonyma, the running time complexity of invitation generation is $o(t)$ (t is the required number of inviters) and the verification of invitation requires constant many operations at the administrator. Additionally, we devise an anonymous cross-network invitation-based system, AnonymaX which is a slightly modified variant of Anonyma. AnonymaX empowers users of one social network to act as inviters for another network. AnonymaX achieves provable inviter anonymity the same way as Anonyma does. The proof of invitation unforgeability of AnonymaX is provided basing on the hardness of computational Diffie-Hellman assumption.

In the third module of the framework, Integrita, we address the view consistency issue in a collaborative data-sharing environment like Facebook group pages. The shared data called a shared object is comprised of a sequence of posts which can be generated by any of the authorized users. The view consistency concerns that all the authorized users are shown the same set of posts and with the intact order. In Integrita, we introduce a new level of consistency called *q-detectable consistency* where any inconsistency between users view cannot remain undetected for more than

q posts. Integrita preserves q-detectable consistency as long as one server does not collude with the rest of the servers. In Integrita, q is a function of the number of posts uploaded on the shared object as well as the number of servers. Integrita outperforms the state of the art in two major directions. First, unlike the replication-based solutions, Integrita operates only on one instance of the shared object that is maintained collaboratively by all the servers. As such, Integrita saves 2344 Terabyte of storage annually for an OSN with 2.3 billion users and running on the federation of 20 servers (i.e., service providers). We enable this by trading the strong consistency with the q-detectable consistency. Second, in contrast to the centralized counterparts in which the inconsistency detection relies on the users' direct communication, Integrita detects any fork in the users' views regardless of users direct communication. Nevertheless, distributing the storage of shared data among multiple servers not only does not degrade the performance of our design compared to the centralized architecture, but also our complexity analysis shows that Integrita performs identically to the centralized architecture concerning the communication and computation both at the user and the server-side. Also, Integrita reduces the storage overhead per server by a factor of N (N is the number of servers). Additionally, Integrita does not rely on cross-server communication in resolving users' read and write requests.

9.2 Future Directions

As future work, Privado can be extended to efficiently support any Boolean function of the attributes in a single advertising request. Another extension would be an extensive analysis of the advertising accuracy using real OSN profiles like Facebook. This will provide an insight into the practical optimum group size and group-based advertising threshold. In invitation-based systems, revocation of a current user imposes an overhead that is linear in the size of the system i.e., the total number of users. An efficient revocation mechanism for invitation-based system is an open research direction. Additionally, one can extend the design of Integrita to support view-consistency in the adversarial model where users may also be corrupted.

BIBLIOGRAPHY

[1] S. R. Chowdhury, A. R. Roy, M. Shaikh, and K. Daudjee, "A taxonomy of decentralized online social networks," *Peer-to-Peer Networking and Applications*, pp. 1–17, 2014.

[2] L. Schwittmann, M. Wander, C. Boelmann, and T. Weis, "Privacy preservation in decentralized online social networks," *IEEE Internet Computing*, p. 1, 2013.

[3] S. Taheri-Boshrooyeh, A. Küpçü, and Ö. Özkasap, "Security and privacy of distributed online social networks," in *IEEE 35th International Conference on Distributed Computing Systems Workshops*. IEEE, 2015, pp. 112–119.

[4] A. Bielenberg, L. Helm, A. Gentilucci, D. Stefanescu, and H. Zhang, "The growth of diaspora-a decentralized online social network in the wild," in *Computer Communications Workshops (INFOCOM WKSHPS), IEEE Conference on*. IEEE, 2012, pp. 13–18.

[5] A. Shakimov, H. Lim, R. Cáceres, L. P. Cox, K. Li, D. Liu, and A. Varshavsky, "Vis-a-vis: Privacy-preserving online social networking via virtual individual servers," in *Third International Conference on Communication Systems and Networks*. IEEE, 2011, pp. 1–10.

[6] P. Stuedi, I. Mohomed, M. Balakrishnan, Z. M. Mao, V. Ramasubramanian, D. Terry, and T. Wobber, "Contrail: Enabling decentralized social networks on smartphones," in *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, 2011, pp. 41–60.

[7] B. Krishnamurthy and C. E. Wills, "Characterizing privacy in online social networks," in *WOSN*. ACM, 2008.

[8] A. Narayanan and V. Shmatikov, "De-anonymizing social networks," in *Security and Privacy*. IEEE, 2009.

[9] A. J. Feldman, A. Blankstein, M. J. Freedman, and E. W. Felten, "Social networking with frientegrity: Privacy and integrity with an untrusted provider." in *USENIX*, 2012.

[10] E. De Cristofaro, C. Soriente, G. Tsudik, and A. Williams, "Hummingbird: Privacy at the time of twitter," in *Security and Privacy (SP)*. IEEE, 2012.

[11] J. Sun, X. Zhu, and Y. Fang, "A privacy-preserving scheme for online social networks with efficient revocation," in *INFOCOM*. IEEE, 2010.

[12] A. Tootoonchian, S. Saroiu, Y. Ganjali, and A. Wolman, "Lockr: better privacy for social networks," in *CoNEXT*. ACM, 2009.

[13] A. Barenghi, M. Beretta, A. Di Federico, and G. Pelosi, "Snake: An end-to-end encrypted online social network," in *ICESS*. IEEE, 2014.

[14] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin, "Persona: an online social network with user-defined privacy," in *ACM SIGCOMM*, 2009.

[15] S. Buchegger, D. Schiöberg, L.-H. Vu, and A. Datta, "Peerson: P2p social networking: early experiences and insights," in *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*. ACM, 2009, pp. 46–52.

[16] N. Z. Gong and D. Wang, "On the security of trustee-based social authentications," *IEEE transactions on information forensics and security*, vol. 9, no. 8, pp. 1251–1263, 2014.

[17] G. P. Malar and C. E. Shyni, "Facebookfs trustee based social authentication," in *Int. J. Emerg. Technol. Comput. Sci. Electron*, vol. 12, no. 4, 2015, pp. 224–230.

[18] "https://telegram.org/tour/groups."

[19] S. Mahmood, "Online social networks: Privacy threats and defenses," in *Security and Privacy Preserving in Social Networks*. Springer, 2013, pp. 47–71.

[20] A. Chaabane, G. Acs, M. A. Kaafar *et al.*, "You are what you like! information leakage through users' interests," in *Proceedings of the 19th Annual Network & Distributed System Security Symposium (NDSS)*, 2012.

[21] S. T. Boshrooyeh, A. Küpçü, and Ö. Özkasap, "Privado: Privacy-preserving group-based advertising using multiple independent social network providers." *IACR Cryptology ePrint Archive*, vol. 2019, p. 372.

[22] Q. Zheng and S. Xu, "Verifiable delegated set intersection operations on outsourced encrypted data," in *IC2E*. IEEE, 2015.

[23] F. Kerschbaum, "Collusion-resistant outsourcing of private set intersection," in *Applied Computing*. ACM, 2012.

[24] S. T. Boshrooyeh, A. Küpçü, and Ö. Özkasap, "Anonyma: Anonymous invitation-only registration in malicious adversarial model." *IACR Cryptology ePrint Archive*, vol. 2019, p. 1215.

[25] ——, "Integrita: Protecting view-consistency in online social network with federated servers." *IACR Cryptology ePrint Archive*, vol. 2019, p. 1223.

[26] A. J. Feldman, W. P. Zeller, M. J. Freedman, and E. W. Felten, "Sporc: Group collaboration using untrusted cloud resources." in *OSDI*, vol. 10, 2010, pp. 337–350.

[27] A. Sattikar and D. R. Kulkarni, "A review of security and privacy issues in social networking," *International Journal of Computer Science and Information Technologies*, vol. 2, no. 6, pp. 2784–2787, 2011.

[28] M. Beye, A. Jeckmans, Z. Erkin, P. Hartel, R. Lagendijk, and Q. Tang, "Literature overview-privacy in online social networks," Centre for Telematics and Information Technology, University of Twente, 2010.

[29] E. Novak and Q. Li, "A survey of security and privacy in online social networks," *College of William and Mary Computer Science Technical Report*, 2012.

[30] A. Verma, D. Kshirsagar, and S. Khan, "Privacy and security: Online social networking," *International Journal of Advanced Computer Research*, vol. 3, no. 8, pp. 310–315, 2013.

[31] M. M. Lucas and N. Borisov, "Flybynight: mitigating the privacy risks of social networking," in *Proceedings of the 7th ACM workshop on Privacy in the electronic society*. ACM, 2008, pp. 1–8.

[32] M. Conti, A. Hasani, and B. Crispo, "Virtual private social networks," in *Proceedings of the first ACM conference on Data and application security and privacy*. ACM, 2011, pp. 39–50.

[33] E. De Cristofaro, C. Soriente, G. Tsudik, and A. Williams, "Hummingbird: Privacy at the time of twitter," in *Security and Privacy (SP), IEEE Symposium on*. IEEE, 2012, pp. 285–299.

[34] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin, "Persona: an online social network with user-defined privacy," in *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4. ACM, 2009, pp. 135–146.

[35] S.-W. Seong, J. Seo, M. Nasielski, D. Sengupta, S. Hangal, S. K. Teh, R. Chu, B. Dodson, and M. S. Lam, "Prpl: a decentralized social networking infrastructure," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*. ACM, 2010, p. 8.

[36] S. Buchegger, D. Schiöberg, L.-H. Vu, and A. Datta, "Peerson: P2p social networking: early experiences and insights," in *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*. ACM, 2009, pp. 46–52.

[37] L. A. Cutillo, R. Molva, and T. Strufe, "Safebook: A privacy-preserving online social network leveraging on real-life trust," *Communications Magazine, IEEE*, vol. 47, no. 12, pp. 94–101, 2009.

[38] S. Nilizadeh, S. Jahid, P. Mittal, N. Borisov, and A. Kapadia, "Cachet: a decentralized architecture for privacy preserving social networking with caching," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM, 2012, pp. 337–348.

[39] R. Sharma and A. Datta, "Supernova: Super-peers based architecture for decentralized online social networks," in *Communication Systems and Networks (COMSNETS), Fourth International Conference on*. IEEE, 2012, pp. 1–10.

[40] D. Sandler and D. S. Wallach, "Birds of a fethr: open, decentralized micropublishing." in *IPTPS*, 2009, p. 1.

[41] T. Xu, Y. Chen, J. Zhao, and X. Fu, "Cuckoo: towards decentralized, socio-aware online microblogging services and data measurements," in *Proceedings of the 2nd ACM International Workshop on Hot Topics in Planet-scale Measurement*. ACM, 2010, p. 4.

[42] S. Guha, K. Tang, and P. Francis, "Noyb: Privacy in online social networks," in *Proceedings of the first workshop on Online social networks*. ACM, 2008, pp. 49–54.

[43] J. Katz and Y. Lindell, *Introduction to modern cryptography*. CRC Press, 2014.

[44] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Advances in Cryptology–EUROCRYPT*. Springer, 2005, pp. 457–473.

[45] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Security and Privacy, SP'07. IEEE Symposium on.* IEEE, 2007, pp. 321–334.

[46] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security.* Acm, 2006, pp. 89–98.

[47] A. Fiat and M. Naor, "Broadcast encryption," in *Advances in Cryptology—CRYPTO'93.* Springer, 1994, pp. 480–491.

[48] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Advances in cryptology.* Springer, 1985, pp. 47–53.

[49] C. Delerablée, "Identity-based broadcast encryption with constant size ciphertexts and private keys," in *Advances in Cryptology–ASIACRYPT.* Springer, 2007, pp. 200–215.

[50] F. Raji, A. Miri, M. D. Jazi, and B. Malek, "Online social network with flexible and dynamic privacy policies," in *Computer Science and Software Engineering (CSSE), CSI International Symposium on.* IEEE, 2011, pp. 135–142.

[51] A. J. Feldman, A. Blankstein, M. J. Freedman, and E. W. Felten, "Privacy and integrity are possible in the untrusted cloud." *IEEE Data Eng. Bull.*, vol. 35, no. 4, pp. 73–82, 2012.

[52] S. Jarecki and X. Liu, "Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection," in *Theory of Cryptography.* Springer, 2009, pp. 577–594.

[53] T. Paul, A. Famulari, and T. Strufe, "A survey on decentralized online social networks," *Computer Networks*, vol. 75, pp. 437–452, 2014.

[54] A. Datta, S. Buchegger, L.-H. Vu, T. Strufe, and K. Rzadca, "Decentralized online social networks," in *Handbook of Social Network Technologies and Applications*. Springer, 2010, pp. 349–378.

[55] B. Carminati, E. Ferrari, and M. Viviani, "Security and trust in online social networks," *Synthesis Lectures on Information Security, Privacy, & Trust*, vol. 4, no. 3, pp. 1–120, 2013.

[56] P. Stuedi, I. Mohomed, M. Balakrishnan, Z. M. Mao, V. Ramasubramanian, D. Terry, and T. Wobber, "Contrail: Enabling decentralized social networks on smartphones," in *Middleware*. Springer, 2011, pp. 41–60.

[57] C. Zhang, J. Sun, X. Zhu, and Y. Fang, "Privacy and security for online social networks: challenges and opportunities," *Network, IEEE*, vol. 24, no. 4, pp. 13–18, 2010.

[58] A. Juels, M. Luby, and R. Ostrovsky, "Security of blind digital signatures," in *Advances in Cryptology—CRYPTO'97*. Springer, 1997, pp. 150–164.

[59] O. Goldreich and Y. Oren, "Definitions and properties of zero-knowledge proof systems," *Journal of Cryptology*, vol. 7, no. 1, pp. 1–32, 1994.

[60] M. Backes, M. Maffei, and K. Pecina, "A security api for distributed social networks." in *NDSS*, vol. 11, 2011, pp. 35–51.

[61] C. Huang, Y. Chen, W. Wang, Y. Cui, H. Wang, and N. Du, "A novel social search model based on trust and popularity," in *Broadband Network and Multimedia Technology (IC-BNMT), 3rd IEEE International Conference on*. IEEE, 2010, pp. 1030–1034.

[62] P. Jain, P. Jain, and P. Kumaraguru, "Call me maybe: Understanding nature and risks of sharing mobile numbers on online social networks," in *Proceedings of the first ACM conference on Online social networks*. ACM, 2013, pp. 101–106.

[63] E. Sarigol, D. Garcia, and F. Schweitzer, "Online privacy as a collective phenomenon," in *Proceedings of the second edition of the ACM conference on Online social networks*. ACM, 2014, pp. 95–106.

[64] S. Guha, B. Cheng, and P. Francis, "Privad: Practical privacy in online advertising." in *NSDI*, 2011.

[65] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas, "Adnostic: Privacy preserving targeted advertising." in *NDSS*, 2010.

[66] P. Joshi and C.-C. Kuo, "Security and privacy in online social networks: A survey," in *Multimedia and Expo (ICME), IEEE International Conference on*. IEEE, 2011, pp. 1–6.

[67] S. T. Boshrooyeh, A. Küpçü, and Ö. Özkasap, "Ppad: Privacy preserving group-based advertising in online social networks," *IFIP Networking Conference*, 2018.

[68] D. Biswas, S. Haller, and F. Kerschbaum, "Privacy-preserving outsourced profiling," in *CEC*. IEEE, 2010.

[69] A. Juels, "Targeted advertising... and privacy too," in *CT-RSA*, 2001.

[70] F. Kerschbaum, "Outsourced private set intersection using homomorphic encryption," in *CCS*. ACM, 2012.

[71] S. Kamara, P. Mohassel, M. Raykova, and S. Sadeghian, "Scaling private set intersection to billion-element sets," in *FC*, 2014.

[72] C. Patsakis, A. Zigomitros, and A. Solanas, "Privacy-aware genome mining: Server-assisted protocols for private set intersection and pattern matching," in *CBMS*. IEEE, 2015.

[73] B. Pinkas, T. Schneider, and M. Zohner, "Scalable private set intersection based on ot extension," *ACM Transactions on Privacy and Security (TOPS)*, vol. 21, no. 2, p. 7, 2018.

[74] D. He, M. Ma, S. Zeadally, N. Kumar, and K. Liang, "Certificateless public key authenticated encryption with keyword search for industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3618–3627, 2018.

[75] Q. Huang and H. Li, "An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks," *Information Sciences*, vol. 403, pp. 1–14, 2017.

[76] B. Zhu, J. Sun, J. Qin, and J. Ma, "The public verifiability of public key encryption with keyword search," in *International Conference on Mobile Networks and Management*. Springer, 2017, pp. 299–312.

[77] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *International conference on the theory and applications of cryptographic techniques*. Springer, 2004, pp. 506–522.

[78] J. Li, X. Lin, Y. Zhang, and J. Han, "Ksf-oabe: outsourced attribute-based encryption with keyword search function for cloud storage," *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 715–725, 2017.

[79] R. Cramer, I. Damgård, and J. B. Nielsen, "Multiparty computation from threshold homomorphic encryption," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2001, pp. 280–300.

[80] I. Damgård and J. B. Nielsen, "Universally composable efficient multiparty computation from threshold homomorphic encryption," in *Annual International Cryptology Conference*. Springer, 2003, pp. 247–264.

[81] T. P. Jakobsen, J. B. Nielsen, and C. Orlandi, "A framework for outsourcing of secure computation," in *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security*. ACM, 2014, pp. 81–92.

[82] P. Ananth, N. Chandran, V. Goyal, B. Kanukurthi, and R. Ostrovsky, "Achieving privacy in verifiable computation with multiple servers–without fhe and without pre-processing," in *International Workshop on Public Key Cryptography*. Springer, 2014, pp. 149–166.

[83] B. Schoenmakers and M. Veeningen, "Universally verifiable multiparty computation from threshold homomorphic cryptosystems," in *International Conference on Applied Cryptography and Network Security*. Springer, 2015, pp. 3–22.

[84] T. Moran and M. Naor, "Split-ballot voting: everlasting privacy with distributed trust," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 2, p. 16, 2010.

[85] X. Zou, H. Li, Y. Sui, W. Peng, and F. Li, "Assurable, transparent, and mutual restraining e-voting involving multiple conflicting parties," in *INFOCOM, Proceedings IEEE*, 2014, pp. 136–144.

[86] O. Oksuz, I. Leontiadis, S. Chen, A. Russell, Q. Tang, and B. Wang, "Sevdsi: Secure, efficient and verifiable data set intersection," Cryptology ePrint Archive, Report 2017/215.(2017). http://ia. cr/2017/215, Tech. Rep.

[87] S. Kamara, P. Mohassel, and B. Riva, "Salus: a system for server-aided secure function evaluation," in *CCS*. ACM, 2012.

[88] F. Kerschbaum, "Adapting privacy-preserving computation to the service provider model," in *CSE*. IEEE, 2009.

[89] A. Herzberg and H. Shulman, "Oblivious and fair server-aided two-party computation," *Information Security Technical Report*, 2013.

[90] P. Mohassel, O. Orobets, and B. Riva, "Efficient server-aided 2pc for mobile phones," *Proceedings on Privacy Enhancing Technologies*, no. 2, pp. 82–99, 2016.

[91] S. Kamara, P. Mohassel, and M. Raykova, "Outsourcing multi-party computation." *IACR Cryptology ePrint Archive*, 2011.

[92] H. Carter, B. Mood, P. Traynor, and K. Butler, "Outsourcing secure two-party computation as a black box," in *Cryptology and Network Security*, 2015.

[93] M. Blanton and F. Bayatbabolghani, "Efficient server-aided secure two-party function evaluation with applications to genomic computation," *Proceedings on Privacy Enhancing Technologies*, 2016.

[94] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Annual Cryptology Conference*. Springer, 2012, pp. 643–662.

[95] N. Volgushev, M. Schwarzkopf, B. Getchell, M. Varia, A. Lapets, and A. Bestavros, "Conclave: secure multi-party computation on big data," in *European Conference on Computer Systems*, 2019.

[96] H. Carter, B. Mood, P. Traynor, and K. Butler, "Secure outsourced garbled circuit evaluation for mobile devices," *Journal of Computer Security*, vol. 24, no. 2, pp. 137–180, 2016.

[97] C. Hazay and Y. Lindell, *Efficient secure two-party protocols: Techniques and constructions*. Springer Science & Business Media, 2010.

[98] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[99] B. Schneier and P. Sutherland, "Applied cryptography: Protocols, algorithms, and source code in c." John Wiley & Sons, Inc., 1995.

[100] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. CRC press, 2014.

[101] C. Hazay, G. L. Mikkelsen, T. Rabin, and T. Toft, "Efficient rsa key generation and threshold paillier in the two-party setting," in *Cryptographers' Track at the RSA Conference*. Springer, 2012, pp. 313–331.

[102] ——, "Efficient rsa key generation and threshold paillier in the two-party setting," *IACR Cryptology ePrint Archive*, p. 494, 2011.

[103] Y. Lindell, "Tutorials on the foundations of cryptography." Springer, 2017.

[104] N. Pettersen, "Applications of paillier s cryptosystem." Master's thesis, NTNU, 2016.

[105] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology—CRYPTO'86*. Springer, 1986, pp. 186–194.

[106] K. Sampigethaya and R. Poovendran, "A survey on mix networks and their secure applications," *Proceedings of the IEEE*, vol. 94, no. 12, pp. 2142–2181, 2006.

[107] K. Peng and F. Bao, "A shuffling scheme with strict and strong security," in *Emerging Security Information Systems and Technologies (SECURWARE), Fourth International Conference on*. IEEE, 2010, pp. 201–206.

[108] H. Kılınç and A. Küpçü, "Optimally efficient multi-party fair exchange and fair secure multi-party computation," in *Cryptographers' Track at the RSA Conference*. Springer, 2015, pp. 330–349.

[109] "https://pieregister.com/features/invitation-based-registrations/."

[110] J. Brainard, A. Juels, R. L. Rivest, M. Szydlo, and M. Yung, "Fourth-factor authentication: somebody you know," in *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, pp. 168–178.

[111] E. Bresson, J. Stern, and M. Szydlo, "Threshold ring signatures and applications to ad-hoc groups," in *Annual International Cryptology Conference*. Springer, 2002, pp. 465–480.

[112] C. A. Melchor, P.-L. Cayrel, P. Gaborit, and F. Laguillaumie, "A new efficient threshold ring signature scheme based on coding theory," in *IEEE Transactions on Information Theory*, vol. 57, no. 7. IEEE, 2011, pp. 4833–4842.

[113] J. K. Liu, V. K. Wei, and D. S. Wong, "A separable threshold ring signature scheme," in *International Conference on Information Security and Cryptology*. Springer, 2003, pp. 12–26.

[114] X. Boyen, "Mesh signatures," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2007, pp. 210–227.

[115] J. Benaloh and D. Tuinstra, "Receipt-free secret-ballot elections," in *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*. ACM, 1994, pp. 544–553.

[116] M. J. Radwin and P. Klein, "An untraceable, universally verifiable voting scheme," in *Seminar in Cryptology*, 1995, pp. 829–834.

[117] S. T. Boshrooyeh and A. Küpçü, "Inonymous: Anonymous invitation-based system," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2017, pp. 219–235.

[118] A. Kiayias and M. Yung, "The vector-ballot e-voting approach," in *International Conference on Financial Cryptography*. Springer, 2004, pp. 72–89.

[119] A. Schneider, C. Meter, and P. Hagemeister, "Survey on remote electronic voting," in *arXiv preprint arXiv:1702.02798*, 2017.

[120] D. G. Nair, V. Binu, and G. S. Kumar, "An improved e-voting scheme using secret sharing based secure multi-party computation," in *arXiv preprint arXiv:1502.07469*, 2015.

[121] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," in *Communications of the ACM*, vol. 24, no. 2. ACM, 1981, pp. 84–90.

[122] T. Isshiki and K. Tanaka, "An (n–t)-out-of-n threshold ring signature scheme," in *Australasian Conference on Information Security and Privacy*. Springer, 2005, pp. 406–416.

[123] D. Bogdanov, "Foundations and properties of shamir's secret sharing scheme research seminar in cryptography," *University of Tartu, Institute of Computer Science May 1st*, 2007.

[124] L. Harn and C. Lin, "Authenticated group key transfer protocol based on secret sharing," *IEEE transactions on computers*, vol. 59, no. 6, pp. 842–846, 2010.

[125] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[126] A. Roy and S. Karforma, "A survey on digital signatures and its applications," *Journal of Computer and Information Technology*, vol. 3, no. 1, pp. 45–69, 2012.

[127] J. Groth, "Non-interactive zero-knowledge arguments for voting," in *International Conference on Applied Cryptography and Network Security*. Springer, 2005, pp. 467–482.

[128] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *Annual International Cryptology Conference*. Springer, 1992, pp. 89–105.

[129] J. Yu, F. Kong, X. Cheng, R. Hao, and G. Li, "One forward-secure signature scheme using bilinear maps and its applications," *Information Sciences*, vol. 279, pp. 60–76, 2014.

[130] A. Rosen, "A note on constant-round zero-knowledge proofs for np," in *Theory of Cryptography Conference*. Springer, 2004, pp. 191–202.

[131] D. W. Kravitz, "Digital signature algorithm," Jul. 27 1993, uS Patent 5,231,668.

[132] B. Schoenmakers, "A simple publicly verifiable secret sharing scheme and its application to electronic voting," in *Annual International Cryptology Conference*. Springer, 1999, pp. 148–164.

[133] P. Williams, R. Sion, and D. Shasha, "The blind stone tablet: Outsourcing durability," in *16th annual network and distributed system security symposium*, 2009.

[134] D. Mazieres and D. Shasha, "Building secure file systems out of byzantine storage," in *Proceedings of the twenty-first annual symposium on Principles of distributed computing*. ACM, 2002, pp. 108–117.

[135] C. Cachin and O. Ohrimenko, "Verifying the consistency of remote untrusted services with conflict-free operations," *Information and Computation*, vol. 260, pp. 72–88, 2018.

[136] C. Cachin, I. Keidar, and A. Shraer, "Fork sequential consistency is blocking," *Information Processing Letters*, vol. 109, no. 7, pp. 360–364, 2009.

[137] P. Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish, "Depot: Cloud storage with minimal trust," *ACM Transactions on Computer Systems (TOCS)*, vol. 29, no. 4, p. 12, 2011.

[138] M. Brandenburger, C. Cachin, and N. Knežević, "Don't trust the cloud, verify: Integrity and consistency for cloud object stores," *ACM Transactions on Privacy and Security (TOPS)*, vol. 20, no. 3, p. 8, 2017.

[139] S. Nilizadeh, S. Jahid, P. Mittal, N. Borisov, and A. Kapadia, "Cachet: a decentralized architecture for privacy preserving social networking with caching," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM, 2012, pp. 337–348.

[140] M. T. Goodrich, J. Lentini, M. Shin, R. Tamassia, and R. Cohen, "Design and implementation of a distributed authenticated dictionary and its applications," Technical report, Center for Geometric Computing, Brown University, Tech. Rep., 2002.

[141] M. T. Goodrich and R. Tamassia, "Efficient authenticated dictionaries with skip lists and commutative hashing," Aug. 14 2007, uS Patent 7,257,711.

[142] D. J. Polivy and R. Tamassia, "Authenticating distributed data using web services and xml signatures." in *XML Security*, 2002, pp. 80–89.

[143] R. Tamassia, "Authenticated data structures," in *European symposium on algorithms*. Springer, 2003, pp. 2–5.

[144] M. T. Goodrich, R. Tamassia, and J. Hasić, "An efficient dynamic and distributed cryptographic accumulator," in *International Conference on Information Security*. Springer, 2002, pp. 372–388.

[145] B. Palazzi, "Outsourced storage services: Authentication and security visualization," Ph.D. dissertation, Ph. D. thesis, Roma Tre University, 2009.

[146] J. Li and D. Maziéres, "Beyond one-third faulty replicas in byzantine fault tolerant systems." in *NSDI*, 2007.

[147] P. Civit, S. Gilbert, and V. Gramoli, "Polygraph: Accountable byzantine agreement," *IACR Cryptology ePrint Archive*, p. 587, 2009.

[148] S. A. Crosby and D. S. Wallach, "Efficient data structures for tamper-evident logging," in *USENIX Security Symposium*, 2009, pp. 317–334.

[149] K. Kurosawa and Y. Desmedt, "A new paradigm of hybrid encryption scheme," in *Annual International Cryptology Conference*. Springer, 2004, pp. 426–442.

[150] T. Strufe, "Safebook: A privacy-preserving online social network leveraging on real-life trust," *IEEE Communications Magazine*, vol. 95, 2009.

[151] K. Graffi, C. Gross, D. Stingl, D. Hartung, A. Kovacevic, and R. Steinmetz, "Lifesocial. kom: A secure and p2p-based solution for online social networks," in *Consumer Communications and Networking Conference (CCNC), IEEE*. IEEE, 2011, pp. 554–558.

[152] A. Loupasakis, N. Ntarmos, P. Triantafillou, and D. Makreshanski, "exo: Decentralized autonomous scalable social networking." in *CIDR*, 2011, pp. 85–95.

[153] F. Tegeler, D. Koll, and X. Fu, "Gemstone: empowering decentralized social networking with high data availability," in *IEEE Global Telecommunications Conference-GLOBECOM*. IEEE, 2011, pp. 1–6.

[154] R. Narendula, T. G. Papaioannou, and K. Aberer, "Privacy-aware and highly-available osn profiles," in *19th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*. IEEE, 2010, pp. 211–216.

[155] S. Han, H. Shen, T. Kim, A. Krishnamurthy, T. Anderson, and D. Wetherall, "Metasync: File synchronization across multiple untrusted storage services," in *2015 {USENIX} Annual Technical Conference ({USENIX} {ATC} 15)*, pp. 83–95.