UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

IMPLEMENTING SECURE REPORTING OF SEXUAL MISCONDUCT

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS MENCIÓN COMPUTACIÓN

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERA CIVIL EN COMPUTACIÓN

ILANA MERGUDICH THAL

PROFESOR GUÍA:
ALEJANDRO HEVIA ANGULO

MIEMBROS DE LA COMISIÓN:
JÉRÉMY BARBAY
TOMÁS BARROS ARANCIBIA
PATRICIO GALDAMES SEPÚLVEDA

Este trabajo ha sido parcialmente financiado por ANID - Subdirección de Capital Humano/Magíster Nacional/2020

SANTIAGO DE CHILE
2021

Resumen

RESUMEN DE LA TESIS PARA OPTAR AL TÍTULO DE: Ingeniera Civil en Computación y grado de Magíster en Ciencias mención Computación
POR: Ilana Mergudich Thal
FECHA: 1/12/21
PROFESOR GUÍA: Alejandro Hevia Angulo

Reportar acoso sexual es extremadamente complejo. A pesar de que más víctimas reportan cada año, existe un importante porcentaje de personas que no hace acusaciones formales [50]. Estudios han mostrado que la mayoría de los episodios de acoso sexual ocurren por parte de personas que han acosado previamente y muestran que las víctimas están más dispuestas a reportar si saben que existen otras víctimas del mismo agresor [19]. Recientemente, Kuykendall, Krawczyk y Rabin [43] propusieron WhoToo, un sistema en que las identidades de quien acusa y quien es acusado son protegidas hasta que un número preestablecido (quórum) de personas reportan al mismo acosador. En esta tesis, revisamos el protocolo desde una perspectiva de implementación, elucidando clarificaciones necesarias y optimizaciones posibles.

Primero identificamos diversas operaciones clave en que la implementación no era clara en la propuesta original. En uno de los casos, si la operación se hubiese implementado de manera directa mediante el uso de otras herramientas de WhoToo se hubiese comprometido el anonimato. La solución para otro caso fue simple pero requirió una nueva (aunque directa) demostración de seguridad. Estas soluciones, a pesar de ser pequeñas, eran fundamentales para un systema cuyo diseño enfatizaba practicidad y operaciones rápidas.

Luego revisamos la eficiencia de WhoToo. Usando una función pseudo aleatoria de *input* distribuido y una variante de encriptación robusta y anónima basada en identidad, mejoramos la eficiencia de la detección de duplicados y acusaciones coincidentes. Dadas N acusaciones, nuestra solución requiere $O(1)$ operaciones distribuidas (la primitiva más costosa en WhoToo) para detectar duplicados y acusaciones coincidentes una vez alcanzado el quórum.

Adicionalmente, entrevistamos a expertos y realizamos *focus groups* con potenciales usuarios, lo que llevó a una discusión sobre nuevos requerimientos.

Nuestros resultados dan pie a `WhoToo`$^+$, una variante práctica y más eficiente de WhoToo que preserva las garantías de seguridad originales. Más aún, implementamos un prototipo de `WhoToo`$^+$ y analizamos su eficiencia teórica y experimental.

Abstract

Reporting sexual assault or harassment is notoriously difficult, and even though more victims are coming forward every year, a significant percentage of victims do not formally report it [50]. Studies have shown that most sexual assault episodes occur by repeat perpetrators and that people are more likely to report if they know that other victims of the same aggressor exist [19]. Recently, Kuykendall, Krawczyk and Rabin [43] proposed WhoToo, a system in which the identities of the accuser and the accused are protected until a certain pre-specified number (quorum) of victims reports the same perpetrator. In this thesis, we revisit this protocol from an implementation perspective, shedding light on necessary clarifications and optimizations.

We first identified several key operations whose implementation was left unclear in the original proposal. One of such operations, if implemented in a straightforward fashion by using other WhoToo subroutines would compromise anonymity. Fixes for another were simple but required a new (but straightforward) security proof. Such fixes, although rather minor, were important for a system whose design emphasizes practicality and fast operations.

We then revisited WhoToo's efficiency. Using a Distributed Input PRF and a variant of Robust Anonymous IBE Encryption, we improved detection of duplicated and matching accusations. Given N accusations, our solution requires $O(1)$ instead of $O(N)$ distributed operations (the most expensive primitive in WhoToo) to detect duplicates and matching accusations once the quorum is reached.

Additionally, we interviewed experts and conducted focus groups with potential users which lead to a discussion of new requirements.

Our results yield `WhoToo`$^+$, a practical and more efficient variant of WhoToo that preserves the original security guarantees. Moreover, we implemented a prototype of `WhoToo`$^+$and analyzed its theoretical and experimental efficiency.

A las mujeres de mi familia.
A las que están cerca, las que están lejos y las que ya no están.
Su pasión por lo que hacen es contagiosa y me inspira todos los días.

Agradecimientos

A mi mamá, que me transimitió su pasión por aprender desde muy chica, me apoyó y acompañó siempre. Saber que siempre vas a estar ahí para apoyarme me hace sentir que soy capaz de cualquier cosa. Gracias por tu cariño infinito, no tengo palabras para expresar lo agradecida que estoy, pero todos mis logros son gracias a ti.

A mi papá, por su cariño incondicional y por mostrarme la importancia de disfrutar el proceso. Gracias por regalonearme, gracias por tu compañía en todos esos trayectos y gracias por estar siempre cerca a pesar de la distancia.

A mi hermana, por ser mi modelo a seguir, por empujarme a ser y hacer lo mejor posible. Gracias por ser mi compañera de vida, gracias por tu cariño y apoyo todos los días de mi vida.

A Max, por estar siempre ahí para mi, por escucharme, apoyarme, abrazarme y acompañarme cada vez que lo necesité. Gracias por enseñarme que a veces necesito parar y dejarme querer, nunca hubiese podido terminar esta tesis sin ese descubrimiento.

A mi tía Karen, mi tío Lucho, mi prima Tamara, mi tía Ety y toda mi familia por estar tan presentes en cada detalle de mi vida.

A mis amigas Clau, Dani S, Dani P y Dani C, que fueron mi todo para mi proceso en la universidad. No se imaginan lo importante que fueron y son para mi. Gracias por su cariño, sus risas, sus abrazos, sus preguntas, gracias por ser parte de mi vida, por estar siempre ahí, por escucharme, por enseñarme y acompañarme todos estos años. Las quiero infinito y agradezco todos los días habernos encontrado.

A la Coni, la Cata, Felipe y Seba, por creer en mi desde siempre. Gracias Coni por inspirarme y por impulsarme a perseguir propósitos inalcanzables.

A Lambda, mi compañero de todos los días, por entregarme amor y alegría infinita.

A todas las mujeres de mi vida, por inspirarme y mostrarme la importancia de trabajar en esto.

A mi profesor guía Alejandro Hevia, porque me dio espacio para explorar y me ayudó a encontrar un tema que realmente me hiciera sentido. No sólo me enseñó a investigar y colaborar en el ámbito académico, sino que se preocupó por mi formación como persona en un sentido mucho más amplio.

De verdad muchas gracias.

Contents

List of Tables **x**

List of Figures **xii**

1 Introduction **1**
1.1 Background and Motivation . . . 1
1.2 Problem . . . 2
1.3 Hypothesis . . . 4
1.4 Objectives . . . 4
1.4.1 General objective . . . 4
1.4.2 Specific objectives . . . 4
1.5 Methodology . . . 5
1.6 Contributions . . . 6
1.7 Related Work . . . 7
1.7.1 Callisto . . . 7
1.7.2 Who Too . . . 7
1.7.3 Secure Allegations Escrow (SAE) . . . 8
1.8 Structure . . . 8

2 Cryptographic Tools **9**
2.1 Preliminaries . . . 9
2.1.1 Notation . . . 9
2.1.2 Probabilistic Polynomial Time (PPT) Algorithm . . . 9
2.1.3 Bilinear pairings . . . 10
2.1.4 Computational Assumptions . . . 10
2.1.5 Communication channels . . . 11
2.1.6 Security Requirements and Threat Model . . . 11
2.2 Building Blocks . . . 12
2.2.1 Pseudo-Random Functions: . . . 12
2.2.2 Hash Function . . . 12
2.2.3 Message Authentication Code (MAC) . . . 13
2.2.4 ElGamal Encryption . . . 14
2.2.5 Zero-knowledge proofs . . . 16
2.2.6 Threshold Operations . . . 17
2.2.7 Distributed Group Signatures . . . 19
2.2.8 Privacy-Preserving Multisets . . . 20

3 A First Review of WhoToo **22**
3.1 Protocol Overviews . . . 22
3.1.1 WhoToo: An Introduction . . . 22
3.1.2 WhoToo$^+$ Overview . . . 24
3.2 Two Issues in WhoToo . . . 25
3.2.1 Securely Evaluating Quorum in WhoToo . . . 25
3.2.2 Identifying Duplicated Accusations . . . 27

4 Improving WhoToo **29**
4.1 Duplicate Revision . . . 29
4.1.1 Distributed Input Pseudorandom Functions (DIPRF): . . . 29
4.1.2 Avoiding mismatched accusations: . . . 32
4.2 Matching accusations . . . 32

5 Variants and Extensions **37**
5.1 Discarded ideas . . . 37
5.2 Flexible quorum . . . 38
5.3 Role . . . 39
5.4 Unknown perpetrator . . . 39
5.5 Additional Public Information . . . 39
5.5.1 Contact for further investigation . . . 39
5.5.2 Repetition counter . . . 40
5.6 Updates . . . 40

6 Description and Security of the New Protocol **41**
6.1 WhoToo$^+$ Description . . . 41
6.2 Security Analysis . . . 45

7 Implementation and Efficiency **52**
7.1 Prototype . . . 52
7.2 Efficiency Analysis and Discussion . . . 53
7.2.1 Theoretical Efficiency . . . 53
7.2.2 Experimental Efficiency . . . 55

8 Concluding Remarks and Future Work **62**
8.1 Concluding Remarks . . . 62
8.2 Future Work . . . 62

9 Bibliography **64**

10 Appendix **69**
10.1 Focus Groups' Guideline . . . 69

List of Tables

7.1 Efficiency comparison given a total number N of accusations, m valid users, a maximum number k of accusations per user and quorum q. 55

List of Figures

1.1 Intuitive comparison between existing sexual assault reporting protocols. . . 3

2.1 Multiplication with shared exponents 19

3.1 Randomized Decryption 26
3.2 Modification in Set.Quorum and PrivatePoly.ZeroTest for consistency with PrivatePoly.Multiply. d stands for the degree of the polynomial. Operations involving shares use arguments with braces (eg. $\{x\}$) as described in Sect. 2.2.6. 26
3.3 Ideal Functionality for PrivatePoly.ZeroTest. 27

4.1 Distributed Input PRF and MAC. Operations involving shares use arguments with braces (eg. $\{x\}$) as described in Sect. 2.2.6. 30
4.2 Strongly Robust Distributed IBE. Operations involving shares use arguments with braces (eg. $\{x\}$) as described in Sect. 2.2.6. 35

5.1 Zero-knowledge proof of equality of ElGamal encodings 40

6.1 The WhoToo$^+$ Protocol 44
6.2 $\mathcal{F}_{\text{WhoToo}^+}$: Ideal Functionality for WhoToo$^+$ 45
6.3 $\mathcal{F}_{DR}$: Ideal Functionality for duplicate revision 49
6.4 $\mathcal{F}_{MA}$: Ideal Functionality for matching accusations 49

7.1 Efficiency comparison for RD and MA Online Operations 55
7.2 Comparison of Total Online Operations 56
7.3 Initialization Time for Different Numbers of Users 57
7.4 Initialization Time for Different Numbers of MACs per User 57
7.5 Online Time for Submission of New Accusations in WhoToo$^+$ 58
7.6 Online Time for Submission of New Accusations that Don't Reach the Quorum 59
7.7 Online Time for Submission of New Accusations that Reach the Quorum . . 60
7.8 Comparison of Online Time for Submission of New Accusations 61

Chapter 1

Introduction

1.1 Background and Motivation

Sexual assault accusations have become more frequent every year, nevertheless, there is still a significant percentage of victims that do not come forward. The issue of sexual assault is specially prevalent in educational settings. A recent study by the Pontificia Universidad Católica de Chile has shown that, in one of the biggest universities in the country, 39,9% of students declare that they have been victims of sexual harassment, yet 65% of them did not formally report it [46]. Similar numbers have been reported for female undergraduate students at other universities [39, 21], showing that a significant number of victims never report it. On the other hand, research has shown that people are more likely to report sexual assault if they know other victims of the same perpetrator exist [19]. Furthermore, an overwhelming majority of sexual misconduct episodes are caused by offenders who have committed sexual assault before [19].

Project Callisto [56] was the first to address this problem considering all these factors. They proposed a protocol in which the identity of the accuser and the accused remained hidden until two victims accused the same perpetrator. Even though it has been shown that the identities are not entirely protected in this protocol, it has been implemented in 13 universities across the United States of America (USA). In only three years, the average time taken to report a sexual assault episode since the moment it occurred has decreased from 11 to 4 months [55], which shows that this is a successful approach to the sexual assault reporting problem.

Since then, two other protocols based on the same premise have been proposed: WhoToo [43] and Secure Allegation Escrows (SAE) [5]. Both of these protocols receive accusations in a distributed setting and define a quorum q so that accusations are revealed only when q accusations against a certain person are submitted. The first provides strong security guarantees, however, it can become increasingly inefficient as the number of unopened accusations in the system grows. SAE is significantly more efficient than WhoToo and even though its security guarantees are stronger than Callisto, it reveals information about matching accusations (accusing the same perpetrator) before the quorum is reached.

Consequently, it is natural to ask which security properties should be guaranteed by a protocol of this kind and if it is possible to design a protocol with such properties. This thesis's original motivation was to define the necessary requisites for a protocol for sexual assault reporting, design a protocol that distributively receives accusations and reveals them only when a specific quorum is reached and prove that this protocol actually guarantees the defined requisites.

1.2 Problem

Considering this problem and the existing solutions, we identify the need for a distributed protocol (i.e, accusations are received in a distributed setting) that does not reveal any partial information about the accusations until a specific quorum is reached, is secure under the simulations model [44] and is efficient enough to be implemented in a university context. Both WhoToo and SAE achieve most of these properties, yet WhoToo becomes increasingly inefficient when there is a backlog of unopened accusations and SAE reveals meta-information about accusation matches before the quorum is reached. Furthermore, we uncovered two gaps in the specification of key aspects of WhoToo (these gaps were not trivial, as they could possibly compromise the protocol security).

Therefore, the problem we aim to solve is to design a protocol that guarantees at least the same security properties as SAE, yet does not allow any server to obtain information or meta-data about the accusations and the relationships between them until the quorum is reached. The protocol should be efficient enough to be implemented in a university context, being at least more efficient than WhoToo.

This can be intuitively represented in Fig 1.1. On one hand, WhoToo (represented in pink) is a protocol that claims to have strong security guarantees, yet adding a new accusation requires $O(N)$ distributed operations. On the other hand, SAE (represented in blue) is not as secure because it reveals meta-information about matching accusations, however, adding a new accusation requires only $O(1)$ distributed operations. We aim to design a protocol in the green area. Our first approach was to start from the SAE protocol and try to improve its security guarantees, this is, go along the blue arrow, willing to sacrifice efficiency in order to achieve this, as long as it was at least more efficient than WhoToo. We later realized that there where key elements in WhoToo that could be replaced to make it more efficient, and because its security guarantees were already stronger, it was better to start from the WhoToo protocol and improve its efficiency, moving along the pink arrow.

Consequently, the problem we aim to solve is to design a protocol that guarantees the following:

1. *Secrecy:* All information about a submitted accusation is protected until the quorum is reached.
2. *Anonymity:* The identities of the accuser and the accused are protected until the quorum is reached.
3. *Accountability:* The identity of the accuser is bound to a real world identity.
4. *Meta-data hiding (MDH):* Other than the total number of accusations, no partial in-

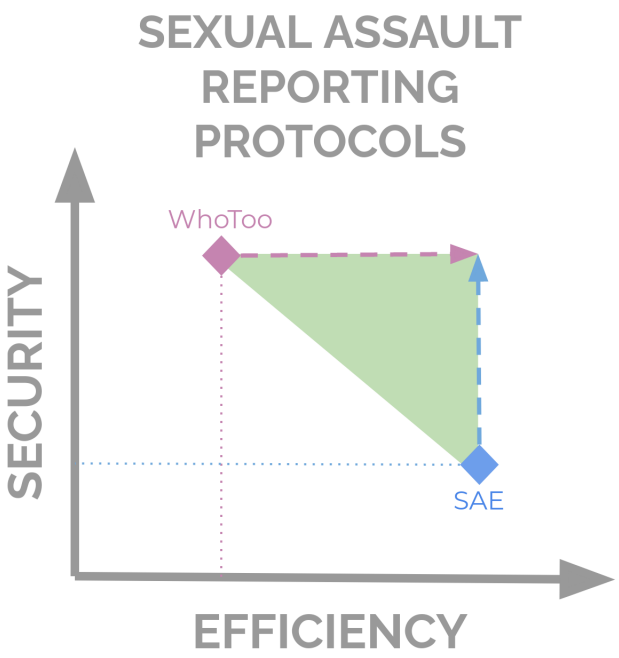

Figure 1.1: Intuitive comparison between existing sexual assault reporting protocols.

formation about the accusations and its matches is leaked.

5. *Distributed threshold authority:* All information is distributed among N servers. While at least $t + 1$ out of N servers are honest, the protocol guarantees secrecy, anonymity, accountability and MDH.
6. *Efficiency:* The protocol is more efficient than WhoToo.

We aim to fix the gaps in the WhoToo description and to improve its efficiency. We identified specific components of this protocol that could be modified or replaced in a more efficient manner. When receiving a new accusation, WhoToo must compare it with every element of a list of unopened accusations, in order to achieve two goals: preventing duplicated accusations (those involving the same pair accuser-accused), and identifying all existent accusations for a given accused, when the quorum has been reached. The protocol's strategy to deal with these two tasks becomes increasingly inefficient as the number of unopened accusations grows: it takes $O(N)$ interactive distributed operations to review a total number of N unopened accusations in the system, for both of the abovementioned goals.

Research Questions

We address this problem through the following research questions:

1. Is it possible to fix the gaps in the specification of WhoToo, guaranteeing the protocol's security properties?
2. Is it possible to improve the efficiency of the WhoToo protocol while preserving secrecy, anonymity, accountability and MDH?
3. Is there an alternative to the duplicate revision process of WhoToo that is more efficient

and does not leak partial information about the accusations?

4. Is there an alternative to the process of finding matching accusations against a specific person proposed by WhoToo that is more efficient and does not leak any information about non-matching accusations?

1.3 Hypothesis

The use of a Distributed Input PRFs (DIPRF) and a variant of strongly robust Identity Based Encryption (IBE) together with threshold cryptography tools allows to fix the gaps and design alternatives to the duplicate revision and matching accusations processes, decreasing the number of distributed operations from $O(N)$ to $O(1)$ while preserving WhoToo's security guarantees. It is possible to formally demonstrate that the modified protocol achieves the abovementioned security properties.

Note that the original hypothesis was the following:

The use of zero-knowledge sets and its variants such as accumulators, oblivious PRFs together and threshold cryptography tools allows to design an alternative to the matching process in SAE to build a protocol that preserves SAE's security guarantees, is more efficient than WhoToo and guarantees meta-data hiding. It is possible to formally demonstrate that the modified protocol achieves the abovementioned security properties.

As mentioned before, we realized that we could obtain better results by improving the efficiency of WhoToo rather than adding meta-data hiding to SAE. Instead of modifying SAE's matching process, we introduced changes in WhoToo's duplicate revision and matching process. Therefore, the tools used to achieve these changed, but the main objective remains the same.

1.4 Objectives

1.4.1 General objective

Design an efficient and secure (with provable security guarantees) protocol that receives and records sexual assault accusations in a distributed manner and does not leak any information about the accusations or the relations between them until the quorum is reached.

1.4.2 Specific objectives

1. Fix the gaps in the WhoToo protocol description.
2. Design an alternative to the WhoToo duplicate revision process that is more efficient and preserves the protocol's security properties.
3. Design an alternative to the WhoToo accusation matching process that is more efficient and preserves the protocol's security properties.
4. Prove that the modified protocol guarantees secrecy, anonymity, accountability and

MDH[1].

5. Formally compare the efficiency of the protocol in a university context with previous solutions.
6. Study and propose a threshold version of strongly robust IBE as a possible alternative to the accusation matching process.
7. Study and evaluate under which conditions users would want to define a variable quorum, this is, each user would define the quorum for their accusation.
8. Study and evaluate under which conditions users would want to define an expiring date for their accusations.
9. Study and evaluate under which conditions users would want to categorize their accusations so that these are only revealed if the quorum is reached within a certain category.
10. Study and evaluate which information should be public under which conditions. For example: The total amount of accusations, the total amount in a specific department, among others.
11. Study and evaluate the need for different matching factors, such as a physical description or partial information about the accused.

The original objectives were 4, 5, 7 and 8. Objectives 1-3, 6 and 9-11 were added during the process of developing this thesis. The first four new objectives were a direct consequence of the decision of improving WhoToo's efficiency rather than adding MDH to SAE. The last three objectives arose during conversations with experts and possible users of such a protocol.

Furthermore, the following specific objectives were included in the original proposal but where not achieved, as they where replaced by the ones mentioned above.

1. Design an alternative to SAE's matching process that achieves MDH.
   This was replaced by objectives 1, 2 and 3.
2. Study and propose an extension to zero-knowledge sets or accumulators in a distributed context as a possible solution by using tools from distributed zero-knowledge proofs.
   This was replaced by objective 6. Following the previous change, the tools used to achieve this also changed.

1.5 Methodology

In order to achieve these objectives, we followed these steps:

1. Bibliography revision: Study Multi Party Computation (MPC) tools, anonymous communications, previous solutions to this problem, distributed input Pseudo-Random Functions (PRFs) and strongly robust Identity Based Encryption (IBE).
2. Interview experts in sexual assault accusations and possible users of the protocol in order to identify and define the protocol's requirements (both functional and in terms of efficiency).

[1]In the original proposal, we proposed proving these under the UC Security[20] model, however, we decided to make the security analysis as a sequence of games because the original proof for the WhoToo protocol was under this model.

3. Using the information from previous solutions together with everything learned in the previous step, define the protocol's characteristics, specifying which information can be revealed under which circumstances. For example, if there are multiple accusations in a specific department of the university, should this fact be revealed even if the quorum has not been reached? Then, the security, privacy and efficiency properties the protocol aims to achieve must be precised.
4. Identify inconsistencies and gaps in the WhoToo protocol description and fix them, providing the necessary security proofs for the modified protocol.
5. Study and propose a distributed version of Strongly Robust IBE in order to replace the accusation matching process of WhoToo with a more efficient alternative, while preserving its security guarantees.
6. Prove that the new protocol achieves secrecy, anonymity, accountability and MDH.
7. Prove that the new protocol is more efficient than WhoToo.
8. Design and implement a functional prototype.

1.6 Contributions

This thesis presents the following results:

1. We identify an inconsistency in a key element of WhoToo used to evaluate whether the quorum has been reached. A straightforward modification fixes the issue, yet it requires a new security proof which we provide.
2. We identify another operation of WhoToo whose specification leaves an implementation gap. In this case, the solution is not as simple, as a straightforward clarification would compromise either the anonymity of the accuser or the anonymity of the accused. Instead of concocting a local fix for the issue, we take a step back and re-examine the particular phase where it arises. Our solution then solves this issue while carrying a positive side effect: an improved efficiency.
3. Inspired by SAE [5], we propose an alternative to the duplicate revision process requiring only a constant number of (interactive distributed) operations, by employing a distributed input PRF (DIPRF) in a new way.
4. By relying on a different technique – a variant of strongly robust identity based encryption (IBE) – we reduce the number of interactive distributed operations required for the accusations matching process from $O(N)$ to a constant number too. This technique has applications to other server-based privacy-preserving protocols and may be of independent interest.
5. We interview experts and conduct focus groups with potential users, which leads to a valuable discussion about the protocol's requirements, which we present.
6. We provide security proofs for our modified protocol $\texttt{WhoToo}^{+}$.
7. We design and implement a functional prototype.

1.7 Related Work

In this section we briefly describe three previous protocols that have been proposed as solutions to the sexual assault reporting problem.

1.7.1 Callisto

The first protocol specifically designed for privacy-preserving sexual assault accusations was Callisto [56]. Its design criteria was explicitly motivated by the fact that *"those who experience unwanted sexual contact may be more willing to report it if they know that others have spoken up as well"* [56] attempting to privately preserve the names of accusers and accused so they can be compared with those of new reports. Using cryptographic tools, Callisto hides the identity of the accuser and the accused until a second accusation is made against the same perpetrator. Using an Oblivous Pseudo-Random Function, the protocol computes a pseudo-random value from the accused person's identifier, and then it stores the value in a database. Finding reports against the same perpetrator simply amounts to searching for repeated values. The accuser's information is also encrypted with a key that is secret shared [59] so that it can only be decrypted if there are at least 2 accusations against the same person.

Kuykendall, Krawczyk, and Rabin [43] described three attacks against Callisto, showing that there is no binding between the accuser's identity and the accusation, and that the identities of neither the accusers and the accused are entirely protected.

Furthermore, the OPRF server learns the accuser's identity, therefore, the security of the protocol is completely compromised if that server is corrupted[5].

1.7.2 Who Too

Following Callisto's principles, Kuykendall, Krawczyk and Rabin [43] propose WhoToo, a distributed protocol that provides stronger binding between the accuser's identity and their accusation, based on stronger security definitions that protect the identities of the accusers and accused. Furthermore, WhoToo works for any fixed quorum q, in contrast to Callisto where the quorum was fixed to 2. Below, we present a brief summary of WhoToo. A more complete picture is described in the next sections as WhoToo is indeed the base of our proposed protocol.

The protocol relies on threshold cryptography, so $t+1$ out of n servers need to agree in order to reveal any information, for some fixed $1 \leq t \leq n$. Accusations are stored in privacy-preserving multisets, represented as (encrypted) polynomials whose roots are the accused people's identifiers. Therefore, in order to add an accusation, it suffices that the servers multiply the existing set (polynomial) by $(x-s)$, where s represents the perpetrator's identifier. Checking if the quorum has been reached is simply done by checking if the $(q-1)^{th}$-derivative of the polynomial evaluated at s is zero. Multiset confidentiality and robustness follows from the servers computing over encrypted polynomials using verifiable secret sharing.

WhoToo seeks to protect all identities and avoid leaking information while at most t servers are corrupt. The protocol relies on a clever combination of ElGamal encryption [29], verifiable secret sharing [54] and group signatures [43] based on signatures of knowledge [17].

Despite the strong foundations of WhoToo, our analysis put forward some shortcomings (which can be avoided, as explained in this work later). First, WhoToo includes two small but potentially serious gaps between theory and implementation, which we deal with in Sect. 3. A more serious issue is WhoToo (in)efficiency under a backlog of N unopened accusations, as the process of checking for duplicates and finding matching accusations once the quorum has been reached becomes increasingly inefficient, requiring $O(N)$ interactive distributed operations.

1.7.3 Secure Allegations Escrow (SAE)

Secure Allegation Escrows (SAE) [5] is a distributed protocol for anonymous allegations that is based in the same quorum principles as Callisto and WhoToo. It provides accusation confidentiality, accuser anonymity, accountability and scalability as long as there is an honest majority. Adding a new accusations requires $O(1)$ distributed operations. Arun, Kate and Garg [5] introduce a Distributed input Verifiable pseudo-Random Function (DVRF), which is computed over a distributed input and key. They describe a bucketing algorithm for accusation matching, where each server locally checks if there are enough repeated DVRF values. Even though this achieves a matching algorithm that needs no distributed operations, it comes at a price: servers get to know if there are repeated values before the quorum is reached. This information leak could be used for potential attacks by a malicious user and an honest but curious server. A corrupted user, for example, could make a false accusation against a specific identifier and an honest but curious escrow server could recover the number of accusations (with equal or lower quorum) made against that identifier.

1.8 Structure

The rest of this thesis is organized as follows. First, in Chapter 2, we present the cryptographic tools needed to understand both the original WhoToo protocol [43] and our extension. In Chapter 3 we present an introduction to the WhoToo protocol and an overview of `WhoToo`$^+$. Then, we identify and correct two operations that where left unclear in the WhoToo protocol, adding a new security proof for the modified protocol. Afterwards, in Chapter 4, we present the optimizations that `WhoToo`$^+$ provides, introducing the new schemes and describing the necessary modifications. In Chapter 5 we discuss variants and extensions that arise from the interviews with experts and focus groups. In Chapter 6, we present the full description of the `WhoToo`$^+$protocol, followed by the security analysis. Finally, in Chapter 7 we describe the prototype design and an analysis on the theoretical and experimental efficiency of `WhoToo`$^+$. Chapter 8 presents concluding remarks and future work.

Part of this work was published in the paper "Implementing Secure Reporting of Sexual Misconduct - Revisiting WhoToo" in LATINCRYPT 2021 [37], including reduced contents of Chapters 1, 2, 3, 4, 6, 7 and 8.

Chapter 2

Cryptographic Tools

In this chapter we present the main concepts and tools needed to understand the WhoToo$^+$ protocol.

2.1 Preliminaries

This section presents some preliminaries needed to understand WhoToo and introduce WhoToo$^+$. Both protocols rely on a distributed authority ($\mathcal{DA}$) formed by n servers. In order to compute or reveal any private information, $t+1$ server must agree or cooperate.

2.1.1 Notation

In what follows, we write $[n]$ to denote $\{1, \ldots, n\}$ for $n \in \mathbb{Z}$. If S is a set, $x \xleftarrow{R} S$ denotes picking x uniformly at random from the elements in S. If V is an algorithm, running V and assigning its output to variable a is denoted by $a \leftarrow V$. When describing distributed protocols which share values among the players or servers, we use the notation $\{x_\mathrm{i}\}$ or simply $\{x\}$ to denote that x is a shared value among servers and x_i to denote the share of x corresponding to the i-th server.

2.1.2 Probabilistic Polynomial Time (PPT) Algorithm

In this work, we say the adversary is a Probabilistic Polynomial Time (PPT) algorithm. This means that we represent our adversary as probabilistic algorithm that runs in polynomial time in the security parameter λ. This security parameter is chosen during initialization and is known by the adversary [41].

We say an algorithm has oracle access if it can query an oracle, which is another machine that provides only the result of the query and no additional information. Each query is considered as one computational step [57].

2.1.3 Bilinear pairings

Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be groups for the operator $\cdot$. We say $\mathrm{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear pairing if it satisfies the following:

1. e is bilinear, this is, $\mathrm{e}(g \cdot g', h) = \mathrm{e}(g, h) \cdot \mathrm{e}(g', h)$ and $\mathrm{e}(h, g \cdot g') = \mathrm{e}(h, g) \cdot \mathrm{e}(h, g')$.
2. e is non-degenerate, this is, $\exists g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$ with $g_1 \neq 1, g_2 \neq 1$ where $\mathrm{e}(g_1, g_2) = 1$
3. There exists a probabilistic polynomial time (PPT) algorithm to compute e.

Furthermore, we classify bilinear pairings in three types:

1. Type-1: $\mathbb{G}_1 = \mathbb{G}_2$
2. Type-2: $\mathbb{G}_1 \neq \mathbb{G}_2$ and there is a known isomorphism $\varphi : \mathbb{G}_1 \to \mathbb{G}_2$ that is efficiently computable.
3. Type-3: $\mathbb{G}_1 \neq \mathbb{G}_2$ and there is no known isomorphism $\varphi : \mathbb{G}_1 \to \mathbb{G}_2$ that is efficiently computable.

In what follows, let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, g_1, g_2)$ be a Type-3 bilinear pairing with prime order p, where g_1 and g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. We suggest using the curves BLS12-381 [18] or BN254 [8] following the original WhoToo protocol [43].

2.1.4 Computational Assumptions

The protocols presented in this work rely on a variety of standard Diffie-Hellman assumptions which are described in this section.

Decisional Diffie-Hellman (DDH): Set $a \xleftarrow{R} \mathbb{Z}_p$ and $g, h, h' \xleftarrow{R} \mathbb{G}_1$. The distributions of (g, h, g^a, h^a) and (g, h, g^a, h') are computationally indistinguishable for any PPT algorithm.

Strong computational Diffie-Hellman (sCDH): Set $g, h \xleftarrow{R} \mathbb{G}_1$ and $a \xleftarrow{R} \mathbb{Z}_p$. Given g^a, the probability of computing h^a is negligible for any PPT oracle algorithm [1].

k-strong Diffie-Hellman (sDH_k): Fix $k \in \mathbb{Z}$. Set $g_2 \in \mathbb{G}_2$, $g_1 \leftarrow \psi(g_2)$ and $\gamma \in \mathbb{Z}_p^*$ where ψ is an isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$. Given $g_1, g_2^{\gamma}, ..., g_2^{\gamma^k}$, the probability of finding a pair $(x, g_1^{1/(\gamma+x)})$ is negligible for any PPT oracle algorithm [16].

q-Decisional Bilinear Diffie Hellman Inversion ($q-$DBDHI) Set $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$ and $x \xleftarrow{R} \mathbb{Z}_p^*$. Given $(g_1, g_1^x, ..., g_1^{x^q}, g_2, g_2^x, ..., g_2^{x^q})$, the probability of computing $\mathrm{e}(g_1, g_2)^{1/x}$ is negligible for any PPT algorithm [15].

Truncated decisional augmented bilinear Diffie-Hellman exponent (q-ABDHE) over Type-3 Pairings: Set $g_1, g_1' \xleftarrow{R} \mathbb{G}_1$ and $g_2, g_2' \xleftarrow{R} \mathbb{G}_2$, $\alpha \xleftarrow{R} \mathbb{Z}_p$ and $Z \xleftarrow{R} \mathbb{G}_T$. The distributions of $(g_1', g_2', g_1'^{\alpha^{q+2}}, g_2'^{\alpha^{q+2}}, g_1, g_1^{\alpha}, ..., g_1^{\alpha^q}, g_2, g_2^{\alpha}, ..., g_2^{\alpha^q}, \mathrm{e}(g_1^{\alpha^{q+1}}, g_2'))$ and $((g_1', g_2', g_1'^{\alpha^{q+2}}, g_2'^{\alpha^{q+2}}, g_1, g_1^{\alpha}, ..., g_1^{\alpha^q}, g_2, g_2^{\alpha}, ..., g_2^{\alpha^q}, Z)$ are are computationally indistinguishable for any PPT algorithm [53].

The original WhoToo protocol is defined over Type-2 Pairings. In order to guarantee that the security properties hold with Type-3 pairings, we assume adversaries with oracle access to $\varphi : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ such that $\varphi(g_1) = g_2$ still have negligible advantage [43].

2.1.5 Communication channels

For this protocol, we require three characterizations for communication channels. Note that these characterizations are not exclusive.

Lets say a channel allows communication from sender A to receiver B.

1. *Authenticated channel:* The input of this channel can only be accessed by a specific sender A, which is known by the receiver B [49]. The adversary can only read, forward or delete messages from A [25].
2. *Confidential channel*: The output of this channel can only be accessed by a specific receiver B, which is known by the sender A [49]. The adversary only learns the length of the messages and can inject new messages [25].
3. *Anonymous channel*: Formal definitions for different types of anonymous channels can be found in the work by Hevia and Micccianco [38]. For this protocol, we need Strong Sender Anonymity, where the identity of the sender is protected and all information about the number and value of the sent messages is hidden. The only information leaked to the adversary is the number of messages received by B.

2.1.6 Security Requirements and Threat Model

Security requirements: As mentioned in the previous chapter, `WhoToo`$^+$guarantees secrecy, anonymity, accountability and meta-data hiding.

Network model: Assume all communication between servers and users during initialization occurs over an *authenticated* and *confidential* channel. This guarantees that users are given their corresponding keys and MACs, but they are hidden from everyone else. Accusations are submitted over an *anonymous* and *confidential* channel, ensuring that the accuser's identity is protected and that only each corresponding server sees their share of the accusation.

Adversary: The adversary is a PPT algorithm, with respect to a chosen security parameter λ. It is interested in revealing the identities of the accusers or the accused, as well as leaking information about matching accusations, before the quorum is reached. It can control any number of users and at most t out of n servers. Assume a static model in which the corrupted users and servers are chosen and fixed before initialization.

Excluded attacks: Protecting against false allegations is outside the scope of this protocol. Nevertheless, the accountability property of `WhoToo`$^+$ helps to discourage this behavior. Furthermore, studies have shown that false sexual assault allegations are extremely rare [52].

2.2 Building Blocks

This section describes the components used in the $\texttt{WhoToo}^+$ construction. We follow the original WhoToo presentation.

2.2.1 Pseudo-Random Functions:

A pseudo-random function (PRF) is a deterministic function which, given a key sk and input x, returns a value that is indistinguishable from a random value for anyone who does not know the secret key sk [41].

We provide the formal definition exactly as presented by Katz and Lindell [41].

Definition 1 *(Pseudo-Random Function)* Let $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be an efficient, length-preserving, keyed function. F is a pseudo-random function if for all probabilistic polynomial-time distinguishers D, there is a negligible function negl such that:

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

where the first probability is taken over uniform choice of $k \in \{0,1\}^n$ and the randomness of D, and the second probability is taken over uniform choice of $f \in \mathsf{Func}_n$ and the randomness of D.

2.2.2 Hash Function

A hash function $H : A \to B$ is a function that compresses its input to a fixed length $|B|$. We say a hash function is secure if it is collision resistant, this is, if the probability of finding two distinct inputs with the same output is negligible [41].

We now present the formal definitions of hash functions, the collision finding experiment and collision resistant hash functions exactly as presented in [41].

Definition 2 *(Hash Function)* A hash function (with output length ℓ) is a pair of probabilistic polynomial-time algorithms (Gen, H) satisfying the following:

- Gen is a probabilistic algorithm which takes as input a security parameter 1^n and outputs a key s. We assume that 1^n is implicit in s.
- H takes as input a key s and a string $x \in \{0,1\}^*$ and outputs a string $H^s(x) \in \{0,1\}^{\ell(n)}$ (where n is the value of the security parameter implicit in s).

Definition 3 *(The collision finding experiment $\mathsf{Hash\text{-}coll}_{\mathcal{A},\Pi}(n)$)*

1. A key s is generated by running $Gen(1^n)$.
2. The adversary $\mathcal{A}$ is given s and outputs $x, x' \in \{0,1\}^{\ell(n)}$.
3. The output of the experiment is defined to be 1 if and only if $x \neq x'$ and $H^s(x) = H^s(x')$. In such case we sat that $\mathcal{A}$ has found a collision.

Definition 4 *(Collision Resistant Hash Function)* A hash function $\Pi = (Gen, H)$ is collision resistant if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there is a negligible function negl such that:

$$|\Pr[\,\mathsf{Hash\text{-}coll}_{\mathcal{A},\Pi}(n) = 1\,] \leq \mathsf{negl}(n).$$

We will regularly use hash functions with codomain $B = \mathbb{Z}_p$. In this work, we treat all hash functions as random oracles (as a black box that outputs uniformly random values in B) for the security proofs.

2.2.3 Message Authentication Code (MAC)

A MAC is a code t that is sent together with a message m so that the receiver can verify that m was not modified in transit. If A wants to send m to B, they share a key k and A computes a tag $t \leftarrow MAC_k(m)$ and sends (m, t). When B receives (m, t) it can use the key k to verify that t is a valid tag. This is done with a verification algorithm that receives k and t and outputs a boolean indicating if the tag was correctly verified or not [41].

We present formal definitions based on [11] and [41].

Definition 5 (MAC) A MAC is a pair of polynomial algorithms (Gen, Tag, Ver) satisfying the following:

- Gen is a probabilistic algorithm which takes as input a security parameter 1^n and outputs a key k with $|k| \geq n$
- Tag can be either a probabilistic or deterministic algorithm. It takes as input a key k and a string $m \in \{0,1\}^*$ and produces a tag $t \leftarrow Tag_k(m)$.
- Ver is a deterministic algorithm that takes as input a key k, string $m \in \{0,1\}^*$ and tag $t \in \{0,1\}^*$ and outputs 1 if and only if $Tag_k(m) = t$

The following experiment captures the property of strong unforegability under chosen message attack, which guarantees that an adversary has negligible probability of generating a valid tag t for a message m without knowing the key k.

Definition 6 *(Strong Unforgeability under Chosen Message Attack Experiment $\mathsf{SUF\text{-}CMA}_{\mathcal{A},MAC}$)* Consider a $MAC = (Gen, Tag, Ver)$, adversary $\mathcal{A}$, and value n for the security parameter.

1. A key k is generated by running $Gen(1^n)$.
2. The adversary $\mathcal{A}$ is given input 1^n and oracle access to $Tag_k(\cdot)$ and $Ver_k(\cdot,\cdot)$.
3. The output of the experiment is defined to be 1 if and only if $\mathcal{A}$ made Ver_k query (m, t) such that the oracle returned 1 and $\mathcal{A}$ did not make the query m to the Tag_k oracle.

Definition 7 *(SUF-CMA Secure MAC)* A $MAC = (Tag, Ver)$ is SUF-CMA secure if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there is a neglegible function negl such that:

$$\Pr[\,\mathsf{SUF\text{-}CMA}_{\mathcal{A},MAC} = 1\,] \leq \mathsf{negl}(n).$$

2.2.4 ElGamal Encryption

ElGamal encryption is a public key encryption scheme [29]. This means there is a public key $pk = (g, h)$ that can be used to encrypt any message and a secret key $sk = x$ needed to decrypt. This scheme is multiplicatively homomorphic, meaning that the product of two ciphertexts is the encryption of the product of the corresponding plaintexts.

WhoToo and $\texttt{WhoToo}^+$ use three variants of ElGamal encryption. The first one is the (standard) multiplicatively homomorphic ElGamal, which allows encryption of group elements, using the following primitives:

$\mathsf{ElGamal.Setup}$: Choose $x \xleftarrow{R} \mathbb{Z}_p$ and $h \xleftarrow{R} \mathbb{G}_1$, and set $g \leftarrow h^{1/x}$. Output $sk \leftarrow x$ and $pk \leftarrow (g, h)$.

$\mathsf{ElGamal.Enc}(pk, m)$: Choose $a \xleftarrow{R} \mathbb{Z}_p$ and output $(c_1, c_2) \leftarrow (g^a, h^a m)$.

$\mathsf{ElGamal.Dec}(sk, (c_1, c_2))$: Output c_2/c_1^x.

The second one is an extension of ElGamal to encrypt strings [1] which uses a symmetric key authenticated encryption scheme[1] $(\mathsf{AuthEnc}, \mathsf{AuthDec})$ with keyspace $\mathcal{K}$ and a hash function $H_{\mathcal{K}}$ with codomain $\mathcal{K}$:

$\mathsf{ElGamal.EncString}(pk, m)$: Choose $a \xleftarrow{R} \mathbb{Z}_p$ and output $(c_1, c_2) \leftarrow (g^a, \mathsf{AuthEnc}(H_{\mathcal{K}}(h^a), m))$.

$\mathsf{ElGamal.DecString}(sk, (c_1, c_2))$: Output $\mathsf{AuthDec}(H_{\mathcal{K}}(c_1^x), c_2)$.

Finally, in our setting, the secret key $sk = \{x\}$ for the ElGamal scheme must be secret shared (we explain what this means in the following section) among the servers, so we use a variant that adds distributed decryption to the previous ElGamal extension:

$\mathsf{ElGamal.DistDec}(c_1, c_2, \{x\})$: Compute $\mathrm{d} \leftarrow \mathsf{SecShare.Exp}(c_1, \{x\})$ and output c_2/d. Here $\mathsf{SecShare.Exp}$ is the distributed exponentiation protocol described later in the following sections.

$\mathsf{ElGamal.DistDecString}(c_1, c_2, \{x\})$: Compute $\mathrm{d} \leftarrow \mathsf{SecShare.Exp}(c_1, \{x\})$ and output $\mathsf{AuthDec}(H(\mathrm{d}), c_2)$.

In order to understand the security achieved by these schemes, we introduce definitions for indistinguishability under chosen plaintext attack (IND-CPA) and indistinguishability under chosen ciphertext attack (IND-CCA), as presented in [41] verbatim.

Definition 8 *(CPA Indistinguishability Experiment $\mathsf{IND\text{-}CPA}_{\mathcal{A},\Pi}$)* Consider an encryption scheme Gen, Enc, Dec, adversary $\mathcal{A}$, and value n for the security parameter.

[1] A symmetric key encryption scheme relies on a unique key k shared by the sender and reciever which is used to encypt and decrypt. An authenticated encryption scheme guarantees authenticity (the message actually comes from the sender) and integrity (the message was not modified during transit).

1. A key k is generated by running $Gen(1^n)$.
2. The adversary $\mathcal{A}$ is given input 1^n and oracle access to $Enc_k(\cdot)$, and outputs a pair of messages m_0, m_1 of the same length.
3. A uniform bit $b \in \{0,1\}$ is chosen, and then a ciphertext $c \leftarrow Enc_k(m_b)$ is computed and given to $\mathcal{A}$.
4. The adversary $\mathcal{A}$ continues to have oracle access to $Enc_k(\cdot)$, and outputs b'.
5. The output of the experiment is defined to be 1 if $b = b'$ and 0 otherwise.

Definition 9 *(IND-CPA Secure Scheme)* An encryption scheme $\Pi = (Gen, Enc, Dec)$ is IND-CPA secure if for all probabilistic polynomial-time adversaries $\mathcal{A}$ the is a negligible function negl such that:

$$\Pr[\,\mathsf{IND\text{-}CPA}_{\mathcal{A},\Pi} = 1\,] \leq \frac{1}{2} + \mathsf{negl}(n),$$

where the probability is taken over the randomness used by $\mathcal{A}$, as well as the randomness used in the experiment.

Definition 10 *(CCA Indistinguishability Experiment $\mathsf{IND\text{-}CCA}_{\mathcal{A},\Pi}$)* Consider an encryption scheme Gen, Enc, Dec, adversary $\mathcal{A}$, and value n for the security parameter.

1. A key k is generated by running $Gen(1^n)$.
2. The adversary $\mathcal{A}$ is given input 1^n and oracle access to $Enc_k(\cdot)$ and $Dec_k(\cdot)$. It outputs a pair of messages m_0, m_1 of the same length.
3. A uniform bit $b \in \{0,1\}$ is chosen, and then a ciphertext $c \leftarrow Enc_k(m_b)$ is computed and given to $\mathcal{A}$. We call c the challenge ciphertext.
4. The adversary $\mathcal{A}$ continues to have oracle access to $Enc_k(\cdot)$ and $Dec_k(\cdot)$, but is not allowed to query the latter on the challenge ciphertext itself. Eventually, $\mathcal{A}$ outputs a bit b'.
5. The output of the experiment is defined to be 1 if $b = b'$ and 0 otherwise.

Definition 11 *(IND-CCA Secure Scheme)* An encryption scheme $\Pi = (Gen, Enc, Dec)$ is IND-CCA secure if for all probabilistic polynomial-time adversaries $\mathcal{A}$ the is a negligible function negl such that:

$$\Pr[\,\mathsf{IND\text{-}CCA}_{\mathcal{A},\Pi} = 1\,] \leq \frac{1}{2} + \mathsf{negl}(n),$$

where the probability is taken over the randomness used by $\mathcal{A}$, as well as the randomness used in the experiment.

Intuitively, IND-CPA means that it is very difficult to determine if a ciphertext c^* is an encryption of m_0 or m_1. IND-CCA is a stronger guarantee as the adversary is also given

access to a decryption oracle, which means it can decrypt any $c \neq c^*$ in order to determine which plaintext is encoded in c^*.

The first and third ElGamal variants are CPA-secure under the Decisional Diffie-Hellman assumption. The extension to strings is CCA-secure under the strong computational Diffie-Hellman assumption if $H_{\mathcal{K}}$ is a random oracle [1].

2.2.5 Zero-knowledge proofs

A zero-knowledge proof is a way to prove a statement without revealing any additional information [47]. We say a prover P wants to prove to a verifier V that statement x is true. In order to convince V, P must prove that it knows the witness w to the statement x, namely, it must output w such that $(x, w) \in R$. Formally, we define a knowledge extractor K and say that "a prover P^* knows the witness to a statement x if $K^{P^*(\cdot)}(x)$ outputs w s.t $(x, w) \in R$ whenever P^* convinces V of x" [45].

Zero knowledge proofs guarantee completeness, knowledge soundness and zero-knowledge(ness). Completeness guarantees that an honest prover will always be able to produce a proof that is accepted by an honest verifier. Knowledge soundness means that any adversary has negligible probability of producing a valid proof for a false statement, namely, it guarantees that the prover knows the witness. The following is the formal definition of knowledge soundness by Lindell [45] verbatim.

Definition 12 *(Knowledge Soundness)* A proof system has knowledge soundness with error κ if there exists a PPT K s.t for every prover P^*, if P^* convinces V of x with probability $\varepsilon > \kappa$, then $K^{P^*(\cdot)}(x)$ outputs w s.t $(x, w) \in R$ with probability at least $\varepsilon(|x|) - \kappa(|x|)$.

Therefore, if a protocol has knowledge soundness, we can always extract the witness of a valid proof.

Finally, zero-knowledge ensures that the verifier does not learn anything other than whether the proof is accepted or not [6, 45]. Zero-knowledge guarantees that there exists a simulator $\mathcal{S}$ that can produce proofs for true statements without knowing the witness, such that the view is indistinguishable from the real protocol [36].

In this protocol, we will use zero-knowledge proofs to prove that c is an ElGamal encryption of plaintext m with randomness a, without revealing anything about m or a.

We require proofs of plaintext knowledge to detect when users submit malformed accusations [43]. These proofs are standard, based on the Schnorr's protocol [58], applying the Fiat-Shamir heuristic [31] to make it non-interactive.

$\mathsf{ElGamal.Prove}(c, a, \rho)$: Given c, en ElGamal encryption of m with randomness a, output π, a non interactive proof of knowledge of m. Here ρ is the value used to derive the random oracle challenge.

$\mathsf{ElGamal.Verify}(pk, \pi, c, \rho)$: Check if π is a valid zero-knowledge proof for the plaintext

encoded in c.

In this case, the witness is the plaintext m encoded in c.

2.2.6 Threshold Operations

Threshold operations are those which are computed by a group of n servers and require at least $t+1$ servers to agree or cooperate. Both WhoToo and $\texttt{WhoToo}^+$ strongly rely on threshold cryptography, and in particular, on the following threshold operations. They require a group $\mathbb{G}$ with generators g and h. In $\texttt{WhoToo}^+$, $\mathbb{G} = \mathbb{G}_1$ and $g = g_1$ unless stated otherwise.

Verifiable Secret Sharing (VSS): All threshold operations are based on VSS which requires Shamir Secret Sharing [59]. In Shamir's protocol, in order to share a value $x \in \mathbb{Z}_p$, a random polynomial P of degree $t \leq n$ is chosen such that $P(0) = x$, and then shares $x_\mathrm{i} = P(\mathrm{i})$ are sent to the i-th server. This way, a collusion of t or less servers cannot recover x, but any greater number of servers can recover the secret by pooling their information. Recovery is done simply by publishing the x_i's and using Lagrange interpolation. Pedersen VSS [54] introduced share verification by asking the party that wants to share x to compute both the sharing of x and the sharing of a random value $r \in \mathbb{Z}_p$. At the same time, it must compute verification values $v_\mathrm{i} \leftarrow g^{a_\mathrm{i}} h^{b_\mathrm{i}}$, for $\mathrm{i} = 0, ..., t$, where a_i and b_i are the coefficients of the polynomials used for secret sharing x and r respectively. Observe that $x = a_0$ and $r = b_0$. When each server $\mathcal{DA}_\mathrm{i}$ receives its shares, it verifies them by checking if $g^{x_\mathrm{i}} h^{r_\mathrm{i}} = \prod_{j=0}^{t} v_j^{\mathrm{i}^j}$ were x_i and r_i are the shares of $\mathcal{DA}_\mathrm{i}$ for x and r respectively. We use the convention that if the operation outputs shares, they are output locally to the servers themselves (each server obtaining their own single share), as opposed to the case it outputs values, which are publicly revealed. This functionality is captured in the following three operations:

$\mathsf{SecShare.Encode}(x)$: output[2] $\{\omega\} = (\{x\}, \{r\})$, v, e_0 and r, where r is the random value used for Pedersen VSS, v are the Pedersen verification values and $\mathrm{e}_0 \leftarrow g^r$.

$\mathsf{SecShare.Verify}(w_\mathrm{i}, v)$: Compute Pedersen verification, as described above. Output False if verification fails and True otherwise [24].

$\mathsf{SecShare.Reconstruct}(\{x\})$: Reconstruct the secret x from its shares.

Many other operations can be build on top of Pedersen VSS such as reconstructing g^x without reconstructing x, or operating on the shares to compute shares for a given function of x.

We use several of them, enumerated below:

$\mathsf{SecShare.Encode}(x)$: output $\{\omega\} = (\{x\}, \{r\})$, v, e_0 and r, where r is the random value used for Pedersen VSS, v are the Pedersen verification values and $\mathrm{e}_0 \leftarrow g^r$.

$\mathsf{SecShare.Verify}(w_\mathrm{i}, v)$: Compute Pedersen verification, as described above. Output false if verification fails and true otherwise [24].

[2]For this work, we slightly modify the semantics for the function $\mathsf{SecShare.Encode}(x)$ so all shares $\{w\}$ are received by the party who invokes the function.

SecShare.Reconstruct($\{x\}$): Reconstruct the secret x from its shares.

SecShare.Verify(w_i, v): Compute Pedersen verification, as described above. Output False if verification fails and True otherwise [24].

SecShare.Reconstruct($\{x\}$): Reconstruct the secret x from its shares.

SecShare.Gen(): Generates a shared secret $\{x\}$ in a distributed manner so $x \in_R \mathbb{Z}_p$ is unknown to t servers or less [33].

SecShare.Add($\{x\}, \{y\}$): Output $\{z\}$ where $z = x + y$.

SecShare.Mult($\{x\}, \{y\}$): Output $\{z\}$ where $z = x \cdot y$ [13, 34].

SecShare.Invert($\{x\}$): Outputs $\{x^{-1}\}$ [7].

SecShare.Exp(b,$\{x\}$): Reconstruct b^x from the shares $\{x\}$ directly by releasing b^{x_i} and performing interpolation on the exponent, namely computing $b^x = \prod (b^{x_\mathrm{i}})^{\lambda_\mathrm{i}}$, for Lagrange coefficients λ_i [43].

SecShare.ExpLocal($b, \{x\}$, U): Output the pair $(b^{x_\mathrm{i}}, \pi_\mathrm{i})$ privately to user U, where π_i and a zero-knowledge proof of equality of discrete logarithm for b^{x_i} and g^{x_i}. This operation is called `PublicExponenciate` in [5].

SecShare.CheckConsistent($\{\omega\}, \mathrm{e}_0$): Extract $(\{s\}, \{r\}) \leftarrow \{\omega\}$ and output True if $\mathrm{e}_0 =$ SecShare.Exp(g, {r}) and False otherwise [43].

For conciseness' sake, we sometimes use a simpler notation, where secret share addition, multiplication, share inversion, and exponentiation are written in infix notation. For example, given shares $\{x\}$ and $\{y\}$, their addition is denoted by $\{x\} + \{y\}$, their multiplication by $\{x\} \cdot \{y\}$, inverting $\{x\}$ by $\{x\}^{-1}$, and exponentiation of $b \in G$ by $b^{\{x\}}$. By $\{x\} \xleftarrow{R} G$ we will also denote running SecShare.Gen(), and share reconstruction by $x \leftarrow \{x\}$.

The following operations can be easily implemented from the previous operations, but we name them to make the presentation easier:

SecShare.Gen(g): Given a generator g of $\mathbb{G}$, generates a shared secret $\{x\}$ in a distributed manner for $x \in_R \mathbb{Z}_p$, and outputs a public value $g^x \in \mathbb{G}$ [33].

SecShare.GenInv(g): Given a generator g of $\mathbb{G}$, generate a random shared secret $\{x\}$ in a distributed manner and output a public value $g^{1/x}$ [32]. This is equivalent to computing $g^{\{y\}}$ where $\{x\} \xleftarrow{R} G$, $\{y\} \leftarrow \{x\}^{-1}$.

SecShare.ExpRR($(\mathrm{e}_0, \mathrm{e}_1), \{x\}$): Given $(\mathrm{e}_0, \mathrm{e}_1)$ an ElGamal encryption of g^m, output an encryption of g^{mx} [43].

SecShare.MultExp($b_1, b_2, \{x\}, \{y\}$): Reconstructs $b_1^x \cdot b_2^y$ without exposing b_1^x nor b_2^y. The procedure is new, needed for `WhoToo`$^+$. It is described in Figure 2.1. We use the simpler notation $\{z\} \leftarrow b_1^{\{x\}} \cdot b_2^{\{y\}}$.

SecShare.MultExp$(b_1, b_2, \{\alpha\}, \{\beta\})$:

1. $\{t\} \leftarrow \mathbb{Z}_p$
2. $\{t^{-1}\} \leftarrow \{t\}^{-1}$
3. $\{\alpha'\} \leftarrow \{\alpha\} \cdot \{t\}$
4. $\{\beta'\} \leftarrow \{\beta\} \cdot \{t\}$
5. $b_1^{\alpha'} \leftarrow b_1^{\{\alpha'\}}$
6. $b_2^{\beta'} \leftarrow b_2^{\{\beta'\}}$
7. $b_1^{\alpha} b_2^{\beta} \leftarrow (b_1^{\alpha'} b_2^{\beta'})^{\{t^{-1}\}}$
8. Output $b_1^{\alpha} b_2^{\beta}$

Figure 2.1: Multiplication with shared exponents

All of these operations are secure multiparty computations and therefore the view of the honest servers can be computationally simulated using only public outputs (the verification values), a fact we use in the security proofs. Indeed, with the exception of SecShare.ExpLocal, the operations that a shared value $\{x\}$ and produce (the shares of) a new shared value $\{y\}$ (e.g. $\{y^{-1}\}$ = SecShare.Invert$(\{x\})$) mentioned above, they also publicly output new verification values. To keep things simple, we do not include these values in the notation. These operations also provide *correctness*, meaning that the reconstruction does actually recover the secret shared value.

2.2.7 Distributed Group Signatures

A signature scheme allows a party A to sign a message with their secret key sk, so that anyone who knows A's public key pk can verify the signature. A group signature scheme allows any member of set of participants to sign a message on behalf of a group without disclosing their identity unless a special participant, the group manager, wishes to trace and expose the signer. In [43] building on [14], Kuykendall et al. present a distributed variant of the Boneh, Boyen, Shacham group signature [17] where the manager is distributed among the servers. This scheme is used to validate an accuser's identity and reveal it once their accusation has reached the quorum. BBS signatures provide a signature of knowledge of a private key α together with an ElGamal Encryption of R. To prevent disclosure of the identity of the signer, the private key to decrypt the second value is distributed among the servers.

The distributed operations we need for both WhoToo and `WhoToo`$^+$ are the following:

DistBBS.Setup(g_1, g_2): The servers compute secret keys $\{x\}$ and $\{\gamma\}$, $h \leftarrow g^x$, $w \leftarrow g_2^{\gamma}$, and publish public keys $pkeg \leftarrow (g_1, h, w)$.

DistBBS.UserKeyIssue$_U(\{\gamma\})$: The servers compute the private key $\{\alpha\}$ for user U and send them their shares so that U can reconstruct α. They also compute the public key $R \leftarrow g_1^{1/(\alpha+\gamma)}$.

BBS.Sign(pk, sk_U, m): Compute $c_R \leftarrow$ ElGamal.Enc$(pkeg, R)$ and σ, the signature of knowledge of sk_U. Output (c_r, σ).

BBS.Verify(pk, m, c, σ): Verify that σ is a valid signature of knowledge.

DistBBS.Trace($c_R, \{x\}$): Output ElGamal.DistDec($c_R, \{x\}$).

The implementation of these functions is detailed in the WhoToo paper [43].

Under the security notions defined by Bellare, Micciancio and Warinschi [10], this scheme is correct, fully anonymous and fully traceable under the k-strong Diffie Hellman assumption in $\mathbb{G}_1$, as long as the secret sharing operations are secure. Correctness guarantees that a correct signature by an honest user is accepted and the signer is correctly identified. Anonymity ensures that an adversary has negligible probability of distinguishing between the signatures of two different parties (before tracing them). Finally, traceability means that an adversary has negligible probability of producing either a valid signature whose origin can not be identified by an honest opener, or a signature whose origin can be identified yet can not be validated [12].

2.2.8 Privacy-Preserving Multisets

One of the innovative aspects of WhoToo was the efficient use of new privacy-preserving data structures. One of them is the multisets proposed by Kissner and Song [42], constructed from encoded polynomials. In this solution, sets are represented by a polynomial $F(x)$ and elements in multisets are represented as the roots of $F(x)$. Adding a new element s then is simply multiplying $F(x)$ by $(x - s)$, and checking if the quorum has been reached after adding a value s is done by calculating $F^{(q)}(x)$, the $(q-1)^{th}$-derivate of $F(x)$, and verifying if $F^{(q)}(s) = 0$. We follow the description and notation from [43], and use $\mathrm{e}_F = (\mathrm{e}_{F_0}, \mathrm{e}_{F_1}, ..., \mathrm{e}_{F_\mathrm{d}})$ to represent the encoded coefficients of polynomial $F(x) = F_0 + F_1 x + ... + F_\mathrm{d} x^\mathrm{d}$, where $\mathrm{e}_{F_\mathrm{i}}$ is an ElGamal encryption of g^{F_i}. Thus, given an ElGamal encryption scheme, any polynomial F can be represented by an encoded polynomial e_F.

Next, we describe the supported operations for polynomials we require, following [43].

PrivatePoly.Subtract($\mathrm{e}_F, \mathrm{e}_G$): Output e_H where $H = F - G$.

Private.Poly.Differentiate(e_F, n): Output e_G where G is the n^{th}-derivate of F.

PrivatePoly.Multiply(e_F, R): Given an encoded polynomial F and a polynomial R in the clear, output e_G where $G = F \cdot R$.

PrivatePoly.MultiplyLinear($\mathrm{e}_F, \{s\}$): Output e_G where $G = F \cdot (x - s)$.

PrivatePoly.ZeroTest($\mathrm{e}_F, \{s\}$): Output true if $F(s) = 0$, false otherwise.

For a detailed description of these operations, see [43]. Using these operations over polynomials, Kuykendall et al. define the following privacy-preserving multiset operations:

Set.Init(): Output e_{g^0}. This creates the set.

Set.Add($\mathrm{e}_F, \{s\}$): Output PrivatePoly.MultiplyLinear(e_F, s). Adding a value s to F is simply multiply F by $(x - s)$.

Set.Quorum($\mathrm{e}_F, \{s\}$): Output true if $(x - s)^q$ divides F and false otherwise. Value q is the

pre-specified quorum for the protocol.

In terms of security notions, we say the data structure is correct if all operations over the multiset (additions, and evaluating if any element has multiplicity q) properly correlates to the above operations over the polynomials. Also, the structure achieves (computational) hiding if any PPT adversary with limited interaction with the data structure (A gets to choose some elements to add, sees some of the other additions but does not get to see all added elements) does not obtain any information other than the size of the set and what it can infer from the multiplicity tests (see the work by Kissner and Song [42] for formal definitions). The WhoToo protocol [43] required that these operations achieve perfect completeness and computational hiding of the elements of the set. The only information revealed should be the size of the set, e.g, the degree of the polynomial.

Chapter 3

A First Review of WhoToo

In this chapter we present overviews of both the original WhoToo protocol and our new WhoToo$^+$protocol. We provide high level descriptions of each of its components, emphasizing the differences between them (Section 3.1). Then, we identify and fix two gaps found in the original protocol (Section 3.2).

3.1 Protocol Overviews

3.1.1 WhoToo: An Introduction

This section provides an informal description of the WhoToo Protocol [43]. At a high level, each accuser can submit an accusation against a certain person and the identities of the accuser and the accused remain hidden until the quorum is reached, namely, until a pre-specified number of accusers file accusations against the same person.

Participating parties: The protocol strongly relies on threshold cryptography, where in order to compute or reveal any information that could reveal private values, $t + 1$ of the n servers need to agree or cooperate. This group of servers is called the Distributed Authority ($\mathcal{DA}$). The cooperation guarantees that as long as no more than t servers are corrupted, all operations are performed correctly and no server learns private values.

Accusers can be any user U of the system which are identified with a public key R computed by the $\mathcal{DA}$ during registration. The identity of the accused is represented by an arbitrary string D: a name, e-mail or any unique identifier. For simplicity, we assume this value is unique but in practice the accuser can file different accusations for every identity under which the accused is known [43].

Registration: WhoToo assumes there exists an external registration authority which verifies the identities of the potential accusers. All registered users are valid accusers. We envision the system used in a community where all members may submit accusations.

For every user U in this list, the $\mathcal{DA}$ computes a public key R and registers that the key

R corresponds to user U. It also computes a private key α which is only obtained by U.

Submitting accusations: In order to submit an accusation against D from a user with identifier R, the user needs to provide encodings c_D and c_R of these values. These values are intrinsically linked to D and R respectively, yet reveal no useful information about D and R unless $t+1$ $\mathcal{DA}$ servers cooperate. Moreover, the $\mathcal{DA}$ needs to be able to make computations on the encoded values in order to know if there are a certain number of accusations against the same D without revealing anything about D.

To achieve this, two main tools are used. First, using verifiable secret sharing [54], the user distributes shares of D to every server so if $t+1$ or more of them cooperate they can recover the secret, yet t or less servers learn nothing about it. A linear secret sharing scheme allows efficient computation of some operations from the shared values without revealing D. Additionally, R and D are encoded using a threshold public key encryption scheme. Any user can encrypt using the public key while cooperation is needed in order to recover the plaintext as the secret key is distributed among the $\mathcal{DA}$ servers. Both of these schemes preserve anonymity of the accuser and the accused as long as at most t servers are corrupted.

Finally, the user also signs these encodings with their private key α, guaranteeing accountability. It also provides zero-knowledge proofs for the encoded values so that the $\mathcal{DA}$ can verify that there are no malformed accusations while preserving anonymity.

Discarding malformed and duplicated accusations: Once the $\mathcal{DA}$ receives an accusation from a user, it verifies the signature and zero-knowledge proofs, and discards any malformed accusations. Then, it must verify that it is not a duplicated accusation, namely, that there is not already an accusation from the same R to the same D in the system. This is non-trivial as the check must not reveal anything about R or D other than if both are equal to a previous accusation. In order to do this, WhoToo introduces a distributed equality testing described in Section 3.2.2 that verifies if two ciphertexts encode the same plaintext. They use this to compare the new accusation to every other c_R and c_D in the system.

Finding matching accusations: Accusations are stored in privacy-preserving multisets, represented as polynomials where the roots are the accused people's identifiers. In order to add an accusation, the servers multiply the existing set by $(x - D)$ and checking if the quorum has been reached is done by verifying if the $(q-1)^{th}$-derivative of the polynomial evaluated at D is zero.

The polynomial coefficients are encrypted using threshold public key encryption that is multiplicatively homomorphic. This allows to efficiently implement the required set operations (adding a new element and checking if the quorum has been reached) described above. The $\mathcal{DA}$ servers use their shares of D to multiply the existing encoded polynomial by $(x-D)$ when adding a new accusation.

Once the quorum is reached, the $\mathcal{DA}$ needs to identify the individual matching accusations. WhoToo uses the same equality testing used to discard duplicated accusations to compare c_D of the last submitted accusation to every other accusation in the system.

Once the matching accusations and its respective accusers are identified, this information is given to the corresponding authority. The way in which this information is used is beyond the scope of this protocol.

3.1.2 WhoToo$^+$ Overview

In WhoToo$^+$, we provide additional steps during registration and accusation submission, which together with significant changes during the verification, duplicate revision and finding matching accusations phases, allow us to correct some inconsistencies found in WhoToo and to improve efficiency. Consequently, WhoToo$^+$ achieves scalability, making it practical for real world implementation, even under a significant backlog of unopened accusations. An overview of these changes is next.

Registration: The $\mathcal{DA}$ servers calculate k tokens for each valid user R using a distributed message authentication scheme (MAC), and privately send the shares to each user who reconstructs the tokens. No one else learns the value of these tokens.

Submitting accusations: The user also provides a sharing of the secret R. The user must submit one of the tokens from the registration which the $\mathcal{DA}$ uses to validate the correctness of the sharing.

Discarding malformed and duplicated accusations: In order to check for duplicated accusations, instead of using the distributed equality testing proposed in WhoToo, we use the additively preserving structure of the secret sharing scheme to produce a combined distributed input $x = f(R, D)$ with an injective f. The $\mathcal{DA}$ computes a distributed input PRF on input x which then each server locally compares with previous submissions to detect duplicates.

Finding matching accusations: During this phase, we also propose an alternative to the equality testing used in the original protocol.

Using a threshold variant of Robust Anonymous IBE, the $\mathcal{DA}$s can distributively compute a valid encoding for a specific identity D starting from only the shares of D.

When an accusation is received, the $\mathcal{DA}$ servers use their shares of D to compute an encoding ρ_D, without learning anything about D. Once the quorum is reached and D is revealed, the servers compute and publish sk_D. Each server can attempt to decrypt $\rho_{D'}$ using sk_D for every other accusation (other than the one which triggered the quorum being reached); if successful, then it is a matching accusation. Since the IBE scheme is strongly robust (meaning ciphertexts do not reveal the intended recipient even when valid secret keys of different recipients are known), the privacy of not matching accusations is preserved.

3.2 Two Issues in WhoToo

In this section, we discuss two key aspects of the WhoToo protocol that (we believe) require some clarifications before a secure and working version of the protocol can be properly implemented. Although they are arguably small, finding a secure working solution seems to require, in one case, revisiting the security proof of a key component of the protocol and, in the other case, coming up with a secure yet not obvious subprotocol to compare four encrypted values in a pairwise fashion.

3.2.1 Securely Evaluating Quorum in WhoToo

There is an inconsistency between functions PrivatePoly.Multiply and Set.Quorum as stated in the published version of [43]. The first function takes two arguments: an encoded polynomial e_F and a polynomial R in the clear. Yet, in the WhoToo paper [43, Fig. 5], Set.Quorum invokes PrivatePoly.Multiply with two encoded polynomials. (The syntax is consistent with this interpretation as SecShare.Gen returns an encoding of the shared value when implementing with Pedersen VSS, as suggested in the paper.)

Of course, at first glance, the most reasonable explanation is a typo. Indeed, it seems we may simply take R (the second and random polynomial) in the clear. This solution, however, requires revisiting Lemma 6.5 [43, page 423] because this version of the Set.Quorum protocol does not hide all information on the set. We notice that since R is "in the clear", taking the derivative of the encoding of $(F \cdot R)'$ and then evaluating it on s (say obtaining a value s^*) reveals some information, as that last value only depends on (the actual, non-encoded) F and publicly known polynomial R. Moreover, we found a successful attack by a corrupted user and an honest but curious server. Consider the WhoToo protocol with quorum 2. Observe the moment when there is only one accusation in the system against s_1. Consider a corrupted user who makes 2 accusations against s_2 and s_3 and an honest but curious server who will learn the answers of Set.Quorum. Then $F(x) = (x - s_1)(x - s_2)(x - s_3)$. When running Set.Quorum, the servers will choose a random polynomial $R(x) = ax^3 + bx^3 + cx + \mathrm{d}$ and compute $(F \cdot R)''(s_3)$. Let $(F \cdot R)''(s_3) = s^*$, then we can isolate s_1 obtaining

$$s_1 = s^* - \frac{s^* - 2(s_3 - s_2)(as_3^3 + bs_3^2 + cs_3 + \mathrm{d})}{2(s_3 - s_2)(3as_3^2 + 2bs_3 + c) + 2(as_3^3 + bs_3^2 + cs_3 + \mathrm{d})}$$

Because all coefficients a, b, c are known, if the corrupted user and honest but curious server cooperate, they can use the values of s_2, s_3 and s^* to learn the value of s_1.

To prevent this attack, we decided to put forward a simple but robust modification of PrivatePoly.ZeroTest so value s^* is randomized if not null before being publicly exposed.

This randomization is done using protocol MsgRand [43], described in Figure 3.1. Figure 3.2 shows the algorithm modified accordingly, with changes shown in blue.

Lemma 3.2.1 The modified PrivatePoly.ZeroTest does not reveal more information other than if the shared value s is a root of the encrypted polynomial e_F.

MsgRand$(c = (c1, c2))$:

1. Each $\mathcal{DA}_\mathrm{i}$ does the following:
   (a) $r_\mathrm{i} \leftarrow_R \mathbb{Z}_p$
   (b) $a^\mathrm{i} \leftarrow (c_1^{r_\mathrm{i}}, c_2^{r_\mathrm{i}})$
   (c) $\pi_\mathrm{i} \leftarrow \mathsf{Prove}_{c_1,c_2}(a^\mathrm{i}, r_\mathrm{i})$
   (d) Publish $a^\mathrm{i}, \pi_\mathrm{i}$
2. For each $\mathcal{DA}_j$, each $\mathcal{DA}_\mathrm{i}$ does the following:
   (a) **if not** $\mathrm{Verify}_{c_1,c_2}(\pi_j, a^j)$: $a^j \leftarrow (1, 1)$
3. $c' \leftarrow (\prod_{\mathrm{i}=1}^n a_1^\mathrm{i}, \prod_{\mathrm{i}=1}^n a_2^\mathrm{i})$
4. **output** $m' \leftarrow \mathsf{ElGamal.DistDec}(c')$

Prove$_{c_1,c_2}(a = (a_1, a_2), r)$:

1. $k \in_R \mathbb{Z}_p$
2. $b_\mathrm{i} \leftarrow c_\mathrm{i}^k$
3. $\alpha \leftarrow H(a_1, a_2, b_1, b_2, c_1, c_2)$
4. $\beta \leftarrow \alpha k + r$
5. **output** $\pi \leftarrow (\beta, b_1, b_2)$

Verify$_{c_1,c_2}(\pi_\mathrm{i}, a^\mathrm{i})$:

1. $(\beta, b_1, b_2) \leftarrow \pi_\mathrm{i}$
2. $\alpha \leftarrow H(a_1, a_2, b_1, b_2, c_1, c_2)$
3. **output** $c_\mathrm{i}^\beta = a_\mathrm{i} b_\mathrm{i}^\alpha$

Figure 3.1: Randomized Decryption

Set.Quorum$(\mathrm{e}_F, \{s\})$:

Distributively generate q random values:

1. **for** $\mathrm{i} \in [q]$
   (a) $\{R_\mathrm{i}\} \xleftarrow{R} \mathbb{Z}_p$
   (b) $R_\mathrm{i} \leftarrow \{R_\mathrm{i}\}$
2. $R \leftarrow (R_0, ..., R_{q-1})$

Multiply by R, Differentiate $q-1$ times and Test if $(F \cdot R)(s) = 0$:

3. $\mathrm{e}_G \leftarrow \mathsf{PrivatePoly.Multiply}(\mathrm{e}_F, R)$
4. $\mathrm{e}_H \leftarrow \mathsf{PrivatePoly.Differentiate}(\mathrm{e}_G, q-1)$
5. **output** $\mathsf{PrivatePoly.ZeroTest}(\mathrm{e}_H, \{s\})$

PrivatePoly.ZeroTest$(\mathrm{e}_F, \{s\})$:

1. $\{r_1\} \xleftarrow{R} \mathbb{Z}_p$
2. $\{t\} \leftarrow \{r_1\}^{-1}$
3. **for** $\mathrm{i} \in [2, ..., \mathrm{d}]$
   (a) $\{r_\mathrm{i}\} \leftarrow \{r_{\mathrm{i}-1}\} \cdot \{r_1\}$
4. $\{x\} \leftarrow \{t\} \cdot \{s\}$
5. $x \leftarrow \{x\}$
6. $c_y \leftarrow \mathrm{e}_{F_0}$
7. **for** $\mathrm{i} \in [1, ..., \mathrm{d}]$
   (a) $\{s_\mathrm{i}\} \leftarrow x^\mathrm{i} \cdot \{r_\mathrm{i}\}$
   (b) $c_{y_\mathrm{i}} \leftarrow \mathrm{e}_{F_\mathrm{i}}^{\{s_\mathrm{i}\}}$
   (c) $c_y \leftarrow c_y c_{y_\mathrm{i}}$
8. **output** $\mathsf{MsgRand}(c_y) = g^0$

Figure 3.2: Modification in Set.Quorum and PrivatePoly.ZeroTest for consistency with PrivatePoly.Multiply. d stands for the degree of the polynomial. Operations involving shares use arguments with braces (eg. $\{x\}$) as described in Sect. 2.2.6.

Proof. (sketch) Consider the protocol PrivatePoly.ZeroTest from Fig. 3.2 and the corresponding ideal functionality $\mathcal{F}^F_{\mathrm{ZeroTest}}$ in Fig. 3.3. We argue that the view of the adversary can be simulated. This is easy for lines 1-4, as they only output random values or shares of random values. Due to the secrecy property of secret sharing, all SecShare.Operations can be simulated. Even the reconstructed values can be simulated, as they follow a uniform distribution, independent from the previous values. In fact, it suffices to notice that the result of the distributed decryption in MsgRand can be adjusted by the simulator depending on the output from $\mathcal{F}_{ZeroTest}$, given the honest majority for the servers. □

Lemma 3.2.2 The modified Set.Quorum hides all elements of the set, revealing only its size and whether or not the multiplicity of a shared value s is above of a fixed threshold.

Proof. Given a random polynomial R, both protocols PrivatePoly.Multiply and PrivatePoly.Differentiate can be publicly computed (by any server), so having R in the clear

$\mathcal{F}^{F}_{ZeroTest}$:

1. Upon creation, initialize $S \leftarrow \emptyset$.
2. Upon receiving a message $(\texttt{Add}, s, P)$ from party P, add s to multiset S, and then send $(\texttt{Element-Added})$ to the adversary. Then receive and ignore the adversary's reply.
3. Upon receiving a message $(\texttt{Multiplicity}, s, P)$ from party P, reply with message b equals true if $F(s) = 0$, and false otherwise.

Figure 3.3: Ideal Functionality for PrivatePoly.ZeroTest.

does not give the servers any more information than they already had. Lemma 4.1 tells us that PrivatePoly.ZeroTest reveals only the multiplicity of a given element s and no additional information, so the result follows. □

3.2.2 Identifying Duplicated Accusations

In WhoToo, in order to validate accusations, servers need to remove duplicate accusations. They are those that identify the same pair of accused D and accuser R. To eliminate them, servers must check whether a given pair $(s = H(D), R)$ has already appeared in some previous valid accusations, on the list `Accusations` of unopened accusations. The protocol uses $s = H(D) \in \mathbb{Z}_p$ as the identifier for the accused so that it can be secret shared. To guarantee privacy, values s and R are both encoded (encrypted) using a multiplicatively homomorphic ElGamal, as $\mathrm{e}_s = (g^{r_s}, g^s h^{r_s})$ and $c_R = (g^{r_R}, R \cdot h^{r_R})$ where r_s and r_R are random values used for the encryption of s and R respectively.

Let $\mathrm{e}_{s'}$ and $c_{R'}$ be encodings associated to an existing (valid) accusation in the set `Accusations`, say the i-th accusation on the list. The WhoToo protocol provides Equal, a distributed equality testing operation that, given two encodings, divides, randomized, and decrypts the result, so the only exposed value is 1 if the plaintexts are equal, and a uniformly random value if not. We could certainly use $\mathsf{Equal}(\mathrm{e}_s, \mathrm{e}_{s'})$ to compare whether two encodings e_s and $\mathrm{e}_{s'}$ have equal plaintexts s and s'. Protocol Equal indeed works perfectly and does not reveal additional information when two encodings are compared. However, it does not suffice in order to identify duplicate accusations, as we need to do more: we must compare e_s and c_R from two different accusations, simultaneously.

The original description for this step (Whotoo.VerifyAcc, line 4) calls for Equal on inputs $(\mathrm{e}_s, c_R), (\mathrm{e}_{s'}, c_{R'})$. The meaning of such call is confusing, at least, as the input comprises four ciphertexts, not two. No description nor explanation is given about how this equality test must be computed with four ciphertexts. At this point, the most natural solution is to compare them sequentially, say first $(\mathrm{e}_s, \mathrm{e}_{s'})$ and then $(c_R, c_{R'})$. This approach, unfortunately, compromises the anonymity of the accuser. Consider the case that $s = s'$ but $R \neq R'$. If we apply the above strategy, everyone learns that there are two accusations against s, and since later we learn that $R \neq R'$, the accusations are not duplicated, thus valid. If the comparison is made starting with $(c_R, c_{R'})$, it is easy to see that the anonymity of the accuser may be compromised this time. Even though no individual value for the "unequal" plaintext is computed, learning that there are other accusations in the system with the same identities would clearly compromise anonymity.

It is not trivial to adapt Equal for multiple inputs. The fact that it is unclear and left open for the protocol implementer to decide, could potentially compromise the security of the entire protocol.

In the following chapter we provide an alternative to this process which preserves anonymity and, in fact, is more efficient.

Chapter 4

Improving WhoToo

This chapter describes the efficiency improvements presented in `WhoToo`$^+$. We introduce modifications in two components, significantly reducing the number of online interactive operations in each one.

WhoToo uses the distributed equality testing mentioned in the previous section in two different components of the protocol: to identify repeated accusations (same pair accuser, accused, the case above) and to identify accusations for the same accused, in order to identify the accusations that triggered the quorum. In each of these components, the test is used to compare a specific accusation to all other unopened accusations. Therefore, the $\mathcal{DA}$ must compute as many distributed operations as there are unopened accusations. If there is a big backlog of accusations that have not reached the quorum, this becomes very inefficient. We propose alternatives to each of these components where the $\mathcal{DA}$ only needs to compute a constant number of distributed operations to obtain the same information.

4.1 Duplicate Revision

As mentioned in the previous section, this component aims to check whether a new accusation has already been submitted, by detecting if the same accuser R has already submitted an accusation against the same $s = H(D)$. Instead of following the (rather problematic) WhoToo approach, we propose an alternative inspired upon the SAE protocol. We use a distributed input pseudorandom function (DIPRF) introduced in SAE [5]. This DIPRF is a distributed variant of the distributed verifiable PRF from [27].

4.1.1 Distributed Input Pseudorandom Functions (DIPRF):

A DIPRF is a pseudo-random function where the key and input are secret shared among the computing parties. It outputs a sharing of the calculated pseudo-random value, which can be sent directly to a user U or simply reconstructed by the servers. The DIPRF introduced in SAE [5] can be securely and efficiently computed among the servers. Their function is also verifiable, but we ignore that feature. Indeed, we use the following result from Dodis and Yampolskiy [27]:

DIPRF.Setup():
1. $g \in_R \mathbb{G}$
2. $(\{sk_\mathrm{d}\}, pk_\mathrm{d} = g^{sk_\mathrm{d}}) \leftarrow$ SecShare.Gen(g)
3. Publish pk_d

DIPRF.Calculate($\{sk_\mathrm{d}\}, \{x\}$, `recipients`):
1. $\{t\} \leftarrow \{sk_\mathrm{d}\} + \{x\}$
2. $\{exp\} \leftarrow \{t\}^{-1}$
3. **if** `recipients` = `all_DAs`:
   (a) **output** $\mathrm{e}(g,g)^{\{exp\}}$
4. **else if**: `recipients` = U:
   (a) $\{res\} \leftarrow$ SecShare.ExpLocal $(\mathrm{e}(g,g), \{exp\}, U)$
   Shares of $\{res\}$ *are sent to user* U
5. **else if** `recipients` = `none`:
   (a) $\{res\} \leftarrow \mathrm{e}(g,g)^{\{exp\}}$

DIMAC.Setup():
1. $(pk_m, \{sk_m\}) \leftarrow$ DIPRF.Setup()
2. $V = \emptyset$

DIMAC.Tag($\{sk_m\}, \tau, U$):
1. $\{j\} \xleftarrow{R} \mathbb{Z}_p$
2. $\{p_j\} \leftarrow$DIPRF.Calculate($\{sk_m\}, \{j\}$, `none`)
3. $\{x\} \leftarrow \tau + \{p_j\}$
4. $\{\mathrm{d}_j\} \leftarrow$ DIPRF.Calculate($\{sk_m\}, \{x\}, U$)
5. Send shares of j and d_j to user U.

DIMAC.Verify($\{sk\}, \{\tau\}, \mathrm{d}_j, j$):
1. **if** $(\mathrm{d}_j, j) \in V$: **output** False
2. $\{p_j\} \leftarrow$ DIPRF.Calculate($\{sk_m\}, j$, `none`)
3. $\{x\} \leftarrow \{\tau\} + \{p_j\}$
4. $\mathrm{d}'_j \leftarrow$ DIPRF.Calculate($\{sk_m\}, \{x\}$, `all_DAs`)
5. $V \leftarrow V \cup \{(\mathrm{d}'_j, j)\}$
6. **output** $\mathrm{d}_j = \mathrm{d}'_j$

Figure 4.1: Distributed Input PRF and MAC. Operations involving shares use arguments with braces (eg. $\{x\}$) as described in Sect. 2.2.6.

Proposition 4.1.1 Given a bilinear group $\mathbb{G}$ with generator g where a q-Decisional Bilinear Diffie Hellman Inversion assumption holds, with $sk \in_R \mathbb{Z}_q$ and $pk = g^{sk}$, $F_{sk}(x) = \mathrm{e}(g,g)^{(1/(x+sk))}$ is a PRF.

A slight modification gives us a DIPRF over Type-III pairings, namely $F_{sk}(x) = \mathrm{e}(g_1, g_2)^{1/(x+sk)}$. We further extend it, based on a similar construction in [5], to obtain a distributed-input PRF variant whose sk is secret shared. It is shown in Fig. 4.1.

To define distributed input PRF, we follow the definition given by Naor, Pinkas, and Reingold [51], adapted and simplified for the shared input setting [5]. A distributed input PRF (DIPRF) is composed of three distributed protocols DIPRF.Setup, DIPRF.ShareInput, and DIPRF.Calculate. DIPRF.Setup is joinly computed by the servers on input the security parameter λ to compute $(\{sk\}, pk)$ where $\{sk\}$ are the local shares associated to the secret key sk, and pk is the corresponding public key. We stress that each share sk_i is locally output to server i while pk is publicly computed. Protocol DIPRF.ShareInput is joinly run by the servers and a user, and on input x, provided by the user, outputs local shares of $\{x\}$ for the servers. Protocol DIPRF.Calculate is run by the servers given a set of recipients, pk and two secret shared inputs $\{sk\}$ and $\{x\}$. The output is a value $y \leftarrow F_{sk}(x)$, where F is a pseudo-random function, and y is privately received by a pre-specified the set of receivers (or made public, if the receivers is set to `all_DAs`).

In terms of security, we extend the two main properties Consistency and Pseudorandomness proposed by Naor, Pinkas and Reingold [51] adapting them to our setting with secret shared inputs. A DIPRF is consistent even under a static adversary corrupting a minority of servers and possibly any user, if the output of DIPRF.Calculate equals $F_{sk}(x)$ where $\{sk\}$ is the secret key privately output by DIPRF.Setup and $\{x\}$ is secret shared defined by

computing DIPRF.ShareInput. A DIPRF is pseudorandom if the distributed evaluation of DIPRF.Calculate on secret key sk (output by DIPRF.Setup) and adversarially chosen input x is computationally indistinguishable from a random value even under the presence of an adversary A controlling a minority of the servers and any number of users such that (1) A initially obtains shares sk_{i} for any server i controlled by the adversary, and (2) A has oracle access to DIPRF.Calculate under secret key $\{sk\}$ and inputs of A's choosing but different from x.

We use a variant of a construction proposed in [5] (which was adapted to from the one in [27]), where $F\colon \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{G}$ is defined over a bilinear group $\mathbb{G}$. property. Protocol DIPRF.ShareInput is simply SecShare.Encode. Both protocol DIPRF.Setup and DIPRF.Calculate rely on SecShare operations (namely Gen,add, Inv,ExpLocal, and Exp). Since the execution of them all under a threshold static adversary are verifiable (robust) and simulatable under an ideal adversary given only the public values, the following result holds.

Lemma 4.1.2 Assuming the q-Decisional Bilinear Diffie-Hellman Inversion assumption holds on group $\mathbb{G}$, the scheme DIPRF (as specified in Fig. 4.1) is consistent and pseudorandom under any PPT adversary statically corrupting a minority of servers and any number of users.

Our solution: The intuition for our solution is the following. We create an input that non-malleably combines the values R and s, and let the servers jointly compute the DIPRF on that input. Upon receiving a valid accusation, each server gets and locally stores the associated DIPRF value. Then for every new accusation the servers only need to compute this value once and compare it with the previously stored ones. If a majority of servers agree that the new DIPRF value has not been calculated before, then the accusation is not duplicated.

Since we can not use R directly, we define $\tau = H'(R)$. When the accusation is prepared, the user must provide a secret sharing of τ. The combined input for the DIPRF will be $\{x\} = \{s\} + \{\tau\}$. Under the random oracle model, the value $s + \tau$ uniquely identify a valid pair of accused and accuser, except with negligible probability. This is captured by the following lemma.

Lemma 4.1.3 Let H, H' be two distinct, collision resistant, cryptographic hash functions into $\mathbb{Z}_p$. Consider random values $a_0, a_1' \in \mathbb{Z}_p$ and set $R_0 \leftarrow g_1^{a_0}, R_1 \leftarrow g_1^{a_1}$, $\tau_0 \leftarrow H'(R_0)$, $\tau_1 \leftarrow H'(R_1)$. In the random oracle model, for any PPT adversary A,

$$\Pr\left[\, (D_0, D_1) \leftarrow A(R_0, R_1) \;:\; D_0 \neq D_1,\; H(D_0) + \tau_0 = H(D_1) + \tau_1 \,\right]$$

is negligible.

Proof. Let $s_0 \leftarrow H(D_0)$, $s_1 \leftarrow H(D_1)$. We model H and H' as two distinct, independent random oracles. First, if $a_0 = a_1$ finding D_0 and D_1 that satisfy the condition implies that A is finding collisions in H. Now, if a_1 and a_2 are chosen uniformly and independently at random under $a_1 \neq a_2$, then R_1 and R_2 are also uniform and independent in $\mathbb{G}_1$ subject to

$R_1 \neq R_2$. Moreover, under the random oracle model, τ_0 and τ_1 are selected uniformly and independently in $\mathbb{Z}_p$ which implies $r = \tau_1 - \tau_0 \neq 0$ with overwhelming probability. So,

$$\Pr[\, s_0 + \tau_0 = s_1 + \tau_1 \,] = \Pr[\, s_1 - s_0 = r \,]$$

This last probability must be $1/p$, thus negligible in the security parameter.

□

4.1.2 Avoiding mismatched accusations:

If we only use the DIPRF in this way, a corrupted user could send shares of τ that do not match R, namely $\tau \neq H'(R)$. In order to ensure that there is no mismatch between R and τ, we build a Distributed Input MAC using the DIPRF in a somewhat standard way. The resulting DIMAC scheme is described in Figure 4.1.

Using the DIMAC scheme, we modify the protocol so that the $\mathcal{DA}$ computes τ during initialization and then calculates k values $T = \text{DIMAC}(\{sk\}, \tau)$, sending the shares of each T directly to the user, so that no servers learn any of the values of the DIMACs. Each value T is a pair (d_j, j), where j is a random value added to the DIPRF input. Our protocol uses k random j's to prevent reusing τ's and to prevent a malicious user from sending another user's τ and a random $(\mathrm{d}_{j'}, j')$, which even though would fail verification, would reveal and invalidate (spend) a real user's DIMAC.

When making a new accusation, a user U sends their shares of τ with any unused d_j and the corresponding j to the $\mathcal{DA}$. To ensure the shares of τ correspond to the value given during initialization, the $\mathcal{DA}$ verifies (recomputes) the DIMAC. As a consequence of this approach, our protocol only allows each user to submit at most k different accusations. We believe this is a reasonable tradeoff.

4.2 Matching accusations

Once Set.Quorum returns true, all accusations against the same perpetrator must be opened. In order to find the matching accusations, WhoToo performs a linear scan, distributively comparing each e_s in the Accusations set to the last accusation. If $N = |\mathsf{Accusations}|$, this means that every time the quorum is reached, N interactive distributed operations must be computed.

A naive approach to reduce this number is to generate public and private keys (pk_s, sk_s) for every possible s and encrypt every sk_s with the servers' public key. When making an accusation against s, the user could include an encryption of some value (say 1) with pk_s. Let $\rho_s = Enc_{pk_s}(1)$. Then, once the quorum is reached for a certain s, the servers could cooperate to obtain sk_s and each server could locally try to decrypt every ρ_s in the Accusations set. If the decrypted value is 1, then it is a matching accusation. This approach would require only one distributed operation, nevertheless, it has two significant issues: (1) generating and encrypting keys for every possible s is extremely inefficient and (2) we need to ensure that no information about accusations that do not match is revealed, which would require a robust encryption scheme that can guarantee $Dec_{sk_s}(Enc_{pk_{s'}}(1)) \neq 1$ for every $s \neq s'$.

We propose a distributed variant of a Strongly-Robust Identity-Based encryption scheme, which allows to have a similar approach solving both issues.

Strongly Robust Distributed IBE:

An Identity-Based Encryption is a public key encryption scheme in which any user can use public parameters and a specific identifier ID to compute a public key for that ID [35]. The corresponding secret key is computed by a centralized trusted authority using a master secret key, public parameters and the given ID. Notice that secret keys to decrypt messages encrypted with public keys can be created after the message is encrypted.

Going back to our problem, we first notice that using IBE encryption solves the first issue above, as secret keys can be computed when needed instead of pre-computing them for all possible identities. To solve the second issue, we need a new property for our (IBE) encryption scheme: Strong Robustness [3]. Strongly robust IBE guarantees that $Dec_{sk_{ID}}(Enc_{sk_{ID'}}(m)) = \perp$ for all m and for all $ID = ID'$. Indeed, this ensures that no information is revealed when trying to decrypt a message encrypted for a specific identity with a (valid) secret key that does not match that identity. This security notion is formalized as SROB [3].

From IBE to Distributed IBE: In our setting, however, we do not have a centralized trusted authority, so we create a variant with a distributed authority. To achieve our goal, the servers operate as a distributed key center, providing secret keys to decrypt IBE encrypted messages to any identity. Indeed, they first compute a shared master private key and publish public parameters that anyone can use to encrypt for any identifier ID. Furthermore, the servers (not the users) compute all IBE encryptions. Interestingly, our solution does not need to encrypt any message (the empty message suffices), it simply must provide an identity-based proof that the ciphertext was generated for the correct ID.

To describe our solution, we present the scheme we use as starting point, and how we modify it to make it distributed.

IBE Syntax: More formally, an identity-based encryption scheme is a 4-tuple (Setup,KeyGen, Enc, Dec) where Setup, on input λ (the security parameter), creates public parameters $params$ and a master private key msk, KeyGen, on input $params$ and a given ID, creates the corresponding secret key sk_{ID}, Enc, on input message M, encrypts it under a given ID using $params$, and Dec, on input ciphertext C and a secret key sk_{ID} returns the corresponding plaintext M.

Security Notions: Consider an IBE scheme $\mathcal{GE}$ = (Setup, KeyGen, Enc, Dec). $\mathcal{GE}$ is ANON-ID-CPA secure if any adversary has negligible probability of deciding whether a ciphertext C was encrypted for identity ID or ID'. The adversary has access to GetEK, GetDK and LR oracles. The first two, given an identity, provide the encryption and decryption keys respectively. The LR oracle receives two different identities that have not been asked to the GetDK oracle and returns an encryption for one of those. The adversary can make multiple queries to every oracle and it breaks ANON-ID-CPA security if it can guess with high probability which identities are being encrypted by the LR oracle. This is a direct simplification of the ANON-IND-ID-CPA definition of [3], which combines data privacy

(IND-CPA) and anonymity (ANON-CPA). We do not need IND-CPA as our scheme does not encrypt a message. $\mathcal{GE}$ achieves strong robustness (SROB) if any adversary has negligible probability of providing a ciphertext C and identities ID and ID' with $ID \neq ID'$ where $Dec(sk_{ID}, C) \neq \perp$ and $Dec(sk_{ID'}, C) \neq \perp$ [3].

We start from an identity-based encryption scheme proposed by Gentry [35] which was later modified to work with Type-3 bilinear groups and to achieve strongly robust ANON-IND-ID-CCA security by Okano et al. [53]. To obtain a strongly robust scheme (SROB), Okano et al. apply the transform proposed by Abdalla, Bellare and Neven [3]. Since our threat model is weaker (the servers will generate the IBE ciphertext, so no chosen-ciphertext is needed), we can simply use the original IBE scheme with the modifications for Type-3 pairings, which is ANON-IND-ID-CPA secure as long as the truncated decision q-ABDHE assumption holds for $(\mathbb{G}, \mathbb{G}_T, \mathrm{e})$ [35]. In the modified version, we require the truncated decision q-$ABDHE$ assumption to hold for $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e})$. Furthermore, since we do not encrypt any message, we do not need IND-CPA security, therefore, we aim to construct a strongly robust ANON-ID-CPA secure scheme.

The basic scheme: Consider a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, g_1, g_2)$ with no efficient known isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p where g_1 and g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. Let $H : \{0,1\}^* \to \mathbb{Z}_p$ be a universal one-way hash function. The following scheme IBE is a straightforward simplification of the scheme by Okano et al. [53].

$\mathsf{IBE.Setup}()$: The authority chooses random $g', h' \in \mathbb{G}_T$ and $h \in \mathbb{G}_2$. Then, it chooses a random value msk from $\mathbb{Z}_p$ as master private key, and a public key $g'_1 \leftarrow g_1^{msk}$. The private output is msk and the public output is $params = (g', h', g_1, g'_1, g_2, h)$ and msk.

$\mathsf{IBE.KeyGen}(params, \{msk\}, ID)$: The authority computes a random $r_{ID} \in \mathbb{Z}_p$ and $h_{ID} \leftarrow (h \cdot g_2^{-r_{ID}})^{1/(msk-ID)}$. The output is the private key $sk_{ID} = (r_{ID}, h_{ID} = (h \cdot g_2^{-r_{ID}})^{1/(msk-ID)})$.

$\mathsf{IBE.Enc}(params, \{ID\})$: The authority chooses random values $\mathrm{dec}, s \in \mathbb{Z}_p$ and computes $com \leftarrow g'^{ID} h'^{\mathrm{dec}}$, $C_1 \leftarrow g'^s_1 g_1^{-sID}$, $C_2 \leftarrow \mathrm{e}(g_1, g_2)^s$, and $C_3 \leftarrow h'^{\mathrm{dec}} \mathrm{e}(g_1, h)^{-s}$. It outputs ciphertext $C \leftarrow (com, C_1, C_2, C_3)$.

$\mathsf{IBE.Dec}(params, sk_{ID}, C, ID)$: Given sk_{ID}, it computes $h'^{\mathrm{dec}} \leftarrow \mathrm{e}(C_1, h_{ID}) \cdot C_2^{r_{ID}} \cdot C_3$ and outputs true if $com = g'^{ID} h'^{\mathrm{dec}}$ otherwise it outputs false.

The following lemma holds [53].

Lemma 4.2.1 Scheme IBE is ANON-ID-CPA secure and strongly robust (SROB) under the q-ABDHE assumption for $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e})$.

The Decentralized scheme: The scheme above can be adapted for a distributed trusted central authority. The scheme, called $\mathsf{DistIBE}$, is in Fig. 4.2. We remark that, despite the name, $\mathsf{DistIBE.Dec}$ is not distributed since sk_{ID} is known by everyone, so this operation is simply performed locally by each server.

DistIBE.Setup():

1. $g', h' \leftarrow^R \mathbb{G}_T$
2. $(\{\alpha_i\}_{t,n}, g_1' = g_1^{\alpha}) \leftarrow \mathsf{SecShare.Gen}(g_1)$
3. $h \leftarrow^R \mathbb{G}_2$
4. params $\leftarrow (g', h', g_1, g_1', g_2, h, H)$
5. $\{\text{msk}\} \leftarrow \{\alpha\}$
6. **output** params, $\{\text{msk}\}$

DistIBE.KeyGen$(params, \{msk\}, ID)$:

1. $\{\gamma\} \leftarrow \{msk\} - ID$
2. $\{\gamma_{Inv}\} \leftarrow \{\gamma\}^{-1}$
3. $r_{ID} \overset{R}{\leftarrow} \mathbb{Z}_p$
4. $h_{ID} \leftarrow (hg_2^{-r_{ID}})^{\{\gamma_{Inv}\}}$
5. $sk_{ID} \leftarrow (\text{ID}, r_{ID}, h_{ID})$
6. **output** sk_{ID}

DistIBE.Enc$(params, \{ID\})$:

1. $\{\text{dec}\} \overset{R}{\leftarrow} \mathbb{Z}_p$
2. $com \leftarrow g'^{\{ID\}} \cdot h'^{\{\text{dec}\}}$
3. $\{s\} \overset{R}{\leftarrow} \mathbb{Z}_p$
4. $\{t\} \overset{R}{\leftarrow} \mathbb{Z}_p$
5. $\{tInv\} \leftarrow \{t\}^{-1}$
6. $\{x\} \leftarrow \{t\} \cdot \{ID\}$
7. $a \leftarrow g_1'^{\{t\}} \cdot (g_1^{-1})^{\{x\}}$
8. $\{y\} \leftarrow \{s\} \cdot \{tInv\}$
9. $C_1 \leftarrow a^{\{y\}}$
10. $C_2 \leftarrow \mathrm{e}(g_1, g_2)^{\{s\}}$
11. $C_3 \leftarrow h'^{\{\text{dec}\}} \cdot (\mathrm{e}(g1, h)^{-1})^{\{s\}}$
12. $C \leftarrow (com, C_1, C_2, C_3)$
13. **output** C

DistIBE.Dec$(params, sk_{ID}, C, ID)$:

1. $h'^{\text{dec}} \leftarrow \mathrm{e}(C_1, h_{ID}) C_2^{r_{ID}} C_3$
2. **output** $com = g'^{ID} h'^{\text{dec}}$

Figure 4.2: Strongly Robust Distributed IBE. Operations involving shares use arguments with braces (eg. $\{x\}$) as described in Sect. 2.2.6.

In terms of security notions, we adapt both ANON-ID-CPA and SROB to the distributed setting in the obvious way.

Lemma 4.2.2 DistIBE is ANON-ID-CPA secure and strongly robust (SROB) under the q-ABDHE assumption for $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e})$ in the distributed setting.

Proof. We show that any attack against the distributed IBE scheme can be translated into an attack against the original Gentry IBE [35]. Fix a group of corrupted servers for an adversary $\mathcal{A}$ against which the secret sharing primitives are secure. Note that it is not necessary to fix a set of corrupted users because the distributed IBE scheme is only used by the servers. The users only provide shares of ID, which are verified by other components of the protocol. In order to simulate the view of the adversary, we can replace the distributed protocols with the original ones. Because of the correctness property of the secret sharing scheme and associated operations, we know that the outputs from DistIBE.Setup, DistIBE.KeyGen and DistIBE.Enc are indistinguishable from the outputs of IBE.Setup, IBE.KeyGen and IBE.Enc on the same inputs. We simulate all shares for corrupted servers with random values which are indistinguishable from the distributed IBE shares because of the secrecy property of the secret sharing scheme. These together with the public parameters $g', h', g_1, g_1', g_2, h, H$ allows to completely simulate the view of the adversary. In this way, we can create an adversary $\mathcal{B}$ against the original IBE scheme which has the same advantage as the adversary $\mathcal{A}$.

DistIBE is based on the IBE Scheme by Gentry [35], which is proven to be ANON-IND-ID-CPA secure. Abdalla, Bellare and Neven [3] prove that adding a binding and hiding commitment during encryption and verifying the commitment during decryption to an ANON-IND-

ID-CPA scheme results in a strongly robust ANON-IND-ID-CPA scheme. We use Pedersen commitment scheme [54] which is proven to be binding and hiding. From Lemma 5.1 we know that distributing operations during setup, key generation and encryption preserve ANON-IND-ID-CPA security. Therefore, just by applying Abdalla's Theorem we conclude our IBE scheme is strongly robust.

□

OUR SOLUTION: When an accusation is received, the servers encrypt a message with ID=s, which they obtain as a shared secret from the user in WhoToo.Accuse, and add it to Accusations. We will call this value ρ_s. Similarly to the naive approach, once the quorum is reached and s is revealed, the servers jointly compute the private key for $ID = s$ and then locally try to decrypt every ρ_s in Accusations with that key. If the value can be decrypted with that key, then it is an accusation against s.

Because it is a strongly robust scheme, the main result is a test of whether the message can be decrypted by that key, a test that anyone who knows the secret key for identity ID can perform. We can think of this scheme as an instance of searchable encryption [2], where the keywords that can be searched for are the identities.

Chapter 5

Variants and Extensions

In order to define the functional and efficiency requirements of the protocol, we interviewed experts who worked at Universidad de Chile with victims of sexual assault, as well as people who work in non-profit organizations which provide support to victims. We conducted four focus groups, which were characterized as follows:

1. Female students at the Law School Faculty of Universidad de Chile who had suffered and reported sexual assault at the university.
2. Female students at Faculty of Physical and Mathematical Sciences (FCFM) of Universidad de Chile who had suffered and reported sexual assault at the university.
3. Female students at Faculty of Physical and Mathematical Sciences (FCFM) of Universidad de Chile who had suffered sexual assault at the university, yet did not report it.
4. Female students at Faculty of Physical and Mathematical Sciences (FCFM) of Universidad de Chile who had never suffered sexual assault.

The focus groups had 3-6 participants each. We tried to reach people from other estates (academics or officials), however there were no volunteers. We also tried to conduct focus groups with male students, however, there were not enough volunteers to form a group.

The interviews were specific to each interviewees' profile. The focus groups guideline was constructed with the help of a psychologist who has worked directing focus groups for over 20 years, it can be found in Appendix 10.1. The focus groups were conducted in Spanish, therefore the guideline is also in Spanish.

After all of these interviews and conversations, we came to the following conclusions:

5.1 Discarded ideas

We believe it is important to describe and explain why we made the decision to discard the following ideas as it might be useful for future designs of this type of protocols.

We had the idea of giving users the option of categorizing accusations (for example, the

universities' regulation has the following categories: sexual assault, gender discrimination, gender violence, workplace harassment and arbitrary discrimination) but concluded this was not a good idea. People felt it might be hard to categorize a difficult experience and not knowing how to categorize it would lead to not making the accusation. We had also considered having different levels of accusations so that accusers could choose if they wanted the quorum to be reached within their category or any categories equal or higher. If it is hard to categorize an accusation, it is even harder to put it in a scale, so we discarded this idea as a whole.

We also thought about giving accusations an expiration date that could be either fixed for all accusations or chosen by each user. People where unanimously against this idea. However, they expressed that they would like to not be contacted about it after a certain amount of time. We delve into this idea in the following subsections.

We also discussed the idea of having different ways of matching accusations in case people did not know the name of the person they wanted to accuse. For example, they could state the faculty they study at, some social media handle or other information and if a certain amount of values match, then the accusations would be matched. This factor could be separated into primary and secondary, where primary are more "defining" (like social media handle) and then require less matching fields if those where primary. Nevertheless, we realized that in the context of the university, in almost every case people know the identity of the person so this would not be necessary. Additionally, people felt this could cause mistrust in the system for fear of errors in matching. Thanks to one of the interviews with experts in sexual assault at the university, we came up with an alternative that could help people who do not know the perpetrator's identity, which will also be discussed in the following sections.

5.2 Flexible quorum

One important difference between SAE and WhoToo is that SAE has a flexible quorum, meaning each user can choose the quorum they want for their accusation to be revealed. One of the main things we learned from the interviews and focus groups is that having more options and functionalities is not necessarily better. In a similar way to the categorization idea, having to select an option and not being sure which one to choose might cause the person to not submit the accusation. Reporting sexual assault is a very hard process and anything that makes it harder can lead to giving up on it.

However, some people expressed they would like to be able to choose their quorum. So after discussing this topic we came to the conclusion that if the quorum can be chosen, it should be from a very limited amount of options (for example, 2-5) and there should be a default value (for example 3).

In order to add a flexible quorum to WhoToo, the servers would need to have multiple privacy preserving polynomials $P_2, ..., P_\gamma$ where P_i is the polynomial for quorum i. When a user makes a new accusation with quorum q, it should be added to polynomials P_j for all $j \geq q$. This extension is simple and we are not certain it would benefit the system, therefore we do not include it in the `WhoToo`$^+$description, however, it can be easily added.

5.3 Role

People felt it was important to have statistics indicating the role of the perpetrator. To achieve this, victims could optionally mark their accusations indicating their relationship with the perpetrator. There could be different ways to inform the relationship between the accuser and the accused (for example: classmate, teaching assistant, boss, among others). This value could be included as plaintext in the `Accusations` set which will be described in the next chapter. Defining the possible roles and scenarios is out of the scope of this thesis, so we describe the interest in this characteristic together with a simple way to add it to WhoToo, yet we do not include it in the `WhoToo`$^+$description.

5.4 Unknown perpetrator

People who do not know their perpetrator should be able to submit accusations that will not add to the quorum but will count for the accusations total. Experts agree that it is important to give everyone space to make their accusation and be heard and offered psychological help. These accusations are not really submitted to the `WhoToo`$^+$ protocol, but if there was eventually a platform to submit accusations, people in this category should be given space to write their story if they want to.

People who want to continue with further investigation should not have to write their story because they will need to write it again together with lawyers and having them retell or rewrite their stories might be revictimizing.

5.5 Additional Public Information

One of the main discussions we had in all focus groups was determining which information should be publicly revealed even if the quorum has not been reached. We concluded that there are two kinds of information revealing:

1. Once the quorum is reached, the identities of the accusers and the accused should be revealed to the corresponding authority (for example, the Gender Division at the university). This information should not be public unless the accusers decide they want it to be.
2. Some specific statistics (like total number of accusations) should be public and should be informed to the community once every certain period of time (for example, every 6 months). This information is not protected by the protocol and can be accessed by any corrupted server. In this section, we describe the data that should be in this category.

5.5.1 Contact for further investigation

The process of reporting sexual assault is very difficult and potentially revictimizing and therefore victims would like to have the option of not being contacted. Some people would like to add to the quorum because they want others to know they have not been the only ones but do not wish to go further with a formal investigation. Therefore we have decided to

ElGamal.ProveEquality$(c_R,\ c'_R)$:

1. $x = \alpha - \beta$ `//`α `and` β `are the randomness for` c_R `and` c'_R `respectively`
2. $r \xleftarrow{R} \mathbb{Z}_q$
3. $a \leftarrow (a_1, a_2) = (g^r, h^r)$
4. $c = H(g, g^x, a)$
5. $\varphi \leftarrow r + cx$
6. $\theta \leftarrow (a, c, \varphi)$
7. **output** θ

ElGamal.VerifyEquality(c_R, c'_R, θ):

1. $(a, c, \varphi) \leftarrow \theta$
2. $(a_1, a_2) \leftarrow a$
3. $(y, z) \leftarrow c_R / c'_R$
4. **if** any of the following fail: **halt**
   (a) $g^\varphi = a_1 y^c$
   (b) $h^\varphi = a_2 z^c$
5. **output** True

Figure 5.1: Zero-knowledge proof of equality of ElGamal encodings

add a bit to the accusation that indicates if the victim wishes to be contacted for a further investigation. Nevertheless, all victims should be offered psychological help.

Accusers will be able to change this bit if they wish in case they change their minds.

5.5.2 Repetition counter

Even though duplicate accusations should not be accepted and should not add to the quorum, it should be possible to express a new assault by the same perpetrator. Therefore, we add a counter for each accusation which indicates the number of sexual assault episodes. Users can update this counter as many times as they want. This information will be public (in order to know the total amount of sexual assault episodes) but it will not add to the quorum.

5.6 Updates

As mentioned in the previous section, the contact bit and repetition counter can be updated so we need a way to update accusations. In order to do this, we will use the DIMAC d_j as an accusation id. At any time the user can use this id and their private key to update the accusation. If user U wants to update an accusation with a specific id, they have to send $(\mathrm{id}, c'_R, \texttt{info-to-change}, \theta)$ where c'_R is a new encryption of the user's public key R with the servers' private key, `info-to-change` is either `contact-bit` or `counter` and θ proves that c_R (sent when submitting the accusation) and c'_R are encodings of the same value.

For the purpose of providing a zero-knowledge proof of equality of c_R and c'_R, we use the zero-knowledge proof of discrete logarithm proposed by Chaum and Pedersen [23] as shown in Figure 5.1.

In this way, when the servers, receive $(\mathrm{id}, c'_R, \texttt{info-to-change}, \theta)$, they verify θ and update the contact bit (changing it) or counter (adding 1) of the accusation with that id accordingly.

Chapter 6

Description and Security of the New Protocol

In this chapter, we present the full version of $\texttt{WhoToo}^+$, including the fixes described in Chapter 3, the efficiency improvements introduced in Chapter 4 and the extensions presented in Chapter 5. We provide a detailed description together with the all the necessary algorithms (Section 6.1). Then, we sketch the security argument for $\texttt{WhoToo}^+$, following the outline of the original WhoToo proof as a sequence of games (Section 6.2).

6.1 $\texttt{WhoToo}^+$ Description

Fig. 6.1 presents a full description of $\texttt{WhoToo}^+$, including the modifications needed to use DIPRF for duplicate revision and DistIBE for finding matched accusations. Changes are shown in blue. We also provide a detailed description of the new protocol.

Initialization: During $\texttt{WhoToo}^+$.Initialize, the $\mathcal{DA}$ gets a list of valid users. Then, it runs the setup algorithms for DistBBS, DIPRF and DistIBE, obtaining the public and private keys (pk, msk), $(pk_{\mathrm{d}}, sk_{\mathrm{d}})$, and $(params, msk_{IBE})$ respectively. All private keys are distributed among the $\mathcal{DA}$ servers. It initializes an identity map and generates public and private keys (R, sk_U) for every user, recording that key R corresponds to user U in the identity map. sk_U is obtained as a shared secret and each server sends its share to user U, who reconstructs it. The $\mathcal{DA}$ also computes $\tau = H'(R)$, where H' is a one-way hash function. As mentioned in the previous section, the servers compute k authentication values (d_j, j) for each user. Finally, it initializes the privacy-preserving multiset S and the Accusations and UniqueAccs sets.

Submitting accusations: The process of submitting accusations is captured by $\texttt{WhoToo}^+$.Accuse and it consists of five main steps. First, user U prepares an accusation en sends $(acc)_{\mathrm{i}}$ to each $\mathcal{DA}_{\mathrm{i}}$ with its shares of the accusation. Then, the servers verify the accusation, ensuring it is not malformed or a duplicate. This steps are captured in $\texttt{WhoToo}^+$.PrepareAcc and $\texttt{WhoToo}^+$.VerifyAcc and will be described in detail in what follows.

If the accusation is verified, the servers compute ρ_s by using the distributed IBE scheme to encrypt for identity s, which they received as a shared secret in the first step. After, the $\mathcal{DA}$ adds the new accusation to the privacy-preseving multiset and to the `Accusations` set. Only s is added to the privacy-preseving multiset, while $\mathrm{id} = \mathrm{d}_j$, c_R, ρ_s, c_D, b and c are saved in the `Accusations` set. If the quorum is reached, c_R will be used to reveal the accuser's identity and both ρ_s and c_D will be needed to find the matching accusations. b is the contact bit and c is a public repetition counter, which is set to 1 for new accusations. Finally, the servers use Set.Quorum to verify if the quorum has been reached for identity s, in which case they run `WhoToo`$^+$.OpenAccusations.

Preparing accusations If user U with public key R wants to make an accusation against D, it first needs to calculate $s = H(D)$ and compute a secret sharing of s and $\tau = H'(R)$ (received during initialization) with randomness r_s and r_τ respectively. From this process, the accuser obtains $\{\omega\} = (\{s\}, \{r_s\})$, $v = (v_0, ..., v_t)$ (the Pedersen VSS verification values), $\mathrm{e}_0 = g_1^{r_s}$ and $\{\omega_\tau\}, v_\tau, \mathrm{e}_{0\tau}, r_\tau$. $s, \tau \in \mathbb{Z}_p$ are used during the protocol as the identities of the accuser and the accused, they are parallel to D and R. The user will also encode D with randomness r_D using ElGamal.EncString, obtaining c_D. For each $\mathcal{DA}$ server, U sets $m_\mathrm{i} \leftarrow c_D||\omega_\mathrm{i}||v||\mathrm{e}_0$ and signs m_i with the private key sk_U received during initialization, producing c_R and σ. The accuser also has to provide proofs of knowledge of s and D, so it computes π_0 and π_1 using ElGamal.Prove with encodings e_s, c_D and randomness r_s, r_D respectively. In order to prove that the shares of τ match the value received during initialization, U has to choose one of the unused d_j values received during initialization and add d_j and j to the accusation. At last, the user sends $acc_\mathrm{i} = (c_R, c_D, \omega_\mathrm{i}, v, v_\tau, \mathrm{e}_0, \mathrm{e}_{0\tau}, \sigma, \pi_0, \pi_1, \mathrm{d}_j, j, b)$ to $\mathcal{DA}_\mathrm{i}$, where b is the contact bit.

Verifying accusations: Once each server receives its part acc_i of an accusation, they must verify that the accusation is valid and not duplicate. In order to do so, they each verify the proofs π_0 and π_1 using the ElGamal public key computed during initialization (as one of the outputs of DistBBS.Setup). Then, they locally verify the signature σ and their shares ω_i using the verification values v. After, they all cooperate to verify that the shares ω are consistent with e_0 and that the MAC (d_j, j) is valid for ω_τ.

Once all these verifications are done, the $\mathcal{DA}$ proceeds to corroborate that the new accusation is not a duplicate from a previous one, namely, that there is not already an accusation from R to D in the system. Therefore, it computes the DIPRF with input $\{s\} + \{\tau\}$ making the result available to all servers. In this way, each $\mathcal{DA}_\mathrm{i}$ can locally verify if the resulting value p is in the set UniqueAccs. If $p \in$ UniqueAccs, the accusation is duplicated and should be discarded. Otherwise, they add p to UniqueAccs.

If none of these verifications fail, `WhoToo`$^+$.Verify outputs True.

Finding matching accusations: Once the quorum has been reached with a new accusation (c_R, ρ_s, c_D), the servers run `WhoToo`$^+$.OpenAccusations to find all accusations against the same s. First, the $\mathcal{DA}$ reconstructs s and generates the distributed IBE key sk_s for identity s running DistIBE.KeyGen. For each element $(c_{R'}, \rho_{s'}, c_{D'}, b, c)$ in Accusations, each server locally tries to decrypt $\rho_{s'}$ with key sk_d. If they can, it means the accusation was also

against s, so they distributively decrypt c_R and find the corresponding user U in the identity map. Then they decrypt $c_{D'}$, if $s = H(D')$, they set $D \leftarrow D'$. Once they find all matching accusations, they run $\mathsf{Investigate}(D, \text{Accusers})$, where Accusers is the set of all users U who submitted an accusation against D and its respective repetition counters and contact bits. Only accusers with $b = 1$ should be contacted for further investigation.

UPDATING ACCUSATIONS: If a user U wants to update an accusation, it must prepare the update with $\texttt{WhoToo}^{+}.\mathsf{RequestUpdate}$. The user sets $m \leftarrow (\text{id}, \texttt{contact-bit})$ or $m \leftarrow (t, \texttt{counter})$ depending on what they want to update. Then, the user signs the update, computing c'_R and σ. It also needs to compute θ to prove that c'_R encrypts the same value as c_R corresponding to Accusations[id]. If the counter is updated it always adds 1, while the contact bit is changed, so it is not necessary to provide additional information. Then, each $\mathcal{DA}$ server locally verifies the update by verifying the signature and using θ to verify that c_R and c'_R match. If these verifications are successful, the $\mathcal{DA}$ updates the element in Accusations[id] by updating the contact bit or increasing the repetition counter.

WhoToo$^+$.Initialize():

1. ValidAccuser$\leftarrow$ GetUsers()
2. $(pk, \{msk\}) \leftarrow$ DistBBS.Setup()
3. $(pk_{\mathrm{d}}, \{sk_{\mathrm{d}}\}) \leftarrow$ DIPRF.Setup()
4. $(pk_m, \{sk_m\}) \leftarrow$ DIMAC.Setup()
5. $(params, \{msk_{IBE})\} \leftarrow$ DistIBE.Setup()
6. IdentityMap $\leftarrow \emptyset$
7. **for** each $U \in$ ValidAccusers:
   (a) $R \leftarrow$ DistBBS.UserKeyIssue$_U(\{msk\})$
   (b) $\tau \leftarrow H'(R)$
   (c) IdentityMap$[R]\leftarrow U$
   (d) **for** i $\in [k]$:
      i. $\mathrm{d}_j, j \leftarrow$ DIMAC.Tag$(\{sk_m\}, \tau, U)$
8. $S \leftarrow$ Set.Init()
9. Accusations$\leftarrow \emptyset$
10. UniqueAccs $\leftarrow \emptyset$

WhoToo$^+$.Accuse():

1. U :
   (a) $(acc_{\mathrm{i}})_{\mathrm{i}\in[n]} \leftarrow$ WhoToo.PrepareAcc(D)
   (b) Send acc_{i} to server i of the $\mathcal{DA}$ over anonymous confidential channel
2. $\mathcal{DA}_{\mathrm{i}}$: **if** not WhoToo.VerifyAcc(acc_{i}): **halt**
3. $\rho_s \leftarrow$ DistIBE.Enc(params, $\{s\}$)
4. $S \leftarrow$ Set.Union$(S, \{s\})$
5. Accusations $\leftarrow$ Accusations $\cup$ $\{(\mathrm{id} = \mathrm{d}_j, c_R, \rho_s, c_D, b, c = 1)\}$
6. **if** Set.Quorum$(S, \{s\})$
   (a) Run WhoToo.OpenAccusations$(\rho_s, \{s\}, c_D)$

WhoToo$^+$.OpenAccusations$(\mathrm{id}, \rho_s, \{s\}, c_D, b, c)$:

1. Accusers $\leftarrow \emptyset$
2. $s \leftarrow \{s\}$
3. $D \leftarrow$ Null
4. $sk_{ID} \leftarrow$ DistIBE.KeyGen (params, $\{msk_{IBE}\}, \{s\}$)
5. **for each** $(\mathrm{id}, c_{R'}, \rho_{s'}, c_{D'}, b, c) \in$ Accusations
6. **if** DistIBE.Dec(params, $sk_{ID}, \rho_{s'}$)
7. $R' \leftarrow$ ElGamal.DistDec$(skeg, c_{R'})$
8. $U \leftarrow$ IdentityMap$[R']$
9. Accusers $\leftarrow$ Accusers$\cup\{(U, c)\}$
10. $D' \leftarrow$ ElGamal.DistDec$(\{skeg\}, c_{D'})$
11. **if** $H(D') = s$: $D \leftarrow D'$
12. **if** $D =$Null: **halt**
13. Run Investigate$(D,$ Accusers$)$

WhoToo$^+$.UpdateAcc:()

1. U:
   (a) $u \leftarrow$WhoToo$^+$.RequestUpdate
   (b) Send u to the $\mathcal{DA}$
2. $\mathcal{DA}$: WhoToo$^+$.ReceiveUpdate()

WhoToo$^+$.PrepareAcc(D):

1. $s \leftarrow H(D)$
2. $\{\omega\}, v, \mathrm{e}_0, r_s \leftarrow$ SecShare.Encode(s)
3. $\mathrm{e}_s \leftarrow (\mathrm{e}_0, v_0)$
4. $r_D \xleftarrow{R} \mathbb{Z}_p$
5. $c_D \leftarrow$ ElGamal.EncString$(pkeg, D, r_D)$
6. $\{\omega_\tau\}, v_\tau, \mathrm{e}_{0\tau}, r_\tau \leftarrow$ SecShare.Encode(τ)
7. **for each** i $\in [n]$
   (a) $m_{\mathrm{i}} \leftarrow c_D||\omega_{\mathrm{i}}||v||\mathrm{e}_0$;
   (b) $(c_R, \sigma) \leftarrow$ BBS.Sign(m_{i}, sk_U)
   (c) $\pi_0 \leftarrow$ ElGamal.Prove$(\mathrm{e}_s, r_s, c_R||\sigma)$
   (d) $\pi_1 \leftarrow$ ElGamal.Prove$(c_D, r_D, c_R||\sigma)$
   (e) $acc_{\mathrm{i}} \leftarrow (c_R, c_D, \omega_{\mathrm{i}}, \omega_{\tau_{\mathrm{i}}}, v, v_\tau, \mathrm{e}_0, \mathrm{e}_{0\tau}, \sigma, \pi_0, \pi_1, \mathrm{d}_j, j, b)$
8. **output** $(acc_{\mathrm{i}})_{\mathrm{i}\in[n]}$

WhoToo$^+$.VerifyAcc$(c_R, c_D, \omega_{\mathrm{i}}, \omega_{\tau_{\mathrm{i}}}, v, v_\tau, \mathrm{e}_0, \mathrm{e}_{0\tau}, \sigma, \pi_0, \pi_1, \mathrm{d}_j, j)$:

1. $\mathrm{e}_s \leftarrow (\mathrm{e}_0, v_0)$
2. $m_{\mathrm{i}} \leftarrow c_D||\omega_{\mathrm{i}}||v||\mathrm{e}_0$
3. $(s_{\mathrm{i}}, r_{\mathrm{i}}) \leftarrow \omega_{\mathrm{i}}$
4. $\tau_{\mathrm{i}}, r_{\tau_{\mathrm{i}}} \leftarrow \omega_{\tau_{\mathrm{i}}}$
5. **if** any of the following fail: **output** False
   (a) DIMAC.Verify$(\{sk_m\}, \{\omega_\tau\}, \mathrm{d}_j, j)$
   (b) ElGamal.Verify$(pkeg, \pi_0, \mathrm{e}_s, c_R||\sigma)$
   (c) ElGamal.VerifyString$(pkeg, \pi_1, c_D, c_R||\sigma)$
   (d) DistBBS.Verify$(pk, m_{\mathrm{i}}, c_R, \sigma)$
   (e) SecShare.Verify(ω_{i}, v)
   (f) SecShare.Verify$(\omega_{\tau_{\mathrm{i}}}, v_\tau)$
   (g) SecShare.CheckConsistent$(\{r_s\}, v, \mathrm{e}_0)$
   (h) SecShare.CheckConsistent$(\{r_\tau\}, v_\tau, \mathrm{e}_{0\tau})$
6. $\{x\} \leftarrow \{s\} + \{\tau\}$
7. $p \leftarrow$ DIPRF.Calculate$(\{sk_{\mathrm{d}}\}, \{x\},$ `all_DAs`$)$
8. **if** p in UniqueAccs: **output** False
9. UniqueAccs $\leftarrow$ UniqueAccs$\cup\{p\}$
10. **output** true

WhoToo$^+$.RequestUpdate()

1. $m \leftarrow (\mathrm{id},$ `info-to-change`$)$
2. $(c'_R, \sigma) \leftarrow$ DistBBS.Sign(m, sk_U)
3. $\theta \leftarrow$ ElGamal.ProveEquality(c_R, c'_R)
4. **output** $(m, c'_R, \sigma, \theta)$

WhoToo$^+$.ReceiveUpdate$(m, c'_R, \sigma, \theta)$

1. $(\mathrm{id},$ `info-to-change`$) \leftarrow m$
2. $c_R \leftarrow$ Accusations[id].c_R
3. **if** any of the following fail: **halt**
   (a) ElGamal.VerifyEquality(c_R, c'_R, θ)
   (b) DistBBS.Verify(pk, m, c'_R, σ)
4. **if** `info-to-change` = `counter`: Accusations[id].c += 1
5. **else:** Accusations[id].b = **not** Accusations[id].b

Figure 6.1: The WhoToo$^+$ Protocol

6.2 Security Analysis

In this section, we provide a security argument for the $\texttt{WhoToo}^+$ protocol, which follows the outline of the proof for WhoToo [43] adjusted to consider the new primitives we require. For the proof, we assume composable notions of security of its components hold under adversaries corrupting a minority of the servers. In particular, this includes the anonymously-robust identity-based threshold encryption DistIBE from Sect. 4.2, and the privacy-preserving multisets Private.Poly from Sect. 2.2.8. Indeed, to obtain composability under threshold adversaries, we heavily rely on the fact that the secret sharing operations SecShare used to implement the above distributed operations are verifiable, robust, and simulatable (given the appropriate ideal functionality) from only public outputs under a static adversary that corrupts up to $t < n/2$ servers, under the discrete logarithm assumption [34]. We also need to assume the security of the distributed group signature DistBBS from Sect. 2.2.7, and the distributed input pseudo-random function DIPRF from Section 4.1.1. Finally, we assume the security of the threshold ElGamal encryption scheme ElGamal (and its associated extractable proofs of knowledge) from Sec. 2.2.6. In our proof, we show that $\texttt{WhoToo}^+$ implements the ideal functionality from Fig. 6.2 [43].

Theorem 6.2.1 Let $\mathcal{A}$ be a real world adversary attacking $\texttt{WhoToo}^+$ protocol and corrupting up to t servers and any number of users. Under the assumption stated above, the execution of $\texttt{WhoToo}^+$ protocol under $\mathcal{A}$ can be simulated by an ideal adversary given the ideal functionality $\mathcal{F}_{\texttt{WhoToo}^+}$.

Initialize:

1. ValidAccusers $\leftarrow$ GetUsers()
2. Accusers $\leftarrow \emptyset$
3. UniqueAccs $\leftarrow \emptyset$

Accuse:

1. ValidAccusers $\leftarrow$ GetUsers()
2. Receive D from user U
3. **if** $U \notin$ ValidAccusers: **halt**
4. **else if** $(D, U) \in$ UniqueAccs: **halt**
5. **else**:
   (a) Accusers$[D] \leftarrow$ Accusers$[D] \cup \{U\}$
   (b) UniqueAccs $\leftarrow$ UniqueAccs $\cup \{(D, U)\}$
   (c) **if** $|\text{Accusers}[D]| \geq q$:
      i. Run Investigate(D, Accusers$[D]$)

Figure 6.2: $\mathcal{F}_{\texttt{WhoToo}^+}$: Ideal Functionality for $\texttt{WhoToo}^+$

As mentioned above, under the discrete logarithm assumption, each secret sharing operation SecShare from Sec. 2.2.6 is secure (as multiparty protocol) against a PPT adversary that statically corrupts a minority of the servers and any number of users. This follows from [34] which also relies on Pedersen VSS [54]. This result implies that the execution of each operation under such an adversary can be robustly computed, and the view of all parties can be simulated from public outputs alone.

For the DIPRF scheme, we assume it is consistent and pseudorandom (as in Section 4.1.1). The security of our DIMAC protocol (unforgeability under chosen message attacks, UF-CMA) easily follows as the latter is simply the canonical usage of DIPRF.

For the security of the distributed group signature, we rely on Lemma 5.1 from [43] which states that, assuming SecShare is secure, any threshold adversary attacking the construction DistBBS in the distributed setting can translated to an adversary to the original non-distributed BBS scheme (as defined in [43, Sect. 3.5]). This will suffice for our proof.

We now sketch the proof. Let A be a PPT adversary that statically corrupts up to t out of the $n >= 2t + 1$ servers and an arbitrary number of users. Our argument will proceed in steps, considering a sequence of games $G_1, \ldots, G_\ell$, where G_1 is the $\texttt{WhoToo}^+$ in real world under the real world adversary $A_1 = A$, and G_ℓ is the ideal world with functionality $F_{\texttt{WhoToo}^+}$ and ideal adversary A_ℓ. In each game G_i for $\mathrm{i} = 2, \ldots, \ell$, we show how adversary $A_{\mathrm{i}-1}$ can be replaced by a new adversary A_i in such a way that the view for all players and adversary $A_{\mathrm{i}-1}$ in game G_1 is computationally indistinguishable from the view for all players and adversary A_i. This will prove the theorem.

GAME G_1: This game is the execution of the $\texttt{WhoToo}^+$ protocol under adversary $A_1 = A$.

GAME G_2: (Make DistBBS and ElGamal centralized). In this game, we replace each distributed SecShare operation in DistBBS and ElGamal with their corresponding ideal functionality. Let ${\texttt{WhoToo}^+}_2$ be an augmented version of $\texttt{WhoToo}^+$ where all secret shared operations in DistBBS and ElGamal are executed by a trusted third party that runs the corresponding operation. Under the discrete log assumption, all SecShare operations are robust and simulatable from the public values. It immediately implies that the view for all players and adversary A_1 can be simulated by an ideal adversary A_2 only from the public values computed by the trusted third party.

GAME G_3: (Make DIPRF and DIMAC centralized). In this game, adversary A_2 is attacking protocol ${\texttt{WhoToo}^+}_2$. Let ${\texttt{WhoToo}^+}_3$ be the same protocol where we replaced the secret share SecShare operations in DIPRF and DIMAC with their corresponding ideal functionality executed in a trusted-third party. As distributed protocol DIPRF relies on the DKG protocol from Gennaro et al. [33], one can simulate the key generation so sk_d and sk_m are known to the simulator. For DIPRF (when invoked as part of DIMAC as well by itself), however, we go further, using an ideal functionality that chooses random values as output (and remembers them). Prop. 4.1.1 guarantees that this approach is sound: no adversary can distinguish $F_{sk}(x)$ from values chosen uniformly at random, where F is the PRF function behind DIPRF. Then, under the q-Decisional Bilinear Diffie Hellman Inversion, it is possible to simulate the view of the adversary in both protocols given those random outputs (except with negligible probability). We let A_3 be the adversary that works like A_2 but, instead of running the SecShare operations, performs the simulation as described. We conclude that the execution of protocol ${\texttt{WhoToo}^+}_2$ under A_2 is computationally indistinguishable from the execution of protocol ${\texttt{WhoToo}^+}_3$ under adversary A_3.

GAME G_4: (Make DistIBE centralized). Our goal is to replace each distributed SecShare operation in DistIBE with their corresponding ideal functionality. In this game, adversary A_3 is attacking protocol ${\texttt{WhoToo}^+}_3$. Let ${\texttt{WhoToo}^+}_4$ be the same protocol where we replaced the secret share SecShare operations in DistIBE with their corresponding ideal functionality (namely the IBE scheme) executed in a trusted-third party. Under a similar argument to the

one used in previous games, we argue that there is an adversary A_4 whose interaction with protocol $\texttt{WhoToo}^+_4$ is indistinguishable.

GAME G_5: (Replace the remaining secret share operations for their ideal functionality). Protocol $\texttt{WhoToo}^+_4$ still includes two calls to SecShare.Encode (in $\texttt{WhoToo}^+$.PrepareAcc), and two to SecShare.Encode and SecShare.CheckConsistent (both in $\texttt{WhoToo}^+$.VerifyAcc). Each one of these operations can be substituted by a call to their natural ideal functionality (see [28]) in a simulatable way. Let $\texttt{WhoToo}^+_5$ be $\texttt{WhoToo}^+_4$ with those substitutions. If we let A_5 be as adversary A_4 with the extra code to simulate the calls to the ideal functionalities, then, under the discrete log assumption, the execution of $\texttt{WhoToo}^+_4$ under adversary A_4 is computationally indistinguishable from that of $\texttt{WhoToo}^+_5$ under adversary A_5.

GAME G_6: (Make PrivatePoly centralized). Using an argument analogous to the one used in previous games, we argue we can replace the secret share SecShare operations in PrivatePoly.MultiplyLinear and PrivatePoly.ZeroTest with their corresponding ideal functionalities. This yields protocol $\texttt{WhoToo}^+_6$ and adversary A_6.

GAME G_7: (From encrypted polynomials to Set Operations). In this game, our goal is to replace the (centralized) PrivatePoly operations (namely Substract, Differentiate, Multiply, Multiplylinear, and ZeroTest) with (centralized) versions of Set.Init, Set.Add, and Set.Quorum in G_6. To simplify the proof, we assume stronger abstractions of the Set operations: Set.Init, Set.Add and Set.ZeroTest do not output encrypted polynomials but only output the size of the resulting set after performing the operation. Additionally, Set.ZeroTest returns the test result. We call $\texttt{WhoToo}^+_7$ the resulting protocol.

Recall that, in this game, adversary A_6 is interacting with $\texttt{WhoToo}^+_6$. In this protocol, all players (servers, users, and adversary A_6) have access to the encrypted polynomials given as inputs and returned as outputs during the execution of the PrivatePoly operations. They do not have access, however, to any individual element value s used as input to MultiplyLinear, and ZeroTest as such values are privately sent from the participant to the trusted third-party running those operations.

It is easy to see that, under DDH assumption over $\mathbb{G}_1$, all the encrypted polynomials are simulatable from the output of the Set operations under the stronger abstraction mentioned before. We call such an adversary A_7. Computational indistinguishability between the execution of $\texttt{WhoToo}^+_7$ under A_7 and $\texttt{WhoToo}^+_6$ under A_6 follows from [43, Lemma 6.4] which shows that set operations achieve perfect completeness and computational hiding, even under adversarially chosen inputs, as well as lemma 3.2.1.

GAME G_8: (Removing mismatched accusations). We now show that A_7 can be reduced to an adversary A_8 who always instructs any corrupted player R to submit accusations with correct τ, that is, $\tau = H'(R)$. The protocol for the game remains the same, so $\texttt{WhoToo}^+_8 = \texttt{WhoToo}^+_7$.

Let $\tau = \mathsf{SecShare.Reconstruct}(\{\omega_\tau\})$ and $R = \mathsf{ElGamal.Dec}(c_R)$ such that $\tau \neq H'(R)$ for a given accusation. We call this a mismatched accusation. (Notice that mismatched accusations can only come from corrupted users.) There are two possible cases:

1. $\tau \notin \{H'(R_\mathrm{i}) \,|\, R_\mathrm{i} \in \text{IdentityMap}\}$. In this case, for the accusation to be valid, the adversary must also present a valid credential, namely a valid MAC tag (d, j) for message τ, such that d is the value returned by DIPRF.Calculate on input $\tau + y$ where $y = \mathsf{DIPRF.Calculate}(j)$ associated to a randomly chosen j. We first observe that, during the execution, the adversary only sees valid credentials: during initialization, those obtained by corrupted users, namely $(\tau_\mathrm{i}, (\mathrm{d}_\mathrm{i}, j_\mathrm{i}))$ for some valid (yet corrupted) $R_\mathrm{i} \in$ IdentityMap, so $\tau_\mathrm{i} = H'(R_\mathrm{i})$, and, from previous (non mismatched) accusations, those submitted by honest users: $(\tau', (\mathrm{d}', j'))$ where $\tau' = H'(R')$ for some valid R'. Therefore, any success on producing a new credential $(\tau, (\mathrm{d}, j))$ must occur with negligible probability as it involve guessing a random value (recall that DIMAC is computed with an ideal version of DIPRF.Calculate which returns consistent random values). In consequence, the accusation will be discarded with high probability. This interaction can be easily simulated from public values, so A_7's interaction with the protocol is indistinguishable from the interaction with an adversary A_8 whose corrupted users do not submit mismatched accusations.

2. $\tau = H'(R')$, where R' is the public key for another user. We have two scenarios:

   (a) The user associated to R' is corrupted, so the adversary knows valid credentials $(\tau, (\mathrm{d}, j))$: In this case, the adversary is able to submit two accusations against some D, one from $(R, \tau = H'(R'))$ and another one from $(R, \tau_R = H'(R))$. But this is done at a cost: now R' cannot lodge (another) accusation against the same D because such accusation would be discarded as duplicated, as it would also come with $\tau = H'(R')$. Even though R' could use now a τ from another corrupted user, since used credentials are "burned" upon validation (stored in a list V of used credentials), it is easy to see that this approach is doomed to failure: the number of accusations the adversary can submit against certain D is no larger than the number of accusation corrupted users can submit as a group. Furthermore, even in this case where two accusations are submitted under the same R's but two different τ's, servers get shares for the submitted τ's. When opening an accusation, servers can reconstruct both sharings and recover both identities. Consequently, the adversary's effect could be emulated by submitting two separated accusations from each corrupted user: one accusation from $(R, \tau_R = H'(R))$ and another from $(R', \tau = H'(R'))$, namely, by submitting no mismatched accusations.

   (b) The user associated to identity R' is honest: Just like the case (1) before, producing valid credentials $(\tau, (\mathrm{d}_j, j))$ only happens with negligible probability as DIMAC.Tag returns a a uniformly-chosen random value (with overwhelming probability). Reusing previously used credentials by honest R' is not possible as valid used credentials are blacklisted (stored in list V during DIMAC.Verify) once validated.

We conclude that adversary A_7 can be reduced to another adversary A_8 who works the same except that never submits mismatched accusations.

GAME G_9: (Removing invalid accusations). In this game, we want to show that any adversary A_8 can be reduced to another adversary who never submits invalid accusations. The protocol is unchanged, so $\texttt{WhoToo}^+{}_9 = \texttt{WhoToo}^+{}_8$.

First, we say an accusation acc is invalid if $\texttt{WhoToo}^+$.VerifyAcc(acc) in step 5 returns `false`. For step 5 to return `false`, one of the following operations must return `false`: DIMAC.Verify, ElGamal.Verify, ElGamal.VerifyString, BBS.Verify, SecShare.Verify, or SecShare.CheckConsistent. Clearly, except for DIMAC.Verify, none of these operations change the state of the system when `false` is returned. Moreover, ElGamal.Verify, ElGamal.VerifyString, and BBS.Verify are executed with public parameters as inputs (other than the accusation) so they are simulatable. In addition, SecShare.Verify, and SecShare.CheckConsistent are already executed in a centralized fashion (since game G_5) so simulation is trivial.

It remains to prove that DIMAC.Verify can be simulatable if the accusation is invalid. Indeed, we argue that is always simulatable, not only when the accusation is invalid, as follows. Our idealized version of DIMAC.Verify returns `false` except if the provided credentials are among the set of credentials returned by DIMAC.Tag during initialization. Since adversary A_8 gets to see the credentials obtained by corrupted users, a good strategy for a new adversary A_9 is to return `false` unless the provided credentials are among the set of values returned by DIMAC.Tag to all users. In consequence, if DIMAC.Verify returns `true`, it must be that R's credentials are valid and, moreover, they must have been those obtained by some user during initialization. If that user was corrupted, some of the other checks must fail for the accusation to be invalid – which is simulatable as mentioned before. The case when R's credentials belong to some honest user cannot happen except with negligible probability because credentials are transmitted to honest users via private channels.

We thus conclude that, with high probability, A_8 can be reduced to a new adversary A_9 who never submits invalid accusations.

$\mathcal{F}_{DR}$.**Initialize**:
1. UniqueAccs $\leftarrow \emptyset$

$\mathcal{F}_{DR}$.**CheckDuplicate**:
1. Receive D from user U
2. **if** $(D, U) \in$ UniqueAccs:
   (a) **output** True
3. **else**:
   (a) UniqueAccs $\cup\{(D, U)\}$
   (b) **output** False

Figure 6.3: $\mathcal{F}_{DR}$: Ideal Functionality for duplicate revision

$\mathcal{F}_{MA}$.**Initialize**:
1. Accusers $\leftarrow \emptyset$

$\mathcal{F}_{MA}$.**AddAccusation**:
1. Receive D from user U
2. Accusers[D] $\leftarrow$ Accusers[D]$\cup\{U\}$

$\mathcal{F}_{MA}$.**OpenAccusations**:
1. Receive D
2. **output** Accusers[D]

Figure 6.4: $\mathcal{F}_{MA}$: Ideal Functionality for matching accusations

GAME G_{10}: (Discarding duplicate accusations using DIPRF can be simulated with access to

$\mathcal{F}_{DR}$). First, let $\texttt{WhoToo}^+_{10}$ be as $\texttt{WhoToo}^+_{9}$ but where steps 6 through 10 are replaced by a call to $\mathcal{F}_{DR}$.CheckDuplicate.

Consider the functionality $\mathcal{F}_{DR}$ in Figure 6.3. We show that there exists an adversary A_{10} with access to ideal functionality $\mathcal{F}_{\mathcal{DR}}$ such that the execution of $\texttt{WhoToo}^+_{9}$ under A_9 is computationally indistinguishable from A_{10} interacting with $\texttt{WhoToo}^+_{10}$.

For the simulation, we remember our strategy in game G_3: instead of simulating the execution of DIPRF operations by an adversary who can select sk_{d} (as in [33]), we proceed by simply returning random values whenever DIPRF.Calculate is invoked with the same value. This approach, however, does not work here as the input to DIPRF.Calculate is secret shared and unknown to the simulator. To make it work, it suffices to consider that no honest user will ever submit duplicated accusations (same pair (D, R)). So, the (centralized) operation DIPRF.Calculate returns a random value each time is invoked in line 7 of $\texttt{WhoToo}^+$.VerifyAcc with an accusation by an honest user. If invoked by a corrupted user, the simulator has access to (s, τ) so DIPRF.Calculate can distinguish whether it was invoked with a new accusation (ie. new pair (s, τ)) or an existant accusation (ie. a pair (s, τ) already queried to DIPRF.Calculate). This strategy suffices to simulate $\texttt{WhoToo}^+$ when $\mathcal{F}_{DR}$ is called instead of lines 6 through 10.

At this point, we have shown that A_9 interacting with $\texttt{WhoToo}^+_{9}$ can be computationally simulated by A_{10} interacting with $\texttt{WhoToo}^+_{10}$.

GAME G_{11}: (Identifying matching accusations using DistIBE can be simulated with access to $\mathcal{F}_{MA}$). Let $\texttt{WhoToo}^+_{11}$ be $\texttt{WhoToo}^+_{10}$ where the DistIBE operations are replaced by calls to $\mathcal{F}_{MA}$ (Figure 6.4) as follows:

- DistIBE.Setup in $\texttt{WhoToo}^+$.Accuse, line 5, is replaced by and $\mathcal{F}_{MA}$.Initialize,
- DistIBE.Enc in $\texttt{WhoToo}^+$.Accuse, line 3, is replaced by and $\mathcal{F}_{MA}$.AddAccusation,
- DistIBE.KeyGen in $\texttt{WhoToo}^+$.OpenAccusations, line 3, and
- DistIBE.Dec in $\texttt{WhoToo}^+$.OpenAccusations, line 6.

We argue that the A_{10} on $\texttt{WhoToo}^+_{10}$ can be simulated by a new adversary on $\texttt{WhoToo}^+_{11}$.

It suffices to show the following: (1) IBE.setup can be simulated by an adversary with oracle access to $\mathcal{F}_{MA}$.Initialize, (2) DistIBE.Enc can be simulated (computed) from already shared values, (3) IBE.KeyGen correctly compute the secret key from $\{s\}$, and (4) IBE.Dec does not reveal any information on any ciphertext except those encrypted under identity s. All of them are direct, except (4) which follows from the ANON-IND-ID-CPA property for IBE.

GAME G_{12}: (Simulate accusations (by both honest and corrupted users)).

Let $\texttt{WhoToo}^+_{12}$be as $\texttt{WhoToo}^+_{11}$but where accusations by honest and corrupted users are simulated.

Let *acc* be an accusation made by an honest user. ElGamal encodings c_D, e_s, c_R are simulatable because they are indistinguishable from encodings of random values. σ can be

simulated by signing with any identity from the group, following the anonymity of BBS signatures. Shares ω, ω_τ are also indistinguishable from shares of random values. Finally, j and d_j are also indistinguishable from random values given that j is chosen at random and d_j is the result of a PRF. The resulting shares of DIMAC.Verify can be produced by the honest majority of servers so that they reconstruct to the required value so it can be verified.

Let *acc* be an accusation made by a dishonest user. At this point we know that there are no malformed accusations, as we have removed invalid accusations. Therefore, considering there is an honest majority of servers, the simulator controls all shared secret keys. Following the traceability property of group signatures, R can be obtained from c_R. Furthermore, given that π_1 is a zero-knowledge proof of c_D, D can be extracted. Then, the simulator can map R to U and send (D, U) to $\mathcal{F}_{\texttt{WhoToo}^+}$. If the quorum is reached, the simulator can adapt the shares of honest servers in order to obtain the matching accusations output by $\mathcal{F}_{\texttt{WhoToo}^+}$. Because there is an honest majority, the simulator controls the ElGamal, BBS, DIPRF, DIMAC and DistIBE secret keys and therefore this is simulatable. We conclude that the execution of $\texttt{WhoToo}^+{}_{12}$under $A_{12}$is indistinguishable from the execution of $\texttt{WhoToo}^+{}_{11}$under A_{11}.

GAME G_{13}: We argue this is the ideal world with the ideal adversary interacting with the ideal $\texttt{WhoToo}^+$ functionality. This concludes the proof.

Chapter 7

Implementation and Efficiency

In this chapter we describe the `WhoToo`$^+$ prototype, explaining its main components (Section 7.1). Using this prototype we provide an experimental efficiency analysis, comparing the time taken to setup the system and to make new accusations in WhoToo and `WhoToo`$^+$(Section 7.2.2). We also present a theoretical efficiency analysis, describing the changes in numbers of offline and online distributed operations (Section 7.2.1).

7.1 Prototype

We built our prototype in Python using the charm-crypto framework [22] for all bilinear group operations. We use the 254-bit BN254 Barreto-Naehrig curve [8] implemented in the PBC library [48]. The prototype is open source, and it can be accessed on `https://github.com/ilanamt/WhoTooPlus`.

We modeled servers and users by creating a `Server` and a `User` class. They all had public attributes that could be accessed by other servers in order to receive information. For example, each user has a public attribute `da_shares` which the $\mathcal{DA}$ servers update with shares they need to send the user.

In order to model operations that servers could compute locally in parallel, we use threads provided by the tqdm library [26]. Because of the Python Global Interpreter Lock (GIL) [4], threads can not really be executed simultaneously. Nevertheless, most operations where actually executed faster when using threads. Even tough some were slower, we used threads for consistency. A possible solution would have been to use multiprocessing, however, the overhead of creating processes was greater than the time gained by using multiprocessing. When all servers have to compute the same value and agree, the computation is only done once.

The prototype consists of classes `ElGamal, SecShare, DistBBS, DistIBE, DIMAC, PrivatePoly, WTSet` (Privacy-Preserving Multisets), which provide all functionalities described in Chapter 2, together with the class `WhoToo` which implements all algorithms described in Figure 6.1 (without updates).

During distributed operations, if any verification fails, an exception is raised indicating the id's of the verifier and the server that sent that verification value.

In order to increase efficiency, we implement SecShare.Mult using Beaver triples [9]. This means the prototype pre-computes and secret shares triples $(\{a\}, \{b\}, \{c\})$ where a and b are chosen uniformly and independently in $\mathbb{Z}_p$ and $c = a \cdot b$. Then, given shared values $\{x\}$ and $\{y\}$, the shares of $z = x \cdot y$ can be calculated almost without interaction.

Let $\mathcal{DA}_\mathrm{i}$ hold shares $x_\mathrm{i}, y_\mathrm{i}, a_\mathrm{i}, b_\mathrm{i}$ and c_i. In order to obtain z_i, it computes the following:

1. $\mathrm{d_i} \leftarrow x_\mathrm{i} - a$
2. $\mathrm{e_i} \leftarrow y_\mathrm{i} - b$
3. Publish $(\mathrm{d_i}, \mathrm{e_i})$
4. $\mathrm{d} \leftarrow \{\mathrm{d}\}$
5. $\mathrm{e} \leftarrow \{\mathrm{e}\}$
6. $z_\mathrm{i} \leftarrow c_\mathrm{i} + x_\mathrm{i} \cdot \mathrm{e} + y_\mathrm{i} \cdot \mathrm{d} - \mathrm{e} \cdot \mathrm{d}$

These triples are computed during initialization, together with a set number of sharings of random values and IBE values $(C_2, C_3, \{\mathrm{dec}\}, \{t\}, \{s * t^{-1}\})$. Calculating this values during initializations allows the process of submitting new accusations to be more efficient.

In `main.py` we show an example of use. Additionally, we provide unit tests for each component of the prototype.

7.2 Efficiency Analysis and Discussion

7.2.1 Theoretical Efficiency

This section analyzes the differences in efficiency between the original protocol and `WhoToo`$^+$. We argue that if there is a backlog of at least 5 unopened accusations, `WhoToo`$^+$ is significantly more efficient than WhoToo.

Let N be the total number of accusations in the system and m the total number of valid users. Notice that all generations of random values can be pre-computed.

Duplicate Revision

In the original protocol, the user did not need to compute any values specifically for this purpose. During verification, the servers had to run Equal N times. This equality testing requires 3 interactive operations. Then, the duplicate revision had a cost of $3N$ online distributed operations. This is the cost of comparing one encoded value per accusation, however, if we assume this process had to be computed for e_s and c_R, the total would increase to $6N$ distributed operations.

The `WhoToo`$^+$ alternative adds distributed operations to the initialization, but it requires only a constant amount of online distributed operations. In `WhoToo`$^+$.Initialize, the $\mathcal{DA}$ must generate a public and distributed private key, which requires 2 distributed operations, and

compute k DIMACs for each user. Running DIMAC.Tag requires six threshold operations. This means we added $5mk + 3$ distributed operations to initialization. However, all of these values can be pre-computed. During verification, the servers need to compute one DIPRF and a DIMAC verification. Each of those requires four distributed operations (where one can be pre-computed). The addition of shares is performed locally. This means that verification requires 2 offline and 6 online distributed operations.

Matching accusations

Just as in the duplicate revision, the original protocol did not need to compute any distributed operations for matching accusations, yet run Equal N times during WhoToo.OpenAccusations, resulting in $3N$ online threshold operations.

On the other hand, similarly to the duplicate revision, WhoToo$^+$ proposes an alternative that increases the number of interactive operations during initialization and accusation submission, but requires only a constant number of online distributed operations.

During initialization, the $\mathcal{DA}$ servers have to run DistIBE.Setup to create a public and distributed private key, which requires 2 offline interactive operations.

For the following analysis, observe that SecShare.MultExp requires 2 offline and 4 online interactive operations which entails that DistIBE.Enc requires 9 offline and 13 online distributed operations. Furthermore, DistIBE.KeyGen requires 1 online and 1 offline operation.

Consequently, during WhoToo$^+$.Accuse the servers perform 9 offline and 13 online operations when they encrypt for identity s and during the execution of WhoToo$^+$.OpenAccusations they compute 1 online and 1 offline operation to generate the key for identity s.

Set.Quorum

Even though Set.Quorum was modified, the number of distributed operations does not change for this component. In both WhoToo and WhoToo$^+$ the servers compute $n + q + 5$ offline and $n+2$ online interactive operations. The offline operations come from random generations and computations of random values. Online operations come from the n exponentiations needed to evaluate the polynomial $(F \cdot R)^{(q)}$ on s together with the multiplication of s by a random t and the reconstruction of that product.

General Results

Table 7.1 presents the main results, showing the total number of distributed operations required for duplicate revision (DR) and matching accusations (MA) in both WhoToo and WhoToo$^+$.

Even though there is a big increase in the number of offline distributed operations, they can be pre-computed and do not affect scalability. If there is a backlog of at least 3 unopened accusations, WhoToo$^+$ is more efficient than WhoToo. At the same time, if we consider only online operations for MA, we can see that if there is a backlog of at least 5 unopened accusations, WhoToo$^+$ is more efficient, while RD is more efficient with a backlog of 2 unopened operations.

	RD Offline	RD Online	MA Offline	MA Online	Quorum Offline	Quorum Online	Total Offline	Total Online
WhoToo	0	$6N$	0	$3N$	$N+q+5$	$N+2$	$N+2$	$10N+2$
WhoToo+	$5(mk+1)$	6	12	14	$N+q+5$	$N+2$	$5mk+N$ $+q+22$	$N+22$

Table 7.1: Efficiency comparison given a total number N of accusations, m valid users, a maximum number k of accusations per user and quorum q.

In Figure 7.1, we can see the comparison for the number of online interactive operations for finding matching accusations (MA) and duplicate revision (RD) in the original protocol and in `WhoToo`$^+$. Using distributed IBE and DIPRF as described in Chapter 5 results in having a constant number (20) online distributed operations, whereas WhoToo required $9N$ online distributed operations for this processes.

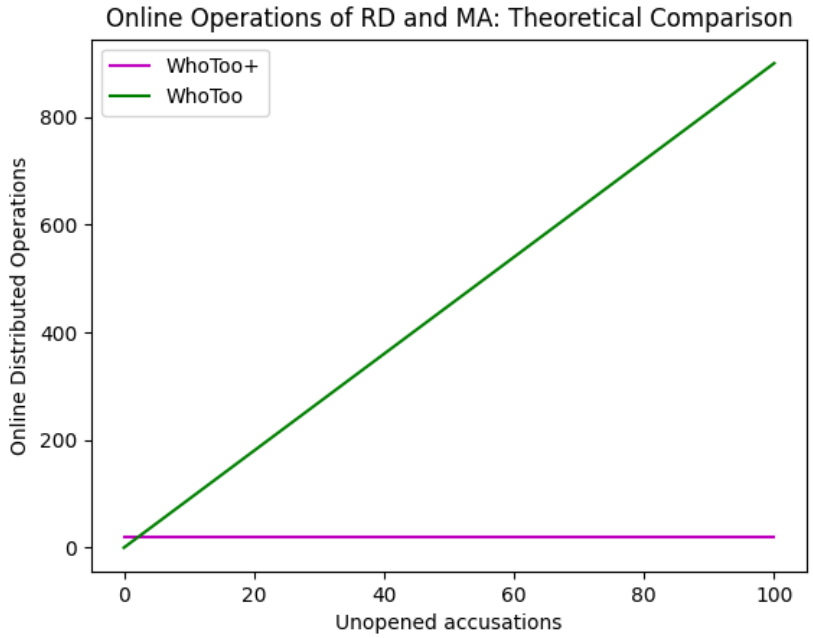

Figure 7.1: Efficiency comparison for RD and MA Online Operations

Figure 7.2 shows the theoretical difference of the total number of required online operations. Notice that even though both protocols are proportional to the number of unopened accusations, WhoToo becomes significantly more inefficient as the backlog increases.

7.2.2 Experimental Efficiency

In this section we show results for the experimental efficiency measured using the prototype described in Section 7.1 on an Intel Core i9 CPU and 32 GB of RAM. All of the following data is run for fixed quorum $q = 2$.

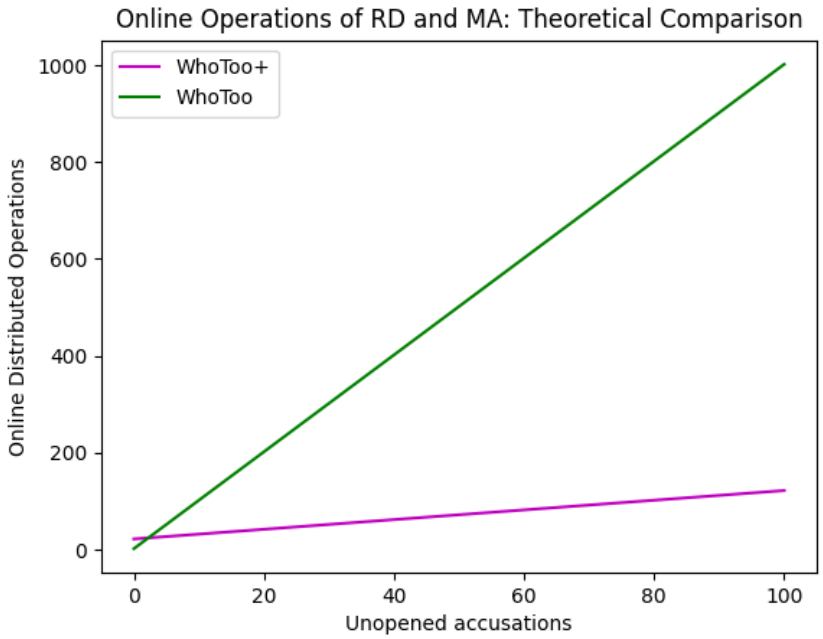

Figure 7.2: Comparison of Total Online Operations

Offline Operations

We measured initialization time, which includes the pre-computation of all MAC values, |ValidAccusers|/4 random values and |ValidAccusers|/4 IBE values, this is, shares of dec, t and $s \cdot t^{-1}$ as well as $C2$ and $C3$.

Figure 7.3 shows the initialization time for different numbers of users with fixed number of MACs per user $k = 3$. As expected, the measured time is proportional to the number of users. This is consistent with the theoretical efficiency shown in Table 7.1.

For all following experiments, we fix the number of users (|ValidAccusers|) to 100. We also measured initialization time depending on the number of MACs per user. This results are shown in Figure 7.4. Again, as expected, initialization time is proportional to the number of MACs per user (k), which define the maximum number of accusations each user can make.

Although initialization takes time, it only need to be run once. At the Faculty of Physical and Mathematical Sciences (FCFM) of Universidad de Chile, there are approximately 5000 students. If we follow the linear interpolation of Figure 7.3, we can predict that initialization for FCFM for would take approximately 3.5 hours. We believe this is a reasonable amount of time for a practical solution.

Online Operations

We now present experimental results of efficiency for online operations. We measure the time taken to submit and process a new accusation. Figure 7.5 shows the measured time for accusations that reach the quorum and those who do not reach it. Both results depend on the number of accusations in the system as expected. Accusations that reach the quorum

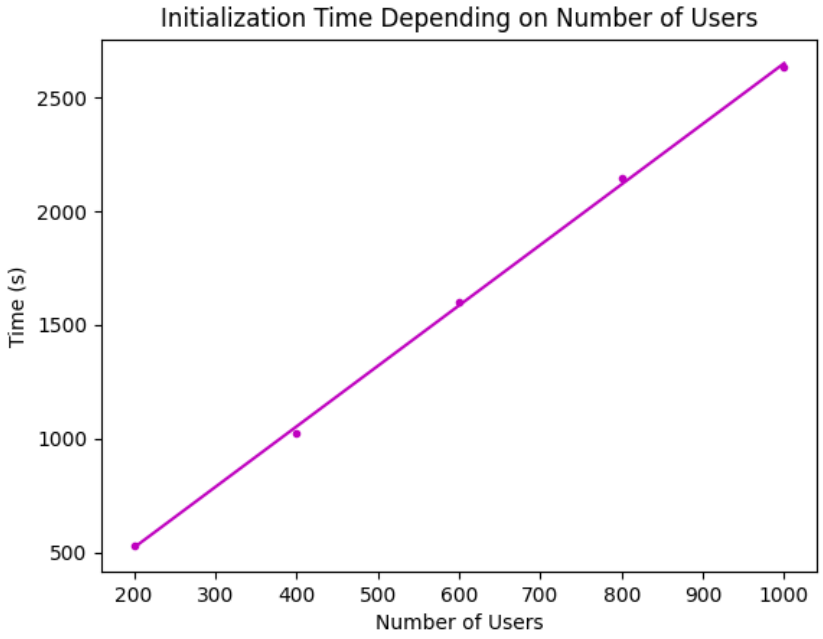

Figure 7.3: Initialization Time for Different Numbers of Users

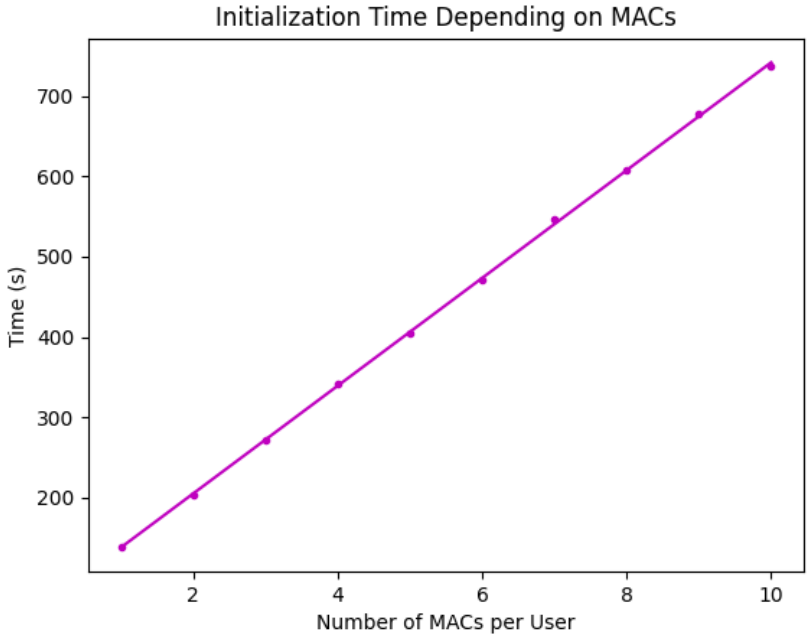

Figure 7.4: Initialization Time for Different Numbers of MACs per User

take longer because the servers need to find which accusations are the ones that match (MA). Even though WhoToo$^+$ requires a constant number of online distributed operations, we observe that both lines are not parallel, which means that MA takes longer if there is a bigger backlog of unopened accusations. This is probably due to the fact that even though servers can locally decrypt IBE values to find matching accusations (and therefore do it in parallel), there are still more values to decrypt if there are more accusations in the system. Moreover, as explained in Section 7.1, our prototype is not really able to run all local operations simultaneously. Nevertheless, the difference in slope is not significant when compared to the original WhoToo protocol, which will be argued in what follows.

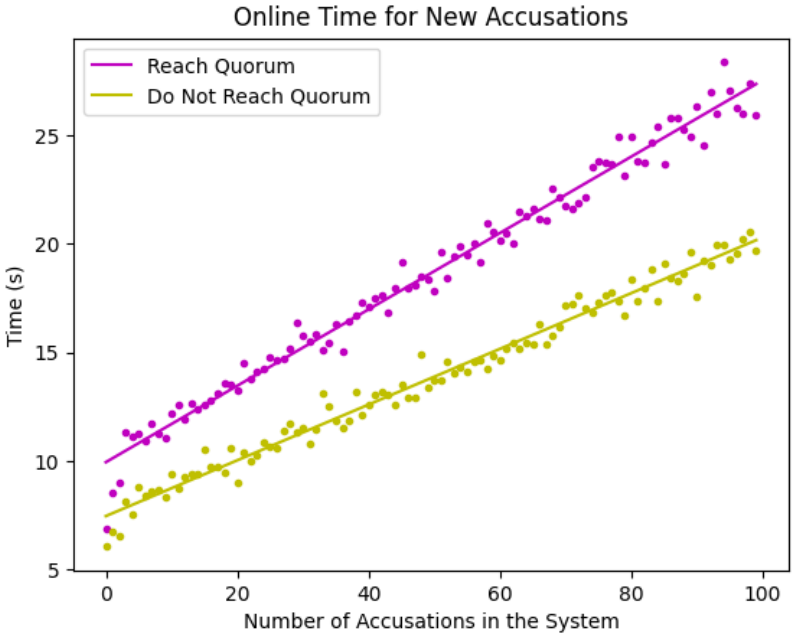

Figure 7.5: Online Time for Submission of New Accusations in WhoToo$^+$

In order to compare the efficiency of WhoToo$^+$ with WhoToo, we simulated the efficiency of the original protocol by removing the DIPRF, DIMAC and IBE schemes and adding N distributed decryptions when adding a new accusation and N distributed decryptions when the quorum was reached, where N is the number of accusations in the system. Therefore, the times measured for the WhoToo protocol are not exact but the data shown can be considered a lower bound for WhoToo efficiency, as we are not running the full protocol, yet all computations would be required in an implementation of WhoToo.

In Figure 7.6 we can observe the difference in time taken to submit a new accusation when that accusation does not reach the quorum depending on the number of accusations in the system. The slope of WhoToo is sigificantly greater than the one of WhoToo$^+$, which reflects the efficiency gained by using the DIPRF for duplicate revision. When there is a small number of accusations, WhoToo is more efficient because comparing the new accusation to all previous ones is still faster than verifying the DIMAC and computing the DIPRF. If the backlog of unopened accusations is 10 or greater, WhoToo$^+$ is more efficient and the difference increases with the number of accusations in the system. This confirms our theoretical efficiency analysis with a slight difference. In the previous section, we argued that our protocol

is more efficient with only 5 unopened accusations (instead of 10), however, the prototype does not pre-compute all possible values, it only calculates random values and IBE values during initialization, which could explain this difference.

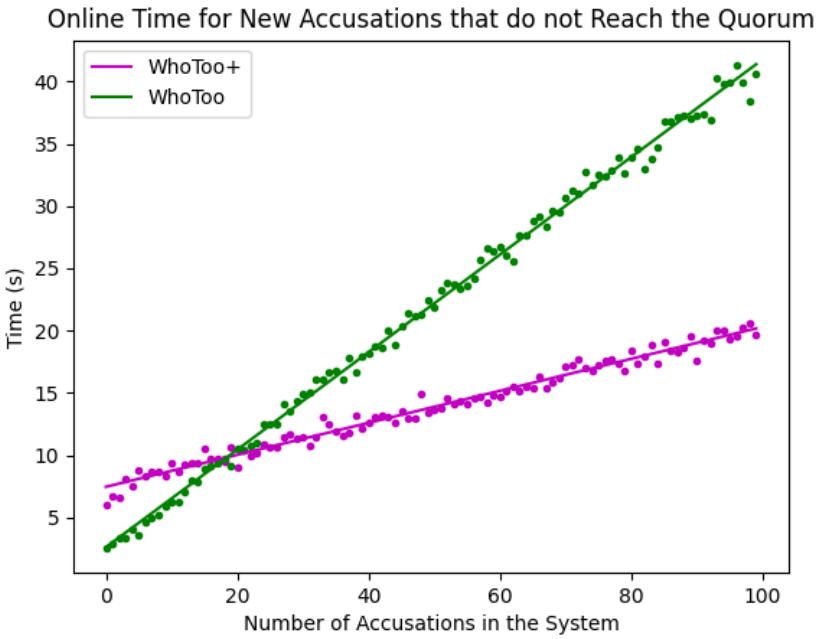

Figure 7.6: Online Time for Submission of New Accusations that Don't Reach the Quorum

Figure 7.7 shows the differences when the new accusations reach the quorum. This results reflect the efficiency gained by both of our improvements: DIPRF for duplicate revision and IBE for finding matching accusations. This results are very similar to the case of accusations that do not reach the quorum, but with a greater difference between the slopes. This makes sense because having both improvements causes a greater efficiency improvement.

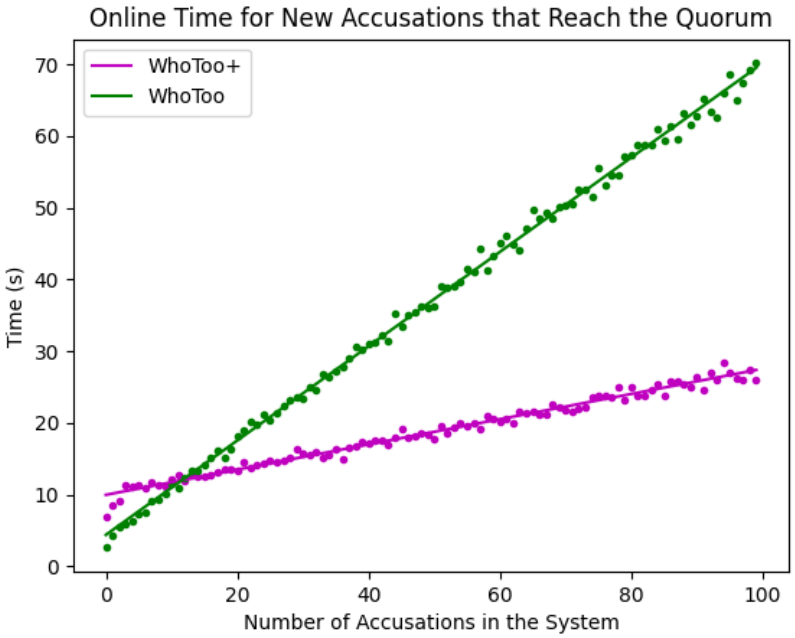

Figure 7.7: Online Time for Submission of New Accusations that Reach the Quorum

Finally, Figure 7.8 shows times measured for new accusations in both cases for each protocol. Notice that the pink and yellow lines, which represent `WhoToo`$^+$ accusations for each case, are almost parallel. Their slopes are appreciably more similar than the green and blue lines, which represent WhoToo accusations. This reflects the fact that using IBE for finding matching accusations requires a constant number of distributed operations, whereas WhoToo requires O(N). Even though the time needed for a new accusation depends on the number of accusations in the system for both WhoToo and `WhoToo`$^+$, the second protocol is significantly more efficient for all kinds of accusations. We believe that the approximately 30 seconds taken to process the 100th accusation is a reasonable amount of time for a real world implementation.

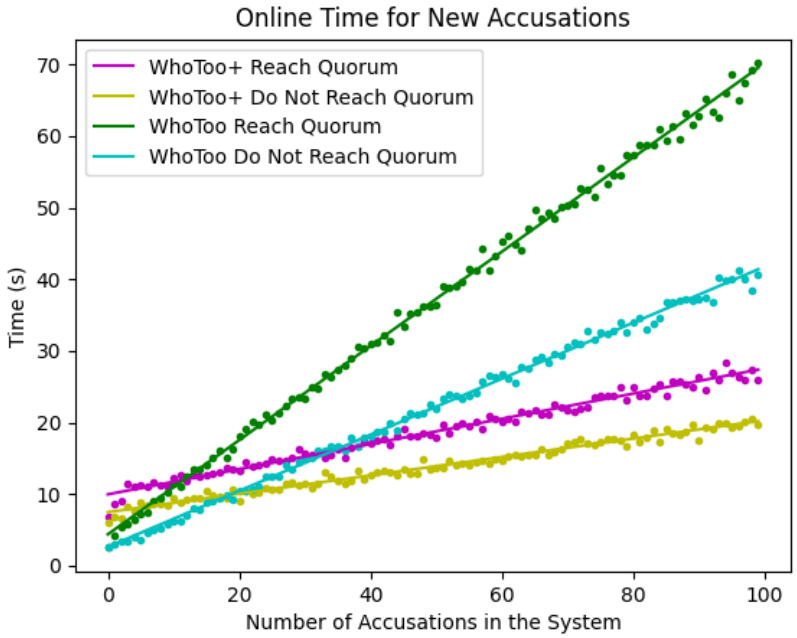

Figure 7.8: Comparison of Online Time for Submission of New Accusations

Chapter 8

Concluding Remarks and Future Work

8.1 Concluding Remarks

The design of a protocol for sexual assault reporting needs to have its users in mind. The process of reporting is extremely difficult and it is much easier to address the problem only from an engineering perspective, adding as many functionalities and options as possible. However, a protocol that does not consider the opinions of experts and potential users might not be a solution at all.

We concluded that a protocol of this kind needs to be distributed and to guarantee secrecy, anonymity, accountability and meta-data hiding. Furthermore, victims have to be able to choose if they want to be contacted for further investigation and they need to be able to change their minds.

People will only use a system they can trust. We realized that trust comes not only from hiding private information, but also from publishing periodic statistics that do not reveal sensitive data.

We designed `WhoToo`$^+$ with all of these ideas in mind. We fixed the gaps in the original WhoToo description and improved its efficiency. Using a sequence of games, we proved that this protocol does not reveal any more information than its ideal functionality.

Finally, we built an open source prototype, which can be used as a starting point for a real life implementation of `WhoToo`$^+$.

8.2 Future Work

As discussed in Section 5.1 having different matching criteria could be useful in contexts where victims do not necessarily know the identity of the person they are accusing. There could be different matching criteria depending on the information victims can provide.

Additionally, it would be beneficial to propose an extension of `WhoToo`$^+$ that allows to specify the quorum for each accusation (within a limited range, as discussed in Section 5.2)

without the need for a separate polynomial for each quorum.

An important contribution of future work would be to fully implement and apply WhoToo$^+$. As mentioned in Section 7.1, it is necessary to implement secure channels for real Servers and Users. A real world implementation should also add verifiability to all implemented threshold operations. Most of these verifications are included in the prototype, however, operations like multiplication and inversion should use the verification values of the original shares to provide verification for the resulting shares. In the prototype, if some verification fails, the protocol halts. In a real world implementation, if a majority of servers agree that some specific server is not behaving as it should, they should be able to remove that server and continue with the rest of the protocol. Finally, opened accusations should be removed from the Accusations set (not from the polynomial), so that they can be directly compared to new open accusations without repeating the distributed decryption process.

Chapter 9

Bibliography

[1] M. Abdalla, M. Bellare, and P. Rogaway. The Oracle Diffie–Hellman Assumptions and an Analysis of DHIES. In CT-RSA 2001, page 143–158. Springer, 2001.

[2] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions. In CRYPTO 2005, pages 205–222. Springer, 2005.

[3] Michel Abdalla, Mihir Bellare, and Gregory Neven. Robust Encryption. Journal of Cryptology, 31(2):307–350, 2018.

[4] Abhinav Ajitsaria. What is the python global interpreter lock (gil)? Last accessed 11 June 2021.

[5] Venkat Arun, Aniket Kate, Deepak Garg, Peter Druschel, and Bobby Bhattacharjee. Finding Safety in Numbers with Secure Allegation Escrows. In NDSS 2020. The Internet Society, 2020.

[6] Karim Baghery. CO6GC: Introduction to Zero-Knowledge Proofs (Part 1). In Cosic Guide To Crypto, KU Leuven, 2020.

[7] J. Bar-Ilan and D. Beaver. Non-Cryptographic Fault-Tolerant Computing in Constant Number of Rounds of Interaction. In PODC'89, page 201–209. ACM, 1989.

[8] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. In SAC'05, page 319–331. Springer, 2005.

[9] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, Advances in Cryptology — CRYPTO '91, pages 420–432, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[10] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on

General Assumptions. In EUROCRYPT 2003, LNCS, pages 614–629. Springer, 2003.

[11] Mihir Bellare and Phillip Rogaway. Introduction to Modern Cryptography. In UCSD CSE 207 Course Notes, page 207, 2005.

[12] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of Group Signatures: The Case of Dynamic Groups. In CT-RSA, 2004.

[13] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In STOC'88, page 1–10. ACM, 1988.

[14] Johannes Blömer, Jakob Juhnke, and Nils Löken. Short Group Signatures with Distributed Traceability. In the 6th International Conference on Mathematical Aspects of Computer and Information Sciences, volume 9582, page 166–180. Springer, 2015.

[15] Dan Boneh and Xavier Boyen. Efficient Selective-ID Secure Identity-Based Encryption Without Random Oracles. In EUROCRYPT 2004, pages 223–238. Springer, 2004.

[16] Dan Boneh and Xavier Boyen. Short Signatures Without Random Oracles. In Advances in Cryptology - EUROCRYPT 2004, pages 56–73. Springer, 2004.

[17] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short Group Signatures. In CRYPTO 2004, pages 41–55. Springer, 2004.

[18] Sean Bowe. BLS12-381: New zk-SNARK Elliptic Curve Construction, 2018. `https://z.cash/blog/new-snark-curve`.

[19] Callisto Homepage. `https://www.mycallisto.org/`. Last accessed 10 Mar 2021.

[20] Ran Canetti. Universally Composable Security. J. ACM, 67(5), September 2020.

[21] D. Cantor, B. Fischer, S. Chibnall, S. Harps, R. Townsend, G. Thomas, H. Lee, V. Kranz, R. Herbision, and K. Madden. Report on the AAU Campus Climate Survey on Sexual Assault and Misconduct. Westat for the Association of American Universities (AAU), 2020.

[22] Charm: A tool for rapid cryptographic prototyping. `http://charm-crypto.io/`. Last accessed 15 Mar 2021.

[23] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, Advances in Cryptology — CRYPTO' 92, pages 89–105, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[24] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults (Extended Abstract). In FOCS, pages 383–395. IEEE Computer Society, 1985.

[25] Sandro Coretti, Ueli Maurer, and Björn Tackmann. Constructing Confidential Channels

from Authenticated Channels—Public-Key Encryption revisited. In Kazue Sako and Palash Sarkar, editors, Advances in Cryptology—ASIACRYPT 2013, volume 8269 of LNCS, pages 134–153. Springer, 12 2013.

[26] Casper da Costa-Luis, Stephen Karl Larroque, Kyle Altendorf, Hadrien Mary, richardsheridan, Mikhail Korobov, Noam Yorav-Raphael, Ivan Ivanov, Marcel Bargull, Nishant Rodrigues, Guangshuo CHEN, Antony Lee, Charles Newey, James, Joshua Coales, Martin Zugnoni, Matthew D. Pagel, mjstevens777, Mikhail Dektyarev, Alex Rothberg, Alexander, Daniel Panteleit, Fabian Dill, FichteFoll, Gregor Sturm, HeoHeo, Hugo van Kemenade, Jack McCracken, Max Nordlund, and Nikolay Nechaev. tqdm: A fast, Extensible Progress Bar for Python and CLI, June 2021.

[27] Yevgeniy Dodis and Aleksandr Yampolskiy. A Verifiable Random Function with Short Proofs and Keys. In Serge Vaudenay, editor, PKC 2005, pages 416–431. Springer, 2005.

[28] Rafael Dowsley, Jörn Müller-Quade, Akira Otsuka, Goichiro Hanaoka, Hideki Imai, and Anderson Nascimento. Universally composable and statistically secure verifiable secret sharing scheme based on pre-distributed data. IEICE Transactions, 94-A:725–734, 01 2011.

[29] Taher ElGamal. A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms. IEEE Transactions on Information Theory, 31(4):469–472, 1985.

[30] Taher ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In George Robert Blakley and David Chaum, editors, CRYPTO 1985, pages 10–18. Springer, 1985.

[31] Amos Fiat and Adi Shamir. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In CRYPTO' 86, pages 186–194. Springer, 1987.

[32] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust Threshold DSS Signatures. In EUROCRYPT '96, pages 354–371. Springer, 1996.

[33] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. J. of Cryptology, 20:51–83, 2006.

[34] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and Fast-Track Multiparty Computations with Applications to Threshold Cryptography. In PODC '98, pages 101–111. ACM, 1998.

[35] Craig Gentry. Practical Identity-Based Encryption Without Random Oracles. In EUROCRYPT 2006, pages 445–464. Springer, 2006.

[36] Jens Groth. Short Non-interactive Zero-Knowledge Proofs. In Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, volume 6477 of LNCS, pages 341–358. Springer, 2010.

[37] Alejandro Hevia and Ilana Mergudich-Thal. Implementing Secure Reporting of Sexual

Misconduct - Revisiting WhoToo. In Patrick Longa and Carla Ràfols, editors, Progress in Cryptology - LATINCRYPT 2021 - 7th International Conference on Cryptology and Information Security in Latin America, Bogotá, Colombia, October 6-8, 2021, Proceedings, volume 12912 of LNCS, pages 341–362. Springer, 2021.

[38] Alejandro Hevia and Daniele Micciancio. An Indistinguishability-Based Characterization of Anonymous Channels. In Proceedings of the 8th International Symposium on Privacy Enhancing Technologies, PETS '08, page 24–43. Springer, 2008.

[39] María Jesús Ibáñez. Universidad de Chile presenta primeros resultados de estudio de acoso sexual. https://www.uchile.cl/noticias/124410/u-de-chile-presenta-primeros-resultados-de-estudio-de-acoso-sexual. Last accessed 18 May 2020.

[40] Markus Jakobsson and Ari Juels. Mix and Match: Secure Function Evaluation via Ciphertexts. In Tatsuaki Okamoto, editor, Advances in Cryptology — ASIACRYPT 2000, pages 162–177. Springer, 2000.

[41] Jonathan Katz and Yehuda Lindell. Introduction to Modern Cryptography, Second Edition. Chapman & Hall/CRC, 2nd edition, 2014.

[42] Lea Kissner and Dawn Song. Privacy-Preserving Set Operations. In CRYPTO 2005, pages 241–257. Springer, 2005.

[43] Benjamin Kuykendall, Hugo Krawczyk, and Tal Rabin. Cryptography for #MeToo. POPETS, 2019(3):409–429, 2019.

[44] Yehuda Lindell. How to Simulate It – A Tutorial on the Simulation Proof Technique, pages 277–346. Springer International Publishing, Cham, 2017.

[45] Yehuda Lindell. Zero-knowledge proofs of knowledge slides, 2018. Center for Research in Applied Cryptography and Cyber-Security, Bar-Ilan University.

[46] A. Lizama-Lefno and A. Hurtado-Quiñones. Acoso Sexual en el Contexto Universitario: Estudio Diagnóstico Proyectivo de la Situación de Género en la Universidad de Santiago de Chile 2019. Pensamiento Educativo. Revista de Investigación Educacional Latinoamericana, pages 1–14, 2019.

[47] Ben Lynn. Notes: Zero knowledge proofs. https://crypto.stanford.edu/pbc/notes/crypto/zk.html. Last accessed 13 June 2021.

[48] Lynn, B. PBC Library. https://crypto.stanford.edu/pbc/, Last accessed May 2021.

[49] Ueli Maurer and Pierre Schmid. A Calculus for Security Bootstrapping in Distributed Systems. Journal of Computer Security, 4(1):55–80, 1996.

[50] R. E. Morgan and B. A. Oudekerk. Criminal Victimization, 2018. Bureau of Justice Statistics, U.S. Department of Justice, 2019.

[51] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed Pseudo-random Functions and KDCs. In EUROCRYPT '99, pages 327–346. Springer, 1999.

[52] National Sexual Violence Resource Center. False Reporting. Last accessed 17 May 2021.

[53] H. Okano, K. Emura, T. Ishibashi, Ohigashi, T., and T. Suzuki. Implementation of a Strongly Robust Identity-Based Encryption Scheme over Type-3 Pairings. IJNC, 10(2):174–188, 2020.

[54] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In Advances in Cryptology — CRYPTO '91, pages 129–140. Springer, 1992.

[55] Project Callisto 2017-2018 Year Report. Last accessed 18 May 2020.

[56] A. Rajan, L. Qin, D. W. Archer, D. Boneh, T. Lepoint, and M. Varia. Callisto: A Cryptographic Approach to Detecting Serial Perpetrators of Sexual Misconduct. COMPASS, 2018:1–4, 2018.

[57] Burton Rosenberg. Probabilistic Polynomial Time Algorithms and Cryptography, 2021. University of Miami CSC507/609-B: Cryptography Course Notes, `https://www.cs.miami.edu/home/burt/learning/csc609.221/`.

[58] C. P. Schnorr. Efficient Signature Generation by Smart Cards. J. Cryptol., 4(3):161–174, 1991.

[59] Adi Shamir. How to Share a Secret. Comm. of the ACM, 22(11):612–613, 1979.

Chapter 10

Appendix

10.1 Focus Groups' Guideline

We now present the guidline used for the Focus Groups with different groups of students at Universidad de Chile.

1. Introducción y explicaciones generales
   (a) Bienvenida, agradecimiento por su tiempo
   "Me ayudan muchísimo con el tiempo que me están dando"
   (b) Presentarme brevemente
   "Ilana, haciendo mi tesis en magíster DCC, en ese marco voy a hacer estar reuniones para que me puedan ayudar con algunos inputs para mi tesis, después les voy a explicar mas en detalle."
   (c) Definir reglas
   "Algunas cosas importantes"
      i. Información es confidencial, sólo será usada para definir ciertas cosas de mi tesis
      ii. Es importante que todas opinen
      "La idea es que todas puedan dar su opinión, es super importante la opinión de todas. Típicamente hay gente más buena para hablar y otra no tanto, ojalá las que son mejores para hablar hagan un esfuerzo por darle un espacio a las demás y las que les cuesta más, ojalá hagan un esfuerzo por ir opinando, para que yo pueda saber que piensa cada una."
      iii. No hay respuestas correctas ni incorrectas
      "La opinión personal y experiencia de cada una es super importante para mi así que siéntanse super libres de decir lo que sientan lo que quieran lo que opinan. A mi me ayudan mucho más si son lo más honestas posibles. A mi no me va a ir mejor ni peor en mi tesis porque a ustedes les gusten las cosas que yo propongo, al contrario, lo que yo necesito es de verdad saber si les parece o no les parece de la manera mas honesta posible, es la mejor manera de ayudarme."

iv. Preguntar por grabación

"El tema que vamos a hablar es delicado y en ese sentido les agradezco un montón la apertura de estar acá y quería preguntarles, a mi me ayudaría muchísimo si pudiera grabara esta reunión porque sino tendría que acordarme o tomar apuntes y eso hace que las pueda escuchar con menos atención. De repente no grabar el zoom pero grabar el puro audio con mi celular, y les prometo que después solo lo voy a usar para tomar apuntes y después lo voy a borrar. Si prefieren que no, no lo voy a grabar. Díganme pero si no les importa me harían demasiado mas fácil la pega. Les parece que grabe o no? Voy a grabar entonces desde ahora, desde este minuto"

v. Abrir espacio para preguntas

(d) Presentaciones

"Para partir vamos a pedir que nos presentemos, cada una con su nombre, en que están en la u, con quien viven si quieren, cosas que les gustan, lo que quieran contar, pero cortito. Partamos con x. Para terminar yo soy Ilana como les dije, estoy en 6to año del DCC trabajando en mi tesis, vivo con mi mamá y con mi hermana y con mi perro que amo que se llama Lambda y anda por aquí y amo la computación o bla. Ya, estamos listas entonces, vamos a partir."

2. Acoso en la Universidad

(a) Asosiación espontánea

- Vamos a partir con una especie de juego ya? Yo voy a decir una palabra y ustedes me van a decir todo todo lo que se les venga a la mente cuando yo digo esa palabra. Pueden ser otras palabras, frases, colores, emociones, sensaciones, ideas, lo que sea ya?
- Todas hablen nomas, aquí no hay que pedir la palabra
- Ya, todo lo que se les viene a la mente cuando digo sol
- Super, lo hicieron super bien
- Ahora vamos a hacer lo mismo, todo lo que se les viene a la mente cuando yo digo acoso en la universidad
  - En que sentido esto, en que sentido lo otro
  - Lo que estoy entendiendo es x, les pido que me corrijan si me equivoco

(b) Experiencias Experiencias de ellas o personas cercanas

- ¿Han denunciado o no?
- ¿Por qué han denunciado o no?
- ¿Por qué a veces si y a veces no?
- ¿Cuales son las barreras para denunciar?
- ¿Como tiene que ser el espacio, que condiciones tienen que cumplirse para que uno denuncie?

3. Temas específicos

- Explicar mi idea, contarles de Callisto
- Quórum
  - ¿Les hace sentido?
  - ¿Fijo o variable?

    - Si fijo, ¿cuánto?
    - Si variable, ¿se puede modificar después?
  - Expiración de denuncias
    - ¿Les hace sentido?
    - ¿Fijo o variable?
    - Si fijo, ¿cuánto?
    - Si variable, ¿se puede modificar después?
  - Categorías de denuncias
    - Ejemplo: comentarios machistas/inapropiados, acoso sexual, abuso sexual
    - En protocolo de la UChile: acoso sexual, violencia de genero, acoso laboral, discriminación arbitraria
    - ¿Les hace sentido tener categorías?
    - ¿Cómo/cuáles deberían ser estas categorías?
    - ¿Les hacen sentido las categorías del protoolo UChile?
    - ¿Les gustaría poder elegir si el quórum debe juntarse con puras personas de su misma categoría? ¿O esa y "mayores"?
    - ¿Quién define categorías y orden de gravedad? ¿Autoridad universitaria?
    - ¿Que pasa con denuncias anteriores si categorías cambian o orden de gravedad cambia?
  - Meta información
    - ¿Qué información debería ocultarse y qué información no?
    - Total de denuncias
    - Total de denuncias por carrera, por departamento
    - Promedio de quórum escogido
    - Si hay mas de n denuncias contra alguien (¿quórum máximo? ¿en ese caso solo revelar denunciado?)
    - Cantidad de matches parciales
  - Factores de matching
    - ¿En su experiencia o la de personas cercanas, que información conocen la persona a la que denunciarían?
    - ¿Qué les parecería la siguiente propuesta?
      * Match con nombre completo, rut, mail o usuario de redes sociales
      * Match si calzan al menos 3 factores "secundarios" (nombre de pila, apodo, carrera/especialidad, año de ingreso a la universidad)
  - Sistema distribuido: ¿Les da mas confianza que el sistema sea distribuido (ej: un servidor en el cec, uno el DCC, uno en el DSTI)
  - Otros: ¿A alguien se le ocurre cualquier otra cosa de la que no hayamos hablado y pueda servir?
4. Agradecimientos y Cierre