Yale University
Department of Computer Science

Security Analysis of Accountable Anonymous Group Communication in Dissent

Ewa Syta Aaron Johnson Henry Corrigan-Gibbs
Shu-Chun Weng David Wolinsky Bryan Ford

YALEU/DCS/TR-1472
January 31, 2013

Abstract

Users often wish to communicate anonymously on the Internet using, for instance, group discussion forums or instant messaging. Misbehaving users may abuse this anonymity to disrupt communication, however, and existing solutions do not adequately address this risk. Messaging protocols such as DC-nets leave groups vulnerable to denial-of-service and Sybil attacks, mix-nets are difficult to protect against traffic analysis, and accountable voting protocols are unsuited to general anonymous messaging.

DISSENT, originally introduced by Corrigan-Gibbs and Ford (2010), is the first general communication protocol that offers provable anonymity, integrity and accountability for moderate-size groups, and efficiently handles unbalanced loads where few members wish to transmit in a given round. We provide a full description of an improved DISSENT protocol, define its precise security properties, and give rigorous proofs of these properties. Our improved protocol is a direct result of this security analysis, which identified several non-trivial attacks on the original protocol stemming from subtle design flaws.

Security Analysis of Accountable Anonymous Group Communication in Dissent*

Ewa Syta† Aaron Johnson‡ Henry Corrigan-Gibbs† Shu-Chun Weng†
David Wolinsky† Bryan Ford†

1 Introduction

Anonymous participation is often considered a basic right in free societies (Yale Law Journal 1961). The limited form of anonymity the Internet provides is a widely cherished feature (Teich, Frankel, Kling, and Lee 1999; Wallace 1999), enabling people and groups with controversial or unpopular views to communicate and organize without fear of personal reprisal (Stein 2003). Yet anonymity makes it difficult to trace or exclude misbehaving participants (Davenport 2002). Online protocols providing stronger anonymity, such as mix-networks (Chaum 1981; Adida 2006), onion routing (Goldschlag, Reed, and Syverson 1999; Dingledine, Mathewson, and Syverson 2004), and Dining Cryptographers Networks or DC-nets (Chaum 1988; Waidner and Pfitzmann 1989; Sirer et al. 2004; Golle and Juels 2004), further weaken accountability, yielding forums in which no content may be considered trustworthy and no reliable defense is available against anonymous misbehavior.

DISSENT (Dining-cryptographers Shuffled-Send Network) is a communication protocol that provides strong integrity, accountability and anonymity. Members of small, private online groups, whose membership is closed and known to its members, are able to send *anonymous* messages to each other, to the whole group, or to a non-member, in that the receiver knows that *some* member sent the message, but no one knows *which* member. DISSENT holds members *accountable*, not by compromising their anonymity but rather by ensuring that communication resources are allocated among all communicating members, and that any disruption results in the identification of some malicious member during a “blame” process. Members are thus unable to corrupt or block other members’ messages, overrun the group with spam, stuff ballots, or create unlimited anonymous Sybil identities (Douceur 2002) or sock puppets (Stone and Richtel 2007) with which to bias or subvert the group’s deliberations.

DISSENT builds on the shuffle of Brickell and Shmatikov (2006a), combining that with DC-net techniques for efficient bulk communication. It uses only readily available cryptographic primitives and handles arbitrarily large messages and unbalanced loads efficiently. Each member sends

*The work of Ewa Syta, Henry Corrigan-Gibbs, Shu-Chun Weng, David Wolinsky, and Bryan Ford (email: ewa.syta@yale.edu) was supported by the Defense Advanced Research Projects Agency (DARPA) and SPAWAR Systems Center Pacific, Contract No. N66001-11-C-4018. Aaron Johnson (email: aaron.m.johnson@nrl.navy.mil) was supported by DARPA. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA or SPAWAR.

†Department of Computer Science, Yale University, CT

‡U.S. Naval Research Laboratory

exactly one message per round, making it usable for voting or assigning pseudonyms with a 1-to-1 correspondence to real group members. DISSENT has limitations, of course. It is not intended for large-scale, "open-access" anonymous messaging or file sharing (Goldschlag, Reed, and Syverson 1999; Clarke, Sandberg, Wiley, and Hong 2000). DISSENT's accountability property assumes closed groups, and may be ineffective if a malicious member can leave and rejoin the group under a new (public) identity. Finally, DISSENT's serialized GMP-SHUFFLE protocol imposes a per-round startup delay that makes DISSENT impractical for latency-sensitive applications. Further discussion on related anonymous communication systems is included in Section 6.

DISSENT was first introduced by Corrigan-Gibbs and Ford (2010). In addition to sketching the protocol and security arguments, they describe practical usage considerations and give the results of several performance experiments based on a prototype implementation. We focus here on a detailed exposition of DISSENT and a rigorous analysis of its security properties.

Indeed, during our analysis of the original protocol, we identified several attacks. For example, anonymity could be broken by replaying protocol inputs in subsequent rounds, by providing at certain points incorrect ciphertexts to some members and correct ones to the rest, or by copying ciphertexts at other points. Accountability for disruption could be avoided by copying the protocol inputs from honest members, and honest members could potentially be falsely accused of disruption by rearranging valid signed messages to create phony logs. Protocol termination could be prevented for some members by causing failures for them while allowing the rest to terminate successfully and thus not participate in a blame process. See the appendix more details of these attacks.

In order to fix these flaws, we made several non-trivial modifications to the original protocol. To prevent replay attacks we added key generation steps. To prevent equivocation attacks we added rebroadcast steps, and have members intentionally cause intermediate protocol failures when equivocation is observed. We add the use of non-malleable commitments to prevent submission duplication, and we add phase numbers to prevent log forgery. Finally, to prevent non-termination of the protocol, we make all steps non-optional, in particular including an opportunity for blame at the end of every execution to ensure accountability.

We are able to give proofs of security for this improved protocol. In particular, we provide rigorous proofs of integrity, accountability, and anonymity. Obtaining a fully secure protocol with proofs required a surprising amount of additional work given the relative simplicity and maturity of the underlying ideas. However, as observed by Wikström (2004), the complexity of anonymous communication protocols has frequently resulted in incomplete proofs and subtle errors (see further discussion in Section 6).

The main contributions of this paper, therefore, are (1) we provide a full description of an improved DISSENT protocol, (2) we present precise definitions of its security properties, and (3) we give rigorous proofs that the protocol satisfies those definitions.

Section 2 outlines DISSENT's framework and security model. Section 3 describes the GMP-SHUFFLE protocol, and Section 4 details the GMP-BULK transfer protocol. Section 5 provides formal security properties and their proofs. Section 6 summarizes related work, and Section 7 concludes.

2 Protocol Overview

DISSENT is designed to be used in a group setting. Each member i of the group is associated with a long term public signing key pair (u_i, v_i). DISSENT provides a *shuffled send* communication primitive that gives sender anonymity among that group. During each protocol run, every group member i secretly creates a message m_i and submits it to the protocol. The protocol effectively collects all secret messages, shuffles their order according to some random permutation π that *no one* knows, and broadcasts the resulting sequence of messages. Each input message m_i can have a different length L_i.

We present a messaging interface, called the General Messaging Protocol, that DISSENT implements. DISSENT in fact defines two protocols implementing this interface: the GMP-SHUFFLE protocol provides anonymous communication for fixed-length messages, and the GMP-BULK protocol builds on this to provide efficient anonymous communication of arbitrary-length messages.

2.1 The General Messaging Protocol

A Group Messaging Protocol GMP is a 3-tuple of algorithms SETUP(v_i), ANONYMIZE(m_i, K, n_R, τ, f) and VERIFY-PROOF(p_j, ℓ_i).

SETUP takes a member's public signing key v_i as input and outputs one or more session nonces n_R, a set K of all members' signing keys, an ordering of members τ, and optionally a message length L. All group members run the SETUP algorithm before each protocol run to agree on common parameters. Such agreement might be achieved via Paxos (Lamport 1998) or BFT (Castro and Liskov 1999). We emphasize that SETUP does not generate members' signing keys; rather, it uses long term signing keys submitted by each member.

ANONYMIZE takes a message m_i, a set K of members' signing keys, one or more round nonces n_R, an ordering of members τ, and optionally a flag f as input, and outputs either $(\text{SUCCESS}, M_i')$, where M_i' is a set of messages, or $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$, where BLAME_i is a set of observed misbehaviors, and ℓ_i is a log of a protocol run. After agreeing on common parameters, the group runs the ANONYMIZE algorithm. The goal of the algorithm is to anonymously broadcast the set of messages submitted by group members. If a protocol run succeeds for a member, then she outputs the anonymized messages. Otherwise, the protocol run fails and the group member produces a set of blame proof(s) for the member misbehavior(s) responsible for the protocol run failure.

VERIFY-PROOF takes a proof p_j of member j's misbehavior and a log ℓ_i as input, and outputs either TRUE indicating that p_j is indeed a proof of j's misbehavior given the observed protocol history represented by log ℓ_i, or FALSE otherwise. If a run of ANONYMIZE fails for member i, then i *blames* at least one dishonest member j by producing a proof p_j of j's misbehavior and a log ℓ_i. VERIFY-PROOF is used to verify that proof p_j does in fact indicate misbehavior by j given ℓ_i.

2.2 The GMP-Shuffle Protocol

The GMP-SHUFFLE protocol enables the anonymous exchange of equally sized messages. However, it incurs extra communication if only one member wishes to send, and its decrypt-and-shuffle phase is inherently serial. GMP-SHUFFLE builds on a data mining protocol by Brickell and Shmatikov (Brickell and Shmatikov 2006b) to broadcast the input set of fixed-length messages, one from each group member, in an unknown permutation, providing cryptographically strong anonymity. Like many anonymous messaging protocols, the original data mining protocol was vulnerable to untraceable denial-of-service (DoS) attacks by malicious group members. We remove this vulnerability by adding *go/no-go* and *blame* phases, which can trace and hold accountable any group member maliciously disrupting the protocol.

In the GMP-SHUFFLE protocol, all members $1, \dots, N$ choose their secret messages $m_1, \dots, m_N$ of equal length L. Each member has a long lived signing key pair (u_i, v_i) and knows the ordering of the group and a session nonce n_R. For a single run of the protocol, each member generates two key pairs, called inner and outer, and shares the public keys with the group. Each member i iteratively encrypts its message m_i using all inner and then all outer public keys. The resulting ciphertext messages are sent to a group leader who strips one layer of encryption from each ciphertext using his outer public key, permutes the messages, and forwards the permuted set to the next member who repeats the process. Removing all layers of outer encryption yields a set of inner ciphertext messages which member N broadcasts to the entire group. All members inspect the set to verify that their inner ciphertext is present. If all members' messages are included and every step of the protocol completes successfully, each member releases its inner private key allowing the set of permuted secret messages to be recovered. If any inner ciphertext is missing or corrupted, however, the inner keys are destroyed and the entire group enters a *blame* phase to find the culprit member(s).

Section 3 details the GMP-SHUFFLE protocol and Section 5 demonstrates its security.

2.3 The GMP-Bulk Protocol

The GMP-BULK protocol uses ideas from DC-nets (Chaum 1988; Waidner and Pfitzmann 1989; Sirer et al. 2004; Golle and Juels 2004) to anonymously transmit variable-length messages. In place of the DoS-prone slot reservation systems in prior DC-nets schemes, however, DISSENT leverages its GMP-SHUFFLE protocol to prearrange the DC-nets transmission schedule, guaranteeing each member exactly one message slot per round.

GMP-BULK uses the GMP-SHUFFLE protocol to broadcast an unknown permutation of the *message descriptors* submitted by each member. Each descriptor d_i contains the length L_i of member i's message m_i, a cryptographic hash of m_i, a vector $\vec{S}_i$ of seeds s_{ij}, where each seed is encrypted with j's session public key and assigns each member j a pseudorandom bulk ciphertext to transmit, and a vector $\vec{H}_i$ of hashes H_{ij} validating each bulk ciphertext. The shuffled order of the message descriptors indicates the order in which the anonymous senders should transmit their secret messages. Then, all group members broadcast bit streams based on pseudorandom seeds included in the message descriptors, so that XORing all members' bit streams together yields a permuted set of all members' variable-length messages. During a member's own transmission slot, he transmits his own message XOR'd with the messages he has instructed all other members to generate. During another group member's transmission slot, members broadcast a pseudorandom bit string generated from an encrypted seed in the slot's message descriptor. Cryptographic hashes in the message descriptors enable members to verify the correctness of each others' bulk transmissions, ensuring message integrity and DoS protection throughout. If any group member sends an invalid bit string, then in a blame phase the owner of that transmission slot uses GMP-SHUFFLE to anonymously broadcast an *accusation* exposing the faulty group member. The GMP-BULK protocol is detailed in Section 4 and Section 5 proves its security.

2.4 Security Model

We assume the adversary is polynomial-time limited. We allow him to control a colluding subset of group members. We define the rest of the members as *honest*, in that they run the prescribed algorithms, and their internal states are hidden from the adversary. We assume that communication channels exist between all members, and that they can be observed by the adversary.

The security properties we wish the protocol to satisfy are *integrity*, *accountability*, and *anonymity*, as we describe below. Formal definitions of these properties and their proofs are given in Section 5.

- **Integrity:** The protocol offers *integrity* if every honest member for whom the protocol completes successfully has the same output and receives the messages of all the other honest members.

- **Accountability:** The protocol offers *accountability* if (*i*) every honest member for whom the protocol failed obtains proof of some member's misbehavior valid under VERIFY-PROOF, and (*ii*) the adversary cannot produce a valid proof of misbehavior by an honest member.

- **Anonymity:** The protocol maintains *anonymity* if the adversary can guess the sources of the messages from honest users with probability no greater than random guessing.

We observe that these properties do not imply that DISSENT completes for all members, and, in fact, we cannot guarantee that the protocol terminates if a member stops participating at some point. However, the protocol execution is very simple: a fixed sequence of phases during which all members send no message or all send one message. If a properly signed message indicating the desired protocol run and phase is received from every member, the protocol proceeds to the next round. Therefore every member knows when another should send a message, and thus gossip techniques such as those used in PeerReview (Haeberlen, Kouznetsov, and Druschel 2007) can be applied in a wrapper protocol to ensure liveness. Moreover, we note that when every member follows the protocol, not only does it complete but it succeeds, as will be clear from the protocol description.

2.5 Cryptographic Primitives and Security Assumptions

DISSENT makes use of several cryptographic tools, and its security depends on certain assumptions about their security.

2.5.1 Hash functions

We use a standard definition (Stinson 2005) of a collision-resistant *unkeyed hash function* and will denote the hash of message m as HASH$\{m\}$. We assume that the hash function used is second-preimage resistant (Rogaway and Shrimpton 2004).

Definition 1. *A hash function is second-preimage resistance if it is computationally infeasible to find any second input which has the same output as any specified input, i.e. given x, to find a second pre-image $x' \neq x$ such that $h(x) = h(x')$.*

2.5.2 Encryption

We use a *cryptosystem* that consists of: *(i)* a key generation algorithm taking a security parameter ρ and producing a private/public key pair (x, y); *(ii)* an encryption algorithm taking public key y, plaintext m, and some random bits R, and producing a ciphertext $c = \{m\}_y^R$; *(iii)* a deterministic decryption algorithm taking private key x and ciphertext c, and returning the plaintext m. A member can save the random bits R used during encryption. The notation $c = \{m\}_{y_1:y_N}^{R_1:R_N}$ indicates iterated encryption via multiple keys: $c = \{\dots\{m\}_{y_1}^{R_1}\dots\}_{y_N}^{R_N}$. We omit R when an encryption's random inputs need not be saved.

We assume that members can check an arbitrary (x, y) purported to be a key pair to verify that it could have been generated by the specified key generation algorithm. We also assume that the underlying public-key cryptosystem provides indistinguishable ciphertexts against a chosen-ciphertext attack. That is, the cryptosystem is IND-CCA2 secure (Bellare, Desai, Pointcheval, and Rogaway 1998).

Definition 2. *A cryptosystem is IND-CCA2 if, for all probabilistic polynomial-time adversaries, the advantage in the distinguishing game is negligible as a function of the security parameter ρ.*

The *distinguishing game* (Bellare, Desai, Pointcheval, and Rogaway 1998; Brickell and Shmatikov 2006a) is played between an adversary A and a challenger C who takes as input the challenge bit b.

1. The challenger C uses ρ to generate a key pair (x, y) and gives the public key y to the adversary A.

2. A may encrypt polynomially many messages m using y and decrypt polynomially many arbitrary ciphertexts c. To decrypt a ciphertext $c = \{m\}_y$, A queries c to C, who sends back $m = \{c\}_x$.

3. Eventually, A chooses two messages m_0 and m_1 and sends them to C.

4. C computes $c_b = \{m_b\}_y$ and sends it to A.

5. A may perform polynomially many encryptions of any m, and polynomially many decryptions of any ciphertexts c, provided that $c \neq c_b$.

6. A outputs a guess $\hat{b} \in \{0, 1\}$ for the value of b.

The adversary's *advantage* in the distinguishing game is equal to

$$\left| Pr\left[A^{C(0)} = 1\right] - Pr\left[A^{C(1)} = 1\right]\right|,$$

where the probability is taken over the randomness of the adversary and the challenger.

2.5.3 Digital Signatures

We use a *signature scheme* that consists of: *(i)* a key generation algorithm taking a security parameter ρ and producing a private/public key pair (u, v); *(ii)* a signing algorithm taking private key u and message m to produce signature $\sigma = \text{SIG}_u\{m\}$; and *(iii)* a deterministic verification algorithm taking public key v, message m, and candidate signature σ, and returning true iff σ is a correct signature of m using v's associated private key u. The notation $\{m\}\text{SIG}_u$ indicates the concatenation of message m with the signature $\text{SIG}_u\{m\}$.

We assume that the underlying digital signature scheme has a strong unforgeability property. That is, it is EUF-CMA secure (Goldwasser, Micali, and Rivest 1995).

Definition 3. *A digital signing scheme is EUF-CMA secure if, for all probabilistic polynomial-time adversaries, the adversary's advantage in the forging game is negligible as a function of the security parameter ρ.*

The *forging game* is played between an adversary A and a challenger C. It is equivalent to a standard EUF-CMA game.

1. The challenger C uses ρ to generate a key pair (x, y) and gives the public key y to the adversary A.

2. A may request signatures on polynomially many messages. A chooses a message m and sends it to C, who sends back σ, a signature on m under y. A is allowed to query C in an adaptive fashion.

3. Eventually, A outputs a pair (m', σ').

The adversary wins the forging game if (m', σ') is a valid message- signature pair under y assuming that m' has never been queried to the challenger. The adversary's advantage is simply the probability of winning the forging game, where the probability is taken over the randomness of the adversary and the challenger.

2.5.4 Pseudo-random Number Generator

We use a standard definition (Stinson 2005) of a *pseudorandom number generator* (PRNG). Let $g(s)$ be a pseudo-random number generator, where s is a seed. We will denote the first L bits generated from $g(s)$ as $\text{PRNG}\{L, s\}$.

Definition 4. *A function $g : \{0,1\}^{\ell_1(\rho)} \to \{0,1\}^{\ell_2(\rho)}$ is a pseudorandom number generator if, for all probabilistic polynomial-time adversaries, the adversary's advantage in the pseudorandomness game is negligible as a function of the security parameter ρ.*

The *pseudorandomness game* is played between an adversary A and a challenger $C(b)$.

1. If $b = 0$, C chooses $s \in \{0,1\}^{\ell_1(\rho)}$ uniformly at random and sets $r = g(s)$. If $b = 1$, C chooses $r \in \{0,1\}^{\ell_2(\rho)}$ uniformly at random.

2. C sends r to A.

3. A outputs a guess $\hat{b} \in \{0,1\}$ for the value of b.

The adversary's advantage in the pseudorandomness game is

$$\left| Pr\left[A^{C(0)} = 1\right] - Pr\left[A^{C(1)} = 1\right]\right|,$$

where the probability is taken over the randomness of the adversary and the challenger.

2.5.5 Non-malleable Commitments

We use the definition by Dolev, Dwork, and Naor (2000) of a *non-malleable commitment*. The notation $x = \text{COMMIT}\{c\}$ indicates that x is a commitment to c, and the notation $c = \text{OPEN}\{x\}$ indicates that c is the opening of the commitment x.

3 GMP-Shuffle

3.1 Protocol Description

The Group Messaging Protocol-Shuffle GMP-SHUFFLE is an instantiation of the Group Messaging Protocol and consists of three algorithms: SETUP-S, ANONYMIZE-S, and VERIFY-PROOF-S. Before each protocol run, all members run the SETUP-S algorithm to agree on the common parameters needed for each run. One parameter thus determined is the fixed message length L. Each member i pads or trims input message m_i to length L. All members use the remaining parameters K, n_R, and τ as inputs to ANONYMIZE-S. This algorithm also takes a fail flag f which is always set to FALSE when the algorithm is run as a part of GMP-SHUFFLE. The fail flag will sometimes be set to TRUE when ANONYMIZE-S is run as a part of GMP-BULK. If a run of GMP-SHUFFLE completes, it can either succeed (Definition 6), revealing a set of anonymized messages, or fail (Definition 7), in which case some faulty member(s) are blamed. The VERIFY-PROOF-S algorithm is used to validate a proof of a member's misbehavior produced upon a protocol failure.

3.2 The Setup-S Algorithm

SETUP-S(v_i) takes each member's public signing key v_i as input, and outputs a session nonce n_R, a set K of all members' signing keys, an ordering of members τ, and a fixed message length L.

3.3 The Anonymize-S Algorithm

The purpose of ANONYMIZE-S(m_i, K, n_R, τ, f) when run by each member in a group on the collective input messages M is to produce anonymized messages M'. ANONYMIZE-S takes a message m of a fixed length L, K, n_R, τ, and a fail flag f as input. A protocol run of ANONYMIZE-S succeeds for member i if an internal flag SUCCESS$_i$ is set to TRUE after completion of ANONYMIZE-S and fails otherwise. After a successful completion of a protocol run, member i outputs (SUCCESS$, M_i')$, where, as we show in Section 5, M_i' consists of N messages including every message submitted by an honest member. After a protocol failure, member i produces (FAILURE, BLAME$_i, \ell_i)$. BLAME$_i$ includes proofs $p_j = (j, c)$ for each member j for whom a check c of her behavior failed in Phase 6 from i's point of view. At least one of the following checks always fails for some member j from i's point of view provided that SUCCESS$_i$ = FALSE. In such situation a proof p_j is added to BLAME$_i$. The checks are listed in the order they are applied by member i during the protocol. Each check is associated with a check number that ANONYMIZE-S uses to form a proof of a particular form of misbehavior, and VERIFY-PROOF-S uses to confirm a record of that misbehavior.

- Check 1 (c_1): Incomplete log or equivocation (different versions of messages in released logs).
- Check 2 (c_2): Mismatched inner key pair in Phase 5.
- Check 3 (c_3): Empty inner private key in Phase 5 without a justifying GO$_k$ = FALSE or broadcast-hash inequality.
- Check 4 (c_4): Mismatched outer key pair or empty outer private key in Phase 6 regardless of a GO$_k$ = FALSE message or broadcast-hash inequality.

- Check 5 ($\mathtt{c}_5$): Invalid public key in Phase 1.
- Check 6 ($\mathtt{c}_6$): Invalid commitment in Phase 2a.
- Check 7 ($\mathtt{c}_7$): Incorrect commitment or invalid ciphertext or identity in Phase 2b.
- Check 8 ($\mathtt{c}_8$): Incorrect set of permuted ciphertexts after decryption in Phase 3.
- Check 9 ($\mathtt{c}_9$): Invalid ciphertext(s) after decryption in Phase 3.
- Check 10 ($\mathtt{c}_{10}$): Duplicate ciphertext(s) after decryption in Phase 3.
- Check 11 ($\mathtt{c}_{11}$): Incorrect GO_j in Phase 4.
- Check 12 ($\mathtt{c}_{12}$): Incorrect broadcast hash in Phase 4.

For every member i, a complete log includes messages sent and received within SETUP-S and the following messages for each phase of ANONYMIZE-S:

- SETUP-S: All protocol messages.
- Phase 1: Sent: μ_{i1}, received: μ_{k1} for all $k \neq i$.
- Phase 2a: Sent: μ_{i2a}, received: μ_{k2a} for all $k \neq i$.
- Phase 2b: Sent: μ_{i2b}, received: if $i = 1$, then μ_{k2b} for all $k \neq i$, if $i \neq 1$, then no message.
- Phase 3: Sent: μ_{i3}, received: if $i = 1$, then no message, if $i \neq 1$, then $\mu_{(i-1)3}$.
- Phase 4: Sent: μ_{i4}, received: μ_{k4} for all $k \neq i$.
- Phase 5: Sent: μ_{i5}, received: μ_{k5} for all $k \neq i$.
- Phase 6: Sent: μ_{i6}, received: μ_{k6} for all $k \neq i$.

Algorithm description. ANONYMIZE-S(m_i, K, n_R, τ, f)

- Phase 1: Generation of Inner and Outer Key Pairs.

  Each member i chooses two ephemeral encryption key pairs (I_i^{sec}, I_i^{pub}) and (O_i^{sec}, O_i^{pub}), and broadcasts
  $$\mu_{i1} = \{I_i^{pub}, O_i^{pub}, n_R, 1, i\}\text{SIG}_{u_i}.$$
  Member i verifies that the messages she receives contain valid public keys. If the verification fails, member i sets an internal flag GO_i to FALSE to indicate that a step of the protocol failed.

- Phase 2a: Data Commitment.

  Each member i encrypts her datum m_i with all members' inner public keys, in reverse order from I_N^{pub} to I_1^{pub}
  $$C_i' = \{m_i\}_{I_N^{pub}:I_1^{pub}}.$$

Member i stores the inner ciphertext C'_i for later use, then further encrypts C'_i with all members' outer public keys to obtain the outer ciphertext

$$C_i = \{C'_i\}_{O_N^{pub}:O_1^{pub}}.$$

If a public key released by some member j was invalid, i generates and uses a random key for j to allow the protocol to go forward.

Now member i calculates a non-malleable commitment to C_i and i

$$X_i = \text{COMMIT}\{C_i, i\}$$

and broadcasts

$$\mu_{i2a} = \{X_i, n_R, 2\text{a}, i\}\text{SIG}_{u_i}.$$

Member i waits to receive such a message from every other member and then verifies that they include valid commitments. If they do not, GO_i is set to FALSE.

- Phase 2b: Data Submission.

  Member i sends member 1 an opening of her commitment

  $$\mu_{i2b} = \{\text{OPEN}\{X_i\}, n_R, 2\text{b}, i\}\text{SIG}_{u_i}.$$

  Member 1 verifies that each μ_{i2b} successfully opens X_i and that the result is a valid ciphertext and i. If not, member 1 sets GO_1 to FALSE.

- Phase 3: Anonymization.

  Member 1 collects the results of opening the commitments into a vector $\vec{C}_0 = (C_1, \ldots, C_N)$, randomly permutes its elements, then strips one layer of encryption from each ciphertext using private key O_1^{sec} to form $\vec{C}_1$. Member 1 sends to member 2

  $$\mu_{13} = \{\vec{C}_1, n_R, 3, 1\}\text{SIG}_{u_1}.$$

  Each member $1 < i < N$ in turn accepts $\vec{C}_{i-1}$, permutes it randomly, strips one layer of encryption using key O_i^{sec} to form $\vec{C}_i$, then sends $\mu_{i3} = \{\vec{C}_i, n_R, 3, i\}\text{SIG}_{u_i}$ to member $i+1$. Member N similarly creates μ_{N3} and broadcasts it to all members. Member i skips decryption for any invalid ciphertext in $\vec{C}_{i-1}$. Any member i who detects a duplicate or invalid ciphertext in $\vec{C}_i$ sets GO_i to FALSE.

- Phase 4: Verification.

  All members now hold $\vec{C}_N$, which should be a permutation of $C'_1, \ldots, C'_N$. Each member i verifies that her own inner ciphertext C'_i is included in $\vec{C}_N$ and sets GO_i to FALSE if not. If $f = \text{TRUE}$ then member i always sets $\text{GO}_i = \text{FALSE}$ regardless of the above verification. If $f = \text{FALSE}$ and the GO_i flag has not yet been set to FALSE, it is now set to TRUE.

  Each member i creates a vector $\vec{B}$ of all broadcast messages - that is, messages for which identical copies should have been delivered to all members - from prior phases: all members' public key messages from phase 1, all members' commitment messages from phase 2a, and member N's phase 3 message containing $\vec{C}_N$. Thus, $\vec{B} = (\mu_{11}, \ldots, \mu_{N1}, \mu_{12a}, \ldots, \mu_{N2a}, \mu_{N3})$. Member i broadcasts

  $$\mu_{i4} = \{\text{GO}_i, \text{HASH}\{\vec{B}\}, n_R, 4, i\}\text{SIG}_{u_i}.$$

- Phase 5: Key Release and Decryption.

  Case 1. If member i receives $\text{GO}_j = \text{TRUE}$ and $\text{HASH}\{\vec{B}_j\} = \text{HASH}\{\vec{B}_i\}$ from *every* member j, and her $\text{GO}_i = \text{TRUE}$, then member i destroys her copy of C'_i and broadcasts her inner private key I_i^{sec} to all members

  $$\mu_{i5} = \{I_i^{sec}, n_R, 5, i\}\text{SIG}_{u_i}.$$

  Upon receiving messages from *every* other member, member i verifies that each non-empty inner private key I_j^{sec} is valid and corresponds to the public key I_j^{pub}. If member i receives at least one empty key or if any key pair fails the verification, then i sets the internal flag SUCCESS_i to FALSE.

  Otherwise, SUCCESS_i is set to TRUE and member i removes the N levels of encryption from $\vec{C}_N$, resulting in $M'_i = \{m'_1, \ldots, m'_N\}$, the anonymized set of messages submitted to the protocol.

  Case 2. If member i received $\text{GO}_j = \text{FALSE}$ or $\text{HASH}\{\vec{B}_j\} \neq \text{HASH}\{\vec{B}_i\}$ from *any* member j, or her own flag $\text{GO}_i = \text{FALSE}$, then member i destroys her inner private key I_i^{sec}, and sends to all members an empty string instead of her inner private key.

  Member i broadcasts

  $$\mu_{i5} = \{0, n_R, 5, i\}\text{SIG}_{u_i}$$

  and sets the internal flag SUCCESS_i to FALSE.

- Phase 6: Blame.

  Case 1. Member i's $\text{SUCCESS}_i = \text{TRUE}$. In this case, member i acknowledges a successful completion of the protocol. Member i creates a vector $\vec{T}$ of all signed messages she sent and received in Phases 1–5, and broadcasts

  $$\mu_{i6} = \{\vec{T}, n_R, 6, i\}\text{SIG}_{u_i}.$$

  Now, member i outputs $(\text{SUCCESS}, M'_i)$, which completes the protocol.

  Case 2. Member i's $\text{SUCCESS}_i = \text{FALSE}$ and for *every* member j $\text{GO}_j = \text{TRUE}$ and $\text{HASH}\{\vec{B}_j\} = \text{HASH}\{\vec{B}_i\}$.

  Member i keeps her outer private key O_i^{pub} secret, and broadcasts an empty string instead of her key and a vector $\vec{T}$ of all signed messages she sent and received in Phases 1–5

  $$\mu_{i6} = \{0, \vec{T}, n_R, 6, i\}\text{SIG}_{u_i}.$$

  Case 3. Member i's $\text{SUCCESS}_i = \text{FALSE}$ and for *any* member j $\text{GO}_j = \text{FALSE}$ or $\text{HASH}\{\vec{B}_j\} \neq \text{HASH}\{\vec{B}_i\}$. Member i broadcasts her outer private key O_i^{sec}, permutation π_i and a vector $\vec{T}$ of all signed messages she sent and received in Phases 1–5

  $$\mu_{i6} = \{O_i^{sec}, \pi_i, \vec{T}, n_R, 6, i\}\text{SIG}_{u_i}.$$

Now, member i continues with the following steps if she executed Case 2 or Case 3. If member i executed Case 1, then the protocol has completed.

Upon receiving a message μ_{j6} from every other member j, member i inspects every log $\vec{T}$ and discards any message in $\vec{T}$ that is not properly signed or does not have the correct round or phase number. Then, member i verifies each member j's $\vec{T}$ to ensure that it contains all messages sent and received by j in Phases 1–5 as well as that the contents of all messages included in $\vec{T}$ match the corresponding messages in the other $\vec{T}$ logs of other members. For every member j whose $\vec{T}$ is incomplete or for whom different versions of any message $\mu_{j\phi}$ are revealed, member i sets $p_j = (j, \mathtt{c}_1)$, where $\mathtt{c}_1$ indicates the failed check number, and adds p_j to BLAME_i. If there is an incomplete $\vec{T}$ or an equivocation is observed, member i creates a log ℓ_i of the protocol run that consists of all messages sent and received by i during SETUP-S and ANONYMIZE-S. Then, member i outputs $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$, which concludes the protocol.

Otherwise, member i uses those messages in the $\vec{T}$ logs but not sent to i to complete her view of Phases 1–5, and thus she proceeds to examine the remaining part of the protocol. She begins by verifying the inner and outer key pairs revealed by other members. Member i blames each member j who revealed his inner private key I_j^{sec} and for whom the verification of his key pair (I_j^{sec}, I_j^{pub}) failed in Phase 5. Member i sets $p_j = (j, \mathtt{c}_2)$ and adds p_j to BLAME_i Then, for every member j who sent an empty inner private key in Phase 5, member i checks the GO_k flags and broadcast hashes. Member i blames each member j whose inner private key was empty if there is no $\text{GO}_k = \text{FALSE}$ or non-matching broadcast hash. Member i sets $p_j = (j, \mathtt{c}_3)$ and adds p_j to BLAME_i. For every member j who revealed his outer private key in Phase 6, member i checks if the outer private key O_j^{sec} is valid and corresponds to the outer public key O_j^{pub}. In addition, for every member j who sent an empty outer private key in Phase 6, member i checks the vector $\vec{T}$ in μ_{j6} of all messages sent and received by j to verify that she justifies not sending O_j^{sec} by showing that in Phase 4 every $\text{GO}{=}\text{TRUE}$ and all broadcast hashes were the same. For every member j whose outer private key is invalid or non-matching, or who was not justified in withholding the outer private key, member i sets $p_j = (j, \mathtt{c}_4)$ and adds p_j to BLAME_i.

Member i continues by replaying the protocol from the perspective of every member j using that member's revealed messages and keys. Any member who does not follow the protocol given the messages she receives is added to BLAME_i. More precisely, member i examines the actions of the other members in each phase as follows:

- Sub-Phase 1: For every member j who sends an invalid public key, member i sets $p_j = (j, \mathtt{c}_5)$ and adds p_j to BLAME_i.
- Sub-Phase 2a: For every member j who sends an invalid commitment, member i sets $p_j = (j, \mathtt{c}_6)$ and adds p_j to BLAME_i.
- Sub-Phase 2b: For every member j who sends an opening that does not successfully open her commitment or that does not result in a valid ciphertext and identity j, member i sets $p_j = (j, \mathtt{c}_7)$ and adds p_j to BLAME_i.
- Sub-Phase 3: In the case that all outer private keys are revealed and all outer private

keys correspond to the outer public keys, member i checks that every member j sends a permutation of the decrypted valid ciphertexts and the invalid ciphertexts as contained in C_{j-1}. For any member that fails this check, member i sets $p_j = (j, \mathsf{c_8})$ and adds p_j to BLAME_i. Member i further checks that the submitted ciphertexts do not cause failures by producing duplicate or invalid ciphertexts after decryption. If the submitted ciphertext C_j of member j contains an invalid ciphertext after d decryptions, $1 \leq d \leq N$, then member i sets $p_j = (j, \mathsf{c_9})$ and adds p_j to BLAME_i. If the submitted ciphertexts C_j and C_k of members $j \neq k$ decrypt to the same ciphertext after d decryptions, $1 \leq d \leq N$, then member i blames members j and k. Member i sets $p_j = (j, \mathsf{c_{10}})$ and $p_k = (k, \mathsf{c_{10}})$, and then adds p_j and p_k to BLAME_i.

– Sub-Phase 4: In the case that all outer private keys are revealed and all outer private keys correspond to the outer public keys, member i verifies that member j properly reported $\text{GO}_j = \text{FALSE}$ based on the messages seen by j in Phases 1–3. At least one of the following checks must have failed from j's point of view to justify a $\text{GO}_j = \text{FALSE}$.
  * Sub-Sub-Phase 1: Member i verifies the validity of public keys using messages $(\mu_{11}, \ldots, \mu_{N1})$ sent by all members.
  * Sub-Sub-Phase 2a: Member i verifies the correctness of the submitted commitments using $(\mu_{12a}, \ldots, \mu_{N2a})$.
  * Sub-Sub-Phase 2b: (This check is done only for member 1) Member i verifies that the commitments correspond to the ciphertexts and that the resulting ciphertexts and identities are valid using $(\mu_{12a}, \ldots, \mu_{N2a})$ and $(\mu_{12b}, \ldots, \mu_{N2b})$.
  * Sub-Sub-Phase 3: Member i verifies that there are no duplicate or invalid ciphertexts sent from j using μ_{j3}.
  * Sub-Sub-Phase 4: Member i verifies that j's inner ciphertext C'_j is included in $\vec{C_N}$. To determine C'_j, member i opens the commitment X_j and decrypts the resulting ciphertext with each of the outer private keys.

  If all of the above checks were successful and $\text{GO}_j = \text{FALSE}$, then member i sets $p_j = (j, \mathsf{c_{11}})$ and adds p_j to BLAME_i.

  In addition, member i checks if the $\text{HASH}\{\vec{B}_j\}$ that she received in μ_{j4} is correctly calculated from the broadcast messages. If not, member i sets $p_j = (j, \mathsf{c_{12}})$ and adds p_j to BLAME_i.

To conclude the protocol, member i creates a log ℓ_i consisting of the messages sent and received during SETUP-S and ANONYMIZE-S and outputs $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$.

3.4 Verify-Proof-S Algorithm

VERIFY-PROOF-S(p_j, ℓ_i) is used to verify a member j's misbehavior. The algorithm takes as input a proof p_j and a log ℓ_i. It should be that $p_j = (j, c)$, where j is a member's identifier and c is the number of a check which failed for j from i's point of view. ℓ_i should be i's log of a protocol run, including all messages sent and received by member i in SETUP-S and ANONYMIZE-S. VERIFY-PROOF-S outputs TRUE if p_j is a verifiable proof of j's misbehavior based on ℓ_i and FALSE otherwise.

3.4.1 Algorithm description.

VERIFY-PROOF-S(p_j, ℓ_i)

- Step 1: Proof verification. Verify that $p_j = (j, c)$, where c is a valid check number and j is a valid member identifier. If so, then proceed to the next phase. Otherwise, output FALSE and stop.

- Step 2: Log verification. All messages included in log ℓ_i are verified to ensure that signatures on the included messages are valid. Each message is checked to verify that it contains a correct round nonce given the execution of the SETUP-S algorithm and a correct phase number. All messages with invalid signatures, round nonces or phase numbers are discarded. If the resulting log does not include all messages that were supposed to have been sent and received by i during SETUP-S and ANONYMIZE-S, as described in the descriptions of those algorithms, then output FALSE and stop.

  Otherwise, verify that the logs of all sent and received messages revealed in Phase 6 by every member j are complete and consistent. That is, for every message μ_{j6}, consider the included vector $\vec{T}$. Discard any message in $\vec{T}$ that is not properly signed or does not have the correct round or phase number, and inspect every $\vec{T}$ to verify that it includes all messages sent and received in Phases 1–5. Then, for every message recorded as sent by one member and received by another, check that the contents match, and, for every message that is supposed to be a broadcast, check that the contents of all observed copies match. If any $\vec{T}$ is incomplete or inconsistent and $c \neq \mathtt{c}_1$, then output FALSE and stop. Otherwise, if $c = \mathtt{c}_1$ or all logs are complete and consistent, then proceed to the next phase.

- Step 3: Proof verification decision.

  If all $\vec{T}$ logs were determined to be complete and consistent, ℓ_i is augmented to contain all Phase 1–5 messages sent and received by all members. Otherwise, $c = \mathtt{c}_1$, and a log ℓ_i of just i's perspective will be sufficient. The resulting ℓ_i is examined as follows to verify that j failed check c:

  - If $c = \mathtt{c}_1$, then we wish to verify that member j sent an incomplete $\vec{T}$ or equivocated in the protocol.

    Using message μ_{j6}, which is either of the form $\{\vec{T}, n_R, 6, j\}\text{SIG}_{u_j}$, $\{0, \vec{T}, n_R, 6, j\}\text{SIG}_{u_j}$ or $\{O_j^{sec}, \pi_j, \vec{T}, n_R, 6, j\}\text{SIG}_{u_j}$, depending on j's execution of the protocol, check if $\vec{T}$ contains all messages sent and received by j in Phases 1–5 such that all messages are properly signed and include correct phase and round numbers. If it does not, then output TRUE and stop. Otherwise, using the logs $\vec{T}$ in the messages μ_{k6} of each member k, determine whether there exist copies of a message $\mu_{j\phi}$ that are properly signed with correct round and phase numbers but have different contents. If such evidence of equivocation exists, then output TRUE and stop; else output FALSE and stop.

  - If $c = \mathtt{c}_2$, then we wish to verify that member j sent an invalid inner key pair.

    Check if j sent μ_{j5} of the form $\{I_j^{sec}, n_R, 5, j\}\text{SIG}_{u_j}$ in Phase 5. If not, then output FALSE and stop. If yes, then using messages $\mu_{j1} = \{I_j^{pub}, O_j^{pub}, n_R, 1, j\}\text{SIG}_{u_j}$ and μ_{j5}, check if I_j^{pub} and I_j^{sec} is a valid key pair under the chosen encryption scheme. If

I_j^{sec} is invalid or does not match I_j^{pub}, then output TRUE and stop, else output FALSE and stop.

- If $c = \mathtt{c}_3$, then we wish to verify that member j improperly sent an empty inner key in Phase 5.

  Check if j sent μ_{j5} of the form $\{0, n_R, 5, i\}\text{SIG}_{u_i}$ in Phase 5. If not, then output FALSE and stop. If so, then check each message μ_{k4} for $\text{GO}_k = \text{FALSE}$ or a non-matching $\text{HASH}\{\vec{B}_k\}$. If none are found, then output TRUE and stop; else output FALSE and stop.

- If $c = \mathtt{c}_4$, then we wish to verify that member j sent an invalid outer key pair or improperly sent an empty outer private key in Phase 6.

  Check if j sent μ_{j6} of the form $\{O_j^{sec}, \vec{T}, n_R, 6, j\}\text{SIG}_{u_j}$ in Phase 6. If so, then using messages $\mu_{j1} = \{I_j^{pub}, O_j^{pub}, n_R, 1, j\}\text{SIG}_{u_j}$ and μ_{j6}, check whether O_j^{pub} and O_j^{sec} is a valid key pair. If O_j^{sec} is invalid or does not match O_j^{pub}, then output TRUE and stop.

  Otherwise, check if j sent μ_{j6} of the form $\{0, \vec{T}, n_R, 6, i\}\text{SIG}_{u_i}$. If not, then output FALSE and stop. If so, then check if j received a message μ_{k4} from some member k that included either a GO_k set to FALSE or a non-matching $\text{HASH}\{\vec{B}_k\}$. If so, then output TRUE and stop; else output FALSE and stop.

- If $c = \mathtt{c}_5$, then we wish to verify that member j sent an invalid public key in Phase 1.

  Using $\mu_{j1} = \{I_j^{pub}, O_j^{pub}, n_R, 1, j\}\text{SIG}_{u_j}$, check if I_j^{pub} and O_j^{pub} are valid public keys. If I_j^{pub} or O^{pub} is not a valid key, then output TRUE and stop; else output FALSE and stop.

- If $c = \mathtt{c}_6$, then we wish to verify that member j sent an invalid commitment in Phase 2a.

  Using $\mu_{j2a} = \{X_j, n_R, \mathtt{2a}, j\}\text{SIG}_{u_j}$, check whether X_j is a valid commitment. If it is not, then output TRUE and stop; else output FALSE and stop.

- If $c = \mathtt{c}_7$, then we wish to verify that member j's commitment is incorrect or results in an incorrect ciphertext or identity.

  Using $\mu_{j2a} = \{X_j, n_R, \mathtt{2a}, j\}\text{SIG}_{u_j}$ and $\mu_{j2b} = \{\text{OPEN}\{X_j\}, n_R, \mathtt{2b}, j\}\text{SIG}_{u_j}$, check whether X_j matches $\text{OPEN}\{X_j\}$ and results in a valid ciphertext. If X_j does not match $\text{OPEN}\{X_j\}$ or does not yield a valid ciphertext and identity j, then output TRUE and stop, else output FALSE and stop.

- If $c = \mathtt{c}_8$, then we wish to verify that member j did not send a permutation of decrypted ciphertexts in Phase 3.

  Check if every member k sent μ_{k6} of the form $\{O_k^{sec}, \pi_k, \vec{T}, n_R, 6, k\}\text{SIG}_{u_k}$ in Phase 6. If not, then output FALSE and stop. If so, then using $\mu_{k1} = \{I_k^{pub}, O_k^{pub}, n_R, 1, k\}\text{SIG}_{u_k}$ and μ_{k6}, check if each member's outer keys O_k^{sec} and O_k^{pub} are valid and matching. If not, then output FALSE and stop. If so, then, using $\mu_{(j-1)3} = \{\vec{C}_{j-1}, n_R, 3, j-1\}\text{SIG}_{u_{j-1}}$, $\mu_{j3} = \{\vec{C}_j, n_R, 3, j\}\text{SIG}_{u_j}$, and μ_{j6}, check whether $\vec{C}_j$ is a permutation of decrypted ciphertexts. That is, using π_j, permute the elements of the vector $\vec{C}_{j-1}$ included in $\mu_{(j-1)3}$, then decrypt each valid ciphertext using O_j^{sec} and verify whether the resulting vector matches the vector in μ_{j3}. If they do not match, then output TRUE and stop, else output FALSE and stop.

– If $c = \mathtt{c}_9$, then we wish to verify that member j's decrypted outer ciphertext C_j results in an invalid ciphertext.

Check if every member k sent μ_{k6} of the form $\{O_k^{sec}, \pi_k, \vec{T}, n_R, \mathtt{6}, k\}\mathrm{SIG}_{u_k}$ in Phase 6. If not, then output FALSE and stop. If so, then using $\mu_{k1} = \{I_k^{pub}, O_k^{pub}, n_R, \mathtt{1}, k\}\mathrm{SIG}_{u_k}$ and μ_{k6}, check if each member's outer keys O_k^{sec} and O_k^{pub} are valid and matching. If not, then output FALSE and stop. If so, then using $\mu_{j2b} = \{\mathrm{OPEN}\{X_j\}, n_R, \mathtt{2b}, j\}\mathrm{SIG}_{u_j}$, produce ciphertext C_j. Then use the outer private keys to iteratively remove the layers of encryption from the ciphertexts in C_j, verifying that a valid ciphertext is produced after every step. If at any point an invalid ciphertext is produced, then output TRUE and stop, else output FALSE and stop.

– If $c = \mathtt{c}_{10}$, then we wish to verify that member j's decrypted outer ciphertext C_j results in a duplicate ciphertext.

Check if every member sent μ_{k6} of the form $\{O_k^{sec}, \pi_k, \vec{T}, n_R, \mathtt{6}, k\}\mathrm{SIG}_{u_k}$ in Phase 6. If not, then output FALSE and stop. If so, then using $\mu_{k1} = \{I_k^{pub}, O_k^{pub}, n_R, \mathtt{1}, k\}\mathrm{SIG}_{u_k}$ and μ_{k6}, check if each member's outer keys O_k^{sec} and O_k^{pub} are valid and matching. If not, then output FALSE and stop. If so, then using the $\mu_{k2b} = \{\mathrm{OPEN}\{X_k\}, n_R, \mathtt{2b}, k\}\mathrm{SIG}_{u_k}$ of every member k, produce the submitted ciphertexts C_k. Use the outer private keys to iteratively remove the layers of encryption from the valid ciphertexts in each C_k, and if at any point the result for C_j is the same as the result for some other C_k, then output TRUE and stop, else output FALSE and stop.

– If $c = \mathtt{c}_{11}$, then we wish to verify that member j sent an incorrect GO_j in Phase 4.

Check if every member sent μ_{k6} of the form $\{O_k^{sec}, \pi_k, \vec{T}, n_R, \mathtt{6}, k\}\mathrm{SIG}_{u_k}$ in Phase 6. If not, then output FALSE and stop. If so, then using $\mu_{k1} = \{I_k^{pub}, O_k^{pub}, n_R, \mathtt{1}, k\}\mathrm{SIG}_{u_k}$ and μ_{k6}, check if each member's outer keys O_k^{sec} and O_k^{pub} are valid and matching. If not, then output FALSE and stop. If so, then check if $\mathrm{GO}_j = \mathrm{FALSE}$ in μ_{j4}. If not, then output FALSE and stop, else continue.

  * A-S Phase 1: Using $(\mu_{11}, \ldots, \mu_{N1})$ check whether j received valid inner and outer public keys. If any key is invalid, then output FALSE and stop.
  * A-S Phase 2a: Using $(\mu_{12a}, \ldots, \mu_{N2a})$ verify whether commitments $(X_1, \ldots, X_N)$ are valid. If any commitment is invalid, then output FALSE and stop.
  * A-S Phase 2b: If $j = 1$, then using $(\mu_{12a}, \ldots, \mu_{N2a})$ and $(\mu_{12b}, \ldots, \mu_{N2b})$ verify whether X_k matches $\mathrm{OPEN}\{X_k\}$ and results in a valid ciphertext and identity k for all $k \in G$. If any commitment does not properly open or results in an invalid ciphertext or identity, then output FALSE and stop.
  * A-S Phase 3: Using μ_{j3}, check whether the contained set of ciphertexts includes duplicate or invalid ciphertexts. If there is an invalid or duplicate ciphertext, then output FALSE and stop.
  * A-S Phase 4: Using μ_{j2b}, $(\mu_{16}, \ldots, \mu_{N6})$, and μ_{N3} verify whether j's inner ciphertext C'_j was included in $\vec{C_N}$. To determine C'_j, open the commitment X_j included in μ_{j2b} and decrypt the resulting ciphertext with each of the outer private keys included in $(\mu_{16}, \ldots, \mu_{N6})$. If the calculated C'_j was not included in C_N, then output FALSE and stop, else output TRUE and stop.

– If $c = \mathtt{c}_{12}$, then we wish to verify that j sent an incorrect HASH$\{\vec{B}\}$. Calculate $\vec{B}'$ using messages $(\mu_{11}, \ldots, \mu_{N1}, \mu_{12a}, \ldots, \mu_{N2a}, \mu_{N3})$ received by j. Then, check whether HASH$\{\vec{B}'\}$ matches the HASH$\{\vec{B}\}$ included in μ_{j4}. If HASH$\{\vec{B}'\} \neq$ HASH$\{\vec{B}\}$, then output TRUE, else output FALSE.

4 GMP-Bulk

4.1 Protocol Description

The Group Messaging Protocol-Bulk GMP-BULK is an instantiation of the Group Messaging Protocol and consists of three algorithms: SETUP-B, ANONYMIZE-B, and VERIFY-PROOF-B. Each member i submits a message m_i of variable length L_i to the ANONYMIZE-B protocol after all members run SETUP-B to agree on common protocol run parameters. If a run of GMP-BULK completes, it can either succeed (Definition 6) or fail (Definition 7). In case of a protocol failure the VERIFY-PROOF-B protocol is used to validate the proofs of member's misbehavior generated upon a protocol failure.

4.2 The Setup-B Algorithm

SETUP-B(v_i) takes each member's public signing key v_i as input, and outputs a session nonce n_R identifying a run of ANONYMIZE-B, a session nonce n_{R_1} identifying a run of ANONYMIZE-S in Phase 3 of ANONYMIZE-B, and a session nonce n_{R_2} identifying a run of ANONYMIZE-S in Phase 7 of ANONYMIZE-B, a set K of members' signing keys , and an ordering of members τ. Since members submit messages of variable lengths, there is no need to agree on a fixed message length L.

4.3 The Anonymize-B Algorithm

The purpose of ANONYMIZE-B$(m_i, K, n_R, n_{R_1}, n_{R_2}, \tau)$ when run by each member in a group on the collective input messages M is to produce anonymized messages M'. The algorithm takes a message m and the output of SETUP-B as input. A run of ANONYMIZE-B succeeds for member i if, upon completion of ANONYMIZE-B, her internal flag SUCCESS$_i$ is set to TRUE, and fails if SUCCESS$_i$ is set to FALSE. If a protocol run succeeds, then member i outputs (SUCCESS$, M'_i)$, where, as we show in Section 5, M'_i consists of N messages including every message submitted by an honest member. If a protocol run fails, then member i produces (FAILURE$,$ BLAME$_i, \ell_i)$. BLAME$_i$ includes proofs $p_j = (j, c)$ for each member j for whom a check c fails in Phase 7 from member i's point of view. The checks in this phase are as follows, listed in the order they are applied by member i during the protocol. As before, each check is associated with a check number that ANONYMIZE-B uses to form a proof of a particular form of misbehavior, and VERIFY-PROOF-B uses to confirm a record of that misbehavior.

- Check 1 ($\mathtt{c}_1$): Equivocation in Phase 4 or Phase 5.
- Check 2 ($\mathtt{c}_2$): Failure of ANONYMIZE-S in Phase 3 or Phase 7 without justification.
- Check 3 ($\mathtt{c}_3$): Empty or incorrect ciphertext(s) sent in Phase 4.

- Check 4 ($\mathtt{c}_4$): Unverifiable proof included in the notification in Phase 4.
- Check 5 ($\mathtt{c}_5$): Invalid public key sent in Phase 1a.
- Check 6 ($\mathtt{c}_6$): Equivocation in Phase 1a.

The log ℓ_i includes all messages sent and received by i during SETUP-B and ANONYMIZE-B as well as the output of ANONYMIZE-S in Phase 3 and Phase 7.
For every member j, a complete log ℓ_j consists of the following messages.

- SETUP-B: All protocol messages.
- Phase 1a: Sent: μ_{j1a}, received: μ_{k1a} for all $k \neq j$.
- Phase 1b: Sent: μ_{j1b}, received: μ_{k1b} for all $k \neq j$.
- Phase 2: No messages.
- Phase 3: Sent: μ_{j3} and all messages sent in shuffle, received: μ_{k3} for all $k \neq j$, and all messages received in shuffle.

  ANONYMIZE-S output: $M'_j = d'_1, \ldots, d'_N$ if ANONYMIZE-S succeeds or $\text{BLAME}_j^{s_1}, \ell_j^{s_1}$ if ANONYMIZE-S fails as well as all messages sent and received within the protocol.
- Phase 4: Sent: μ_{j4}, received: μ_{k4} for all $k \neq j$.
- Phase 5: Sent: μ_{j5}, received: μ_{k5} for all $k \neq j$.
- Phase 6: No messages.
- Phase 7: Sent: μ_{j7} and all messages sent in shuffle; received: μ_{k7} for all $k \neq j$ and all messages received in shuffle.

  ANONYMIZE-S output: $M'_j = A'_1, \ldots, A'_N$ if ANONYMIZE-S succeeds or $\text{BLAME}_j^{s_2}, \ell_j^{s_2}$ if ANONYMIZE-S fails as well as all messages sent and received within the protocol.

Algorithm description. ANONYMIZE-B$(m_i, K, n_R, n_{R_1}, n_{R_2}, \tau)$

- Phase 1a: Session Key Pair Generation.

  Each member i chooses an ephemeral encryption key pair (x_i, y_i) and broadcasts

  $$\mu_{i1a} = \{y_i, n_R, \mathtt{1a}, i\}\text{SIG}_{u_i}.$$

- Phase 1b: Key Verification.

  After receiving a public key from every member j, member i notifies other members about the set of keys she receives. Member i creates $\vec{K}_i^e = \{\mu_{11a}, \ldots, \mu_{N1a}\}$ and broadcasts

  $$\mu_{i1b} = \{\vec{K}_i^e, n_R, \mathtt{1b}, i\}\text{SIG}_{u_i}.$$

- Phase 2: Message Descriptor Generation.

  Member i creates a *message descriptor* d_i of a fixed length Λ_d. Member i sets $L_i = 0$ if she does not wish to send a message in this protocol run and L_i to the desired message length if she wishes to send a message.

  Case 1. Successful key verification. Member i verifies each set of public keys received in Phase 1b to ensure that other members received the same set of valid public keys. If every $\vec{K}^e_j$ contains the same set of public keys and every public key $y_j \in \vec{K}^e_j$ is valid, then member i chooses a random seed s_{ij} for each member j and generates L_i pseudorandom bits from s_{ij} to obtain ciphertext

  $$C_{ij} = \text{PRNG}\{L_i, s_{ij}\} \quad (j \neq i),$$

  where L_i and s_{ij} are of fixed lengths for all members.

  Member i now XORs her message m_i with each C_{ij} for $j \neq i$ to obtain ciphertext C_{ii}:

  $$C_{ii} = C_{i1} \oplus \ldots \oplus C_{i(i-1)} \oplus m_i \oplus C_{i(i+1)} \oplus \ldots \oplus C_{iN}$$

  Member i computes hashes $H_{ij} = \text{HASH}\{C_{ij}\}$, encrypts each seed s_{ij} with j's public key to form $S_{ij} = \{s_{ij}\}^{R_{ij}}_{y_j}$, and collects the H_{ij} and S_{ij} into vectors $\vec{H}_i$ and $\vec{S}_i$:

  $$\vec{H}_i = (H_{i1}, \ldots, H_{iN})$$

  $$\vec{S}_i = (S_{i1}, \ldots, S_{iN})$$

  Member i forms a message descriptor d_i, which has a fixed length Λ_d

  $$d_i = \{L_i, \vec{H}_i, \vec{S}_i\}.$$

  Case 2. Failed key verification. If any $\vec{K}^e_j$ contains a non-matching set of keys or any $\vec{K}^e_j$ contains an invalid key, then member i creates an empty message descriptor of the desired length Λ_d

  $$d_i = 0^{\Lambda_d}.$$

  Case 3. No message to send. If member i chooses not to send a message in this protocol run, she sets $L_i = 0$ and assigns random values to $\vec{H}_i$ and $\vec{S}_i$.

  Member i forms her message descriptor d_i as follows and pads it to the desired length Λ_d

  $$d_i = \{L_i, \vec{H}_i, \vec{S}_i\}.$$

- Phase 3: Message Descriptor Shuffle.

  Each member i runs the ANONYMIZE-S protocol described in Section 3 using $(d_i, K, n_{R_1}, \tau, f_i)$ as input, where the fixed-length descriptor d_i is the secret message to be shuffled. Member i sets $f_i = \text{TRUE}$ if i created an empty message descriptor, and member i sets $f_i = \text{FALSE}$ otherwise.

If ANONYMIZE-S succeeds, member i has a list M_i' of message descriptors in some random permutation π. If the protocol fails outputting $(\text{FAILURE}, \text{BLAME}_i^{s_1}, \ell_i^{s_1})$, member i saves $\text{BLAME}_i^{s_1}$ and $\ell_i^{s_1}$.

If member i set $f_i = \text{TRUE}$, then i prepares a proof p' of the dishonest member j's misbehavior to distribute to other members. If member j sent an invalid key, then member i sets $p' = (j, \mathsf{c}_5, \mu_{j1a})$, where c_5 indicates the failed check number and μ_{j1a} is the message received by i in Phase 1a. If member j equivocated, then member i sets $p' = (j, \mathsf{c}_6, \mu_{j1a}, \mu'_{j1a})$, where μ_{j1a} is the message received by i in Phase 1a and μ'_{j1a} is a message included in some $\vec{K}_k^e$ that contains a different key for j than in μ_{j1a}. If there is more than one culprit member j, member i chooses one j to blame in some way that does not depend on her message (e.g. randomly). If member i received all valid and matching keys, then member i sets $p' = 0$.

Member i broadcasts:

$$\mu_{i3} = \{p', n_R, 3, i\}\text{SIG}_{u_i}.$$

- Phase 4: Data Transmission.

  Case 1. If ANONYMIZE-S fails, then member j sets $\text{GO}_j = \text{FALSE}$ and shares her blame set $\text{BLAME}_j^{s_1}$ and log $\ell_j^{s_1}$ by broadcasting

  $$\mu_{j4} = \{\text{GO}_j, \text{BLAME}_j^{s_1}, \ell_j^{s_1}, n_R, 4, j\}\text{SIG}_{u_j}.$$

  Case 2. If ANONYMIZE-S succeeds, member j sets $\text{GO}_j = \text{TRUE}$ and decrypts each encrypted seed S_{ij} with private key x_j to reveal s_{ij}. If s_{ij} matches the seed s_{jj} that j chose for herself in her own descriptor, then j sets $C_{ij} = C_{jj}$. Otherwise, j sets $C_{ij} = \text{PRNG}\{L_i, s_{ij}\}$. Member j then checks $\text{HASH}\{C_{ij}\}$ against H_{ij}. If the hashes match, j sets $C'_{ij} = C_{ij}$. If S_{ij} is not a valid ciphertext, s_{ij} is not a valid seed, or $\text{HASH}\{C_{ij}\} \neq H_{ij}$, then j sets C'_{ij} to an empty ciphertext, $C'_{ij} = \{\}$.

  Member j now sends each C'_{ij} in π-shuffled order by broadcasting

  $$\mu_{j4} = \{\text{GO}_j, C'_{\pi(1)j}, \ldots, C'_{\pi(N)j}, n_R, 4, j\}\text{SIG}_{u_j}.$$

- Phase 5: Acknowledgment Submission.

  Each member k notifies other members about the outcome of the previous phase.

  Case 1. If $\text{GO}_j = \text{FALSE}$ for any member j, then member k adds each message μ_{j4} containing $\text{GO}_j = \text{FALSE}$ into a vector $\vec{V}_k$.

  Case 2. If $\text{GO}_j = \text{TRUE}$ for every member j but some ciphertext C'_{ij} is empty or satisfies $\text{HASH}\{C'_{ij}\} \neq H_{ij}$, then slot $\pi(i)$ has been corrupted. Member k adds each message μ_{j4} containing such a corrupting ciphertext to a vector $\vec{V}_k$.

  Case 3. If $\text{GO}_j = \text{TRUE}$ for every member j and all ciphertexts C'_{ij} are non-empty and satisfy $\text{HASH}\{C'_{ij}\} = H_{ij}$, then member k sets $\vec{V}_k = \{\}$.

  In every case member k broadcasts

  $$\mu_{k5} = \{\vec{V}_k, n_R, 5, k\}\text{SIG}_{u_k}.$$

- Phase 6: Message Recovery.

  If $\text{GO}_i = \text{TRUE}$ for every member i, then for each uncorrupted slot $\pi(i)$, member k recovers member i's message by computing

  $$m'_i = C'_{i1} \oplus ... \oplus C'_{iN}.$$

  If $\vec{V}_k = \{\}$, then from member k's point of view none of the slots were corrupted and all messages $M'_k = (m'_1, \ldots, m'_N)$ were successfully recovered. If $\vec{V}_k \neq \{\}$, then some message slot was corrupted or a step of the protocol has failed.

- Phase 7: Blame.

  For each member i, if i observed a corrupted slot with a descriptor matching d_i (there may be more than one) and received all $\text{GO}_j = \text{TRUE}$, then i generates an *accusation* naming the member j who sent that incorrect ciphertext. If there is more than one culprit member, member i chooses one to blame in any way that only depends on the output of ANONYMIZE-S and on $\vec{V}_i$. Each accusation has a fixed length Λ_a, indicates the corrupted slot $\pi(i)$, contains the seed s_{ij} that i assigned j, and contains the random bits that i used to encrypt the seed:

  $$A_i = \{j, \pi(i), s_{ij}, R_{ij}\}.$$

  Each member i who does not have an accusation to send submits the empty accusation $A_i = 0^{\Lambda_a}$.

  These accusations will be sent anonymously using the ANONYMIZE-S protocol. However, before running it, members look for evidence of equivocation in the previous two rounds. Every member i compares each message μ'_{j4} that she received in some $\vec{V}_k$ in Phase 5 with the message μ_{j4} that she received directly from j in Phase 4. If the contents of these do not match, ignoring any μ'_{j4} with an improper signature or incorrect round or phase number, then member sets $f_i = \text{TRUE}$ to cause ANONYMIZE-S to fail in order to inform other members about the equivocation. If all such messages match, member i sets $f_i = \text{FALSE}$.

  Member i then runs ANONYMIZE-S$(A_i, K, n_{R_2}, \tau, f_i)$. After ANONYMIZE-S completes, there is an opportunity for members who deliberately failed the shuffle to distribute evidence of equivocation. For a member i who set $f_i = \text{TRUE}$ because of conflicting messages μ'_{j4} and μ_{j4}, i creates a proof of j's equivocation by setting $p' = (j, \mathtt{c}_1, \mu_{j4}, \mu'_{j4})$. If there is more than one culprit member j, member i chooses one j to blame in any way that depends at most on the broadcast messages μ_{k4} and μ_{k5} sent and received by i. If member i had $f_i = \text{FALSE}$, then i sets $p' = 0$. Member i then broadcasts

  $$\mu_{i7} = \{p', n_R, 7, i\}\text{SIG}_{u_i}.$$

  Let O_k be the output of the ANONYMIZE-S protocol for member k. After receiving a message μ_{i7} from every other member i, member k executes one of the following cases.

Case 1: $O_k = (\text{FAILURE}, \text{BLAME}_k^{s2}, \ell_k^{s2})$.

Member k sets $\text{SUCCESS}_k = \text{FALSE}$. Then k considers every blame entry $(i, c) \in \text{BLAME}_k^{s2}$. If $c \neq \mathtt{c}_{11}$, then i could not have justifiably caused the blame shuffle to fail, and so k adds $(i, \mathtt{c}_2)$ to BLAME_k. Otherwise $c = \mathtt{c}_{11}$, and member k looks in μ_{i7} for possible justification of the failure. If μ_{i7} does include two versions of the same ciphertext $C'_{\ell j}$ (included in properly signed messages that include correct phase and round numbers) for some member j, then k adds $(j, \mathtt{c}_1)$ to BLAME_k. Otherwise, k adds $(i, \mathtt{c}_2)$ to BLAME_k.

Case 2: $O_k = (\text{SUCCESS}, M_k^{s2})$ and $\vec{V}_k = \{\}$.

Member k sets $\text{SUCCESS}_k = \text{TRUE}$.

Case 3: $O_k = (\text{SUCCESS}, M_k^{s2})$ and $\vec{V}_k$ includes ciphertexts.

k checks the validity of every accusation $A_i = (j, \pi(i), s_{ij}, R_{ij})$ in M_k^{s2} that targets an incorrect ciphertext received by k. To do so, k replays the encryption $S'_{ij} = \{s_{ij}\}_{y_j}^{R_{ij}}$, checks that the encrypted seed S_{ij} included in d_i matches S'_{ij}, and checks that the hash H_{ij} in d_i matches $\text{HASH}\{\text{PRNG}\{L_i, s_{ij}\}\}$, where L_i is also obtained from d_i. If the accusation is valid, then member k adds $(j, \mathtt{c}_3)$ to BLAME_k. If M_k^{s2} includes no valid accusation targeting an incorrect ciphertext received by k, then k sets $\text{SUCCESS}_k = \text{TRUE}$. Otherwise, member k sets $\text{SUCCESS}_k = \text{FALSE}$.

Case 4: $O_k = (\text{SUCCESS}, M_k^{s2})$ and $\vec{V}_k$ contains $\text{GO}_i = \text{FALSE}$ for some i.

Member k sets $\text{SUCCESS}_k = \text{FALSE}$. Then k considers every $\text{GO}_i = \text{FALSE}$ in V_k.

Member k checks μ_{i4} to see if the contained blame set and log constitute a valid proof of some member j's misbehavior. To do so, member k checks that ℓ_i^{s1} contains n_{R_1} as the round number that is a result of SETUP-B and that $\text{VERIFY-PROOF-S}(p_j, \ell_i^{s1}) = \text{TRUE}$ for some $p_j \in \text{BLAME}_i^{s1}$. If not, then member k blames i by adding $(i, \mathtt{c}_4)$ to BLAME_k. If so, then k considers every $p_j \in \text{BLAME}_i^{s1}$ such that $\text{VERIFY-PROOF-S}(p_j, \ell_i^{s1}) = \text{TRUE}$. If $p_j \neq (j, \mathtt{c}_{11})$, then member k adds $(j, \mathtt{c}_2)$ to BLAME_k. If $p_j = (j, \mathtt{c}_{11})$, then member k examines μ_{j3} to see if member j justifiably caused a failure of ANONYMIZE-S to expose bad key distribution by some member ℓ. If μ_{j3} includes an invalid key y_ℓ in a properly signed message with correct round and phase numbers, then member k adds $(\ell, \mathtt{c}_5)$ to BLAME_k. If μ_{j3} includes two different versions of public key y_ℓ in properly signed messages with correct round and phase numbers, then member adds $(\ell, \mathtt{c}_6)$ to BLAME_k. Otherwise, k adds $(j, \mathtt{c}_2)$ to BLAME_k.

In every case, k concludes as follows. If $\text{SUCCESS}_k = \text{TRUE}$, k outputs $(\text{SUCCESS}, M'_k)$. Otherwise, member k creates a log ℓ_k of the protocol run that all messages sent and received by k during SETUP-B and ANONYMIZE-B as well as the output of the ANONYMIZE-S protocol in Phases 3 and 7. Member k outputs $(\text{FAILURE}, \text{BLAME}_k, \ell_k)$.

4.4 Verify-Proof-B Algorithm

The $\text{VERIFY-PROOF-B}(p_j, \ell_i)$ algorithm is used to verify a member's misbehavior. VERIFY-PROOF-B takes as input a proof p_j and a log ℓ_i. A proof p_j should consist of a tuple (j, c), where j is a member's identifier and c indicates the check that failed for member j from member i's point of view. A log ℓ_i should include all messages sent and received during SETUP-B and ANONYMIZE-B by member i as well as the output of ANONYMIZE-S in Phases 3 and 7. The protocol outputs TRUE if p_j is a proof of j's misbehavior given i's log ℓ_i and FALSE otherwise.

Algorithm description.

VERIFY-PROOF-B(p_j, ℓ_i)

- Step 1: Proof verification.

  Verify that p_j includes a valid check number c and member identifier j. If the proof p_j is valid, then proceed to the next phase. If p_j is invalid, then output FALSE and stop.

- Step 2: Log verification.

  All messages included in the log ℓ_i are verified to ensure that signatures on included messages are valid given the included member identifier. Each message is checked to verify that it contains a correct round nonce given the execution of the SETUP-B protocol and a correct phase number. All messages with invalid signatures, round nonces, or phase numbers are discarded. If the resulting log does not include all messages that were supposed to have been sent and received by i during SETUP-B and ANONYMIZE-B, as described in the descriptions of those algorithms, as well as the output of ANONYMIZE-S in Phases 3 and 7, then output FALSE. Otherwise, proceed to the next phase.

- Step 3: Proof verification decision.

  Log ℓ_i is examined as follows to verify that j failed check c:

  - If $c = \mathtt{c}_1$, then we wish to verify that member j equivocated in Phase 4 or Phase 5.

    Check if ANONYMIZE-S failed in Phase 7. If not, then output FALSE and stop. If yes, then use log ℓ_i^{s2} to check each message $\mu_{k7} = \{p', n_R, 7, k\}\text{SIG}_{u_k}$. If no p' is of the form $(j, \mathtt{c}_1, \mu_{j4}, \mu'_{j4})$, where μ_{j4} and μ'_{j4} are properly signed messages with correct round and phase numbers and are of the form $\{\text{TRUE}, C_1, \ldots, C_N, n_R, 4, j\}\text{SIG}_{u_j}$ for some ciphertexts C_i, then output FALSE and stop. Else, if μ_{j4} and μ'_{j4} contain different messages for any such p', then output TRUE and stop. Else output FALSE and stop.

  - If $c = \mathtt{c}_2$, then we wish to verify that member j caused a failure of ANONYMIZE-S in Phase 3 or Phase 7 without justification.

    Check if either ANONYMIZE-S failed in Phase 7 or there was some μ_{k4} in $\vec{V}_i$ with $\text{GO}_k =$ FALSE. If not, then output FALSE and stop.

    If ANONYMIZE-S failed in Phase 7, then consider each proof $p_j \in \text{BLAME}_i^{s2}$ blaming j. Verify that VERIFY-PROOF-S$(p_j, \ell_i^{s2}) =$ TRUE and that ℓ_i^{s2} uses n_{R_2} as the round number, and if not discard this proof. Otherwise, if $p_j \neq (j, \mathtt{c}_{11})$ then output TRUE and stop. If instead $p_j = (j, \mathtt{c}_{11})$, then we must check whether j caused a protocol failure in order to distribute a proof of equivocation of some other member k. Using message $\mu_{j7} = \{p', n_R, 7, j\}\text{SIG}_{u_j}$, check if p' is of the form $(k, \mathtt{c}_1, \mu_{k4}, \mu'_{k4})$ with $k \neq j$ and where μ_{k4} and μ'_{k4} have different contents and are properly signed with correct round and phase numbers. If not, then output TRUE and stop. If no proof results in an output of TRUE, then output FALSE and stop.

Otherwise, the blame shuffle succeeded for i, but some member indicated a failure of the descriptor shuffle. For every k that sent a μ_{k4} of the form $\{\text{FALSE}, \text{BLAME}_k^{s_1}, \ell_k^{s_1}, n_R, 4, k\}\text{SIG}_{u_k}$, consider every proof $p_j \in \text{BLAME}_k^{s_1}$ blaming j. Verify that $\text{VERIFY-PROOF-S}(p_j, \ell_i^{s_1}) = \text{TRUE}$, and that the round number in $\ell_k^{s_1}$ is n_{R_1}, and if not discard this proof. Otherwise, if $p_j \neq (j, \mathtt{c}_{11})$, then output TRUE and stop. If instead $p_j = (j, \mathtt{c}_{11})$, then we must check whether j caused a protocol failure in order to distribute a proof of misbehavior of some other member k. Using message $\mu_{j3} = \{p', n_R, 3, j\}\text{SIG}_{u_j}$, check if (*i*) p' is of the form $(k, \mathtt{c}_5, \mu_{k1a})$ with $k \neq j$ and where μ_{k1a} contains an invalid public key y_k and is properly signed with correct round and phase numbers, or (*ii*) p' is of the form $(k, \mathtt{c}_6, \mu_{k1a}, \mu'_{k1a})$ with $k \neq j$ and where the keys in μ_{k1a} and μ'_{k1a} are unequal and both messages are properly signed with correct round and phase numbers. If not, then output TRUE and stop. If no proof p_j results in an output of TRUE, then output FALSE and stop.

– If $c = \mathtt{c}_3$, then we wish to verify that member j sent an empty or incorrect ciphertext C'_{kj} in Phase 4.

  Check if (*i*) j sent μ_{j4} of the form $\{\text{TRUE}, C'_{\pi(1)j}, \ldots, C'_{\pi(N)j}, n_R, 4, j\}\text{SIG}_{u_j}$ in Phase 4, and (*ii*) ANONYMIZE-S in Phase 7 succeeded for member i with an accusation $A_k = \{j, \pi(k), s_{kj}, R_{kj}\}$ naming j as a faulty member in its output. If not, then output FALSE and stop.

  Otherwise, we need to check that the accusation against j is valid. Doing so requires comparing the accusation to the descriptors received by j. We need to be sure that j received the descriptors claimed by i. To do so, first recompute the hash of broadcast messages in Phases 1–3 of the descriptor shuffle and compare it to the hash that i sent in Phase 4 of that shuffle. If the hashes are not the same, output FALSE and stop. Otherwise, further compare them to the hash sent by j in Phase 4 of the descriptor shuffle. If they do not match, output FALSE and stop.

  Otherwise, examine the inner private keys received by i in Phase 5 of the descriptor shuffle. If any key I_k^{sec} is invalid or does not match its public key I_k^{pub}, output FALSE and stop.

  Otherwise, use these keys to decrypt the inner ciphertexts contained in the final broadcast of Phase 3. Let $\{L_k, \vec{H}_k, \vec{S}_k\}$ be the resulting descriptor in the slot $\pi(k)$ pointed to by the accusation. Recall that $C'_{\pi(k)j}$ is the ciphertext for this slot that j sent to i in message μ_{j4}. Check if (*i*) $\text{HASH}\{C'_{\pi(k)j}\}$ does not match the hash in the jth element of $\vec{H}_k$, (*ii*) the encryption of the accusation seed s_{kj} under the key sent in μ_{j1a} using the random bits R_{kj} of the accusation is equal to the jth encrypted seed in $\vec{S}_k$, and (*iii*) $\text{HASH}\{\text{PRNG}\{L_k, s_{kj}\}\}$ is equal to the hash in the jth element of $\vec{H}_k$. If not, output FALSE and stop. If so, output TRUE and stop.

– If $c = \mathtt{c}_4$, then we wish to verify that member j unjustifiably reported in Phase 4 a failure of ANONYMIZE-S.

  Check if j sent μ_{j4} of the form $\{\text{FALSE}, \text{BLAME}_j^{s_1}, \ell_j^{s_1}, n_R, 4, j\}\text{SIG}_{u_j}$. If not, then output FALSE and stop. If so, examine μ_{j3} to see if j justifiably caused failure of the descriptor shuffle. If (*i*) it contains an invalid key y_k in a properly signed message with correct round and phase numbers, or (*ii*) it contains two different versions of the same key y_k in properly signed messages with correct round and phase numbers, then output FALSE and stop.

Otherwise, check if (*i*) ℓ_j^{s1} does not contain the round number n_{R_1} that is the output of SETUP-B in ℓ_i, or (*ii*) $\forall p_i \in \text{BLAME}_j^{s1}$ VERIFY-PROOF-S$(p_i, \ell_j^{s1}) = \text{FALSE}$. If so, then output TRUE and stop, else output FALSE and stop.

– If $c = \mathsf{c}_5$, then we wish to verify that member j sent an invalid key in Phase 1a.

  Check if $\mu_{k3} = \{p', n_R, 3, k\}\text{SIG}_{u_k}$ sent by any member k contains p' of the form $(j, \mathsf{c}_5, \mu_{j1a})$, where μ_{j1a} contains an invalid public key y_j and is properly signed with correct round and phase numbers. If yes, then output TRUE and stop, else output FALSE and stop.

– If $c = \mathsf{c}_6$, then we wish to verify that member j equivocated in Phase 1a and sent two different public keys.

  Check if any $\mu_{k3} = \{p', n_R, 3, k\}\text{SIG}_{u_k}$ contains p' of the form $(j, \mathsf{c}_6, \mu_{j1a}, \mu'_{j1a})$ such that μ_{j1a} and μ'_{j1a} have different message contents and are properly signed with correct round and phase numbers. If yes, then output TRUE and stop, else output FALSE and stop.

5 Security properties and proofs

In this section, we formally define and analyze integrity, accountability, and anonymity and prove that DISSENT satisfies these properties. These definitions are precise versions of the notions used by Corrigan-Gibbs and Ford (2010).

5.1 Notation

Let G be the set of all members participating in the protocol, H be the set of honest members and D the set of dishonest members. For security properties expressed as a game between an adversary A and challenger C, we denote the output of the adversary as A^C. We use $\Delta(G^i)$ to denote $\left|Pr\left[G^i(0) = 1\right] - Pr\left[G^i(1) = 1\right]\right|$, which is the *advantage* of game G^i. We also use $\overline{b}$ to indicate the complement of bit b: $\overline{b} = 1 - b$.

5.2 Preliminary Definitions

We use the following technical definitions, some making precise notions discussed earlier and some introduced here, to express the security definitions, theorems, and proofs.

Definition 5. *A function is* negligible *in an input if it is non-negative and goes to zero with that input asymptotically faster than any inverse polynomial. The input is assumed to be a security parameter unless otherwise stated.*

Definition 6. *A protocol run of a* GMP *protocol* succeeds *for member* i *if the* ANONYMIZE *algorithm terminates with output* $(\text{SUCCESS}, M_i')$.

Definition 7. *A protocol run of a* GMP *protocol* fails *for member* i *if the* ANONYMIZE *algorithm terminates with output* $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$.

Definition 8. *A member is* honest *if she faithfully carries out the protocol according to its specification, does not cooperate with the adversary, and is not under his control.*

Definition 9. *A member is* dishonest *if she is not honest.*

Definition 10. *A group member i* blames *member j if $p_j \in \text{BLAME}_i$ upon a protocol failure resulting in $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$.*

Definition 11. *A* verifiable proof *of j's misbehavior given ℓ_i is a p_j such that* $\text{VERIFY-PROOF}(p_j, \ell_i) =$ TRUE.

Definition 12. *A group member i* exposes *member j if i holds a verifiable proof of j's misbehavior given a log ℓ_i of a protocol run in which member j participated using his long-term signing key u_j.*

5.3 Integrity

Definition 13. *A Group Messaging Protocol* GMP *offers* integrity *if after a complete run of the protocol involving N group members*

1. *each honest member i terminates with either $(\text{SUCCESS}, M'_i)$ or $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$, and*
2. *for every honest member who terminates with $(\text{SUCCESS}, M'_i)$, except with negligible probability, M'_i contains exactly N of the same messages, includes each honest member's message, and has the messages in the same order.*

In Section 5.3.1 we provide a proof that the GMP-SHUFFLE protocol maintains integrity. Section 5.3.2 contains a proof for the GMP-BULK protocol. The proofs are structured as follows. First, we show that a protocol run can either succeed or fail for each honest i. Then, we show that each honest i who succeeds obtains a same set M'_i of exactly N messages that includes every honest member's message.

5.3.1 The GMP-Shuffle Protocol

We will show that the GMP-SHUFFLE protocol terminates either with success or failure, depending on the outcome of the verification in Phase 4 and the key release and decryption in Phase 5. If both phases complete successfully, then member i recovers secret messages submitted to the protocol and the protocol completes outputting $(\text{SUCCESS}, M'_i)$. If any step of Phase 4 or 5 fails, then member i outputs $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$ after executing the blame procedures in Phase 6.

Lemma 1. *After a complete run of* GMP-SHUFFLE*, each honest group member i terminates with either $(\text{SUCCESS}, M'_i)$ or $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$.*

Proof. After running ANONYMIZE-S, each honest member i's internal SUCCESS_i flag is set to either TRUE or FALSE indicating the outcome of the protocol from i's point of view. Member i has $\text{SUCCESS}_i = \text{TRUE}$ only if in Phase 4 she has a "go" message and receives a complete set of "go" messages and matching broadcast hashes from every member, and in Phase 5 she receives a complete set of non-empty and matching inner private keys from every member. Otherwise, member i's flag is set to FALSE.

For every honest member i, ANONYMIZE-S outputs $(\text{SUCCESS}, M'_i)$ if $\text{SUCCESS}_i = \text{TRUE}$ and $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$ if $\text{SUCCESS}_i = \text{FALSE}$. Hence, each protocol run of GMP-SHUFFLE terminates with either $(\text{SUCCESS}, M'_i)$ or $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$ for every honest group member i. □

Lemma 2. *For every honest member i who terminates with $(\text{SUCCESS}, M_i')$ after running* GMP-SHUFFLE*, except with negligible probability, M_i' includes the same N messages, includes each honest member's message, and has the messages in the same order.*

Proof. Let i be an honest member for whom the protocol run succeeds. According to the protocol specification, i terminated with $(\text{SUCCESS}, M_i')$ because (*i*) in Phase 4 her own $\text{GO}_i = \text{TRUE}$, (*ii*) in Phase 4 she receives messages such that $\text{GO}_j = \text{TRUE}$ and $\text{HASH}\{\vec{B}_j\} = \text{HASH}\{\vec{B}_i\}$ for every member $j \in G$, and (*iii*) in Phase 5 she received non-empty inner private keys such that I_j^{sec} matched I_j^{pub} for every $j \in G$.

$\vec{B}_i$ contains all broadcast messages member i sent and received in Phases 1–3, and thus, by (*i*) and (*ii*) and the assumption that the hash function is second-preimage resistant, member i is in possession of the same $\vec{C}_N$ and inner public keys as every other honest member j, except with negligible probability. Furthermore, (*iii*) applies to every honest j for which the protocol is successful, and so every such j has inner private keys that match the common inner public keys.

Thus, member i can decrypt each ciphertext included in $\vec{C}_N$ using her set of inner private keys to obtain N messages, and the resulting list contains the same messages in the same order as each honest user j that successfully terminates. Moreover, because member j sends i $\text{GO}_j = \text{TRUE}$, the inner ciphertext C_j' must be in their common $\vec{C}_N$. Therefore, after decryption, i obtains the message m_j of each honest member j. □

Theorem 1. *The* GMP-SHUFFLE *protocol offers integrity.*

Proof. Following Lemma 1 we know that each honest group member i terminates with either $(\text{SUCCESS}, M_i')$ or $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$ after a complete protocol run of GMP-SHUFFLE. Following Lemma 2 we know that, for every honest member who terminates with $(\text{SUCCESS}, M_i')$, except with negligible probability, M_i' contains the same N messages in the same order, including each honest member's message. Thus the GMP-SHUFFLE protocol offers integrity. □

5.3.2 The GMP-Bulk Protocol

We will show that the GMP-BULK protocol terminates either with success or failure, depending on the outcome of the shuffle in Phase 3 and Phase 7. If ANONYMIZE-S succeeds in Phase 3, all ciphertext C_{ij}' are correct, and ANONYMIZE-S succeeds in Phase 7, or if there is no valid accusation for each C_{ij}' that is incorrect after ANONYMIZE-S in Phase 7 succeeds, then the protocol completes successfully outputting $(\text{SUCCESS}, M_i')$. Otherwise, the protocol fails, member i executes the blame procedures and outputs $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$.

Lemma 3. *After a complete run of* GMP-BULK*, each honest group member i terminates with either $(\text{SUCCESS}, M_i')$ or $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$.*

Proof. After running ANONYMIZE-B, each honest member i's internal SUCCESS_i flag is set to either TRUE or FALSE indicating the outcome of the protocol from i's point of view. Member i has $\text{SUCCESS}_i = \text{TRUE}$ only if in Phase 4 she receives a correct and complete set of ciphertexts C_{jk}' for every $k \in G$ and the ANONYMIZE-S protocol succeeds in Phase 7, or there is no valid accusation in Phase 7 for every incorrect ciphertext C_{jk}' received in Phase 4 following a successful run of the ANONYMIZE-S protocol in Phase 7. Otherwise, member i's flag is set to FALSE.

For every honest member i, ANONYMIZE-B outputs $(\text{SUCCESS}, M_i')$ if $\text{SUCCESS}_i = \text{TRUE}$ and $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$ if $\text{SUCCESS}_i = \text{FALSE}$. Hence, each protocol run of GMP-BULK terminates with either $(\text{SUCCESS}, M_i')$ or $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$. □

Lemma 4. *For every honest member i who terminates with* $(\text{SUCCESS}, M_i')$ *after running* GMP-BULK*, except with negligible probability,* M_i' *includes the same* N *messages,* M_i' *includes each honest member's message, and the messages in* M_i' *are in the same order.*

Proof. Assume that there exists an honest member i for whom GMP-BULK terminates successfully. Then, according to the protocol specification, it must be that (*i*) each member $k \in G$ sends i $\text{GO}_k = \text{TRUE}$ in Phase 4, (*ii*) the run of the ANONYMIZE-S protocol completes successfully for i in Phase 7, and (*iii*) either $\text{HASH}\{C'_{jk}\} = H_{jk}$ for all ciphertexts received by i in Phase 4 or no valid accusation is received in Phase 7 for any ciphertext such that $\text{HASH}\{C'_{jk}\} \neq H_{jk}$.

The descriptor and blame shuffles are executed by calling ANONYMIZE-S using the parameters produced by SETUP-B. These parameters are produced in the same way that SETUP-S does as part of GMP-SHUFFLE, and therefore Theorem 1 applies to the descriptor and blame shuffles. Thus every honest member for whom the descriptor shuffle is successful, except with negligible probability, obtains the same N message descriptors in the same order, including a message descriptor for each honest member. By (*i*), the descriptor shuffle is successful for every honest member, and thus they all obtain these same descriptors. Similarly, every honest member for whom the blame shuffle is successful obtains the same N accusations in the same order, including each accusation from an honest member. By (*ii*), the blame shuffle is successful for every honest member for whom the bulk protocol is successful, and thus they all obtain these same accusations.

Therefore, if honest members receive different ciphertexts in Phase 4, the second-preimage resistance of the hash implies that at least one of the ciphertexts must not match the corresponding hash. The recipient of that ciphertext would report the corruption in Phase 5, and the equivocation would prevent the accusation shuffle from succeeding for any honest member, contradicting (*ii*).

Thus all honest members that successfully terminate must have the same sequence of N descriptors and the same ciphertexts. This implies that these members obtain the same N messages in the same order from the bulk protocol.

In addition, as shown, the descriptors obtained by every honest member include the descriptors of all of the honest members in the same slots. Because each honest member receives the same ciphertexts, any corruption of an honest member's slot would be seen by that member. That member would then produce an accusation which, as we have described, would be obtained from the blame shuffle by all honest members who terminate successfully. This would contradict condition (*iii*) of successful termination. Therefore, no slot containing an honest member's descriptor can be corrupted at an honest user. This implies that the messages obtained by an honest member from successful termination of the bulk protocol must contain the messages of all honest members. □

Theorem 2. *The* GMP-BULK *protocol offers integrity.*

Proof. Following Lemma 3 we know that each honest group member i terminates with either $(\text{SUCCESS}, M_i')$ or $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$ after a complete protocol run of GMP-BULK. Following Lemma 4 we know that, for every honest member who terminates with $(\text{SUCCESS}, M_i')$, except with negligible probability, M_i' contains the same N messages in the same order, including each honest member's message. Thus the GMP-BULK protocol offers integrity. □

5.4 Accountability

Definition 14. *A Group Messaging Protocol* GMP *offers* accountability *if, after a complete protocol run,*

1. *the* BLAME_i *set of any honest member* i *for whom the protocol failed is non-empty,*
2. *no honest member is exposed, except with negligible probability, and*
3. *an honest member exposes every member she blames.*

These properties must hold even when the protocol run is preceded by other protocol runs.

In Section 5.4.1 we prove that the GMP-SHUFFLE protocol offers accountability. Section 5.4.2 contains a corresponding proof for the GMP-BULK protocol.

The checks of each protocol form the backbone of each proof. A main argument of the proofs is that the protocol fails when one of the checks fails, each such failure for i results in an addition to BLAME_i, and because VERIFY-PROOF uses the same checks each such addition exposes the blamed member. In addition, the round nonces, phase numbers, and member identities included in each signed message prevent an adversary from creating a log that contains anything but the actual messages sent by an honest member in a given round and phase. The protocols ensure that these sent messages include the messages received by the honest member where necessary. Thus an honest member is always seen in the log as behaving correctly and is not exposed.

5.4.1 The GMP-Shuffle Protocol

Lemma 5. *If, after a complete run of* GMP-SHUFFLE*,* $\text{SUCCESS}_i = \text{FALSE}$ *for an honest member* i*, then* BLAME_i *is non-empty, and every proof it contains is verifiable given log* ℓ_i*.*

Proof. We will show that, whenever $\text{SUCCESS}_i = \text{FALSE}$, i adds a proof p_j to BLAME_i, and every proof it adds is verifiable. In fact, it suffices to show that, whenever $\text{SUCCESS}_i = \text{FALSE}$, i adds a proof p_j to BLAME_i, because it is straightforward to see that any such p_j is verifiable. In VERIFY-PROOF-S, proof verification of p_j (Step 1) always succeeds, because p_j always includes valid check number and member identifier; log verification of ℓ_i (Step 2) always succeeds because the protocol completes by assumption, and i adds all her messages to log ℓ_i; and the proof verification decision (Step 3) always succeeds because it outputs TRUE given p_j for exactly the same logs in which i adds p_j to BLAME_i.

Therefore, we can simply show that, whenever the protocol fails for i, a proof is added to BLAME_i. In ANONYMIZE-S, $\text{SUCCESS}_i = \text{FALSE}$ upon protocol completion only in the following three cases: (1) in Phase 4, $\text{GO}_i = \text{FALSE}$ or a non-matching broadcast hash is received, (2) in Phase 4, $\text{GO}_k = \text{FALSE}$ for some $k \neq i$, (3) in Phase 5, an empty, invalid, or non-matching inner private key is received. In any of these cases, if an inconsistent or incomplete $\vec{T}$ log is received in some μ_{j6}, then $(j, \mathtt{c}_1)$ is added to BLAME_i. Therefore we assume from this point on that all $\vec{T}$ logs are complete and consistent and proceed to examine these cases separately.

Suppose case (1) occurs. We consider the conditions in each of the phases up to Phase 4 that can cause $\text{GO}_i = \text{FALSE}$, and we identify in each case a proof p_j that must be added to BLAME_i:

- In Phase 1, an invalid public key must be received from some j. Then $p_j = (j, \mathsf{c}_5)$.
- In Phase 2a, an invalid commitment must be received from some j. Then $p_j = (j, \mathsf{c}_6)$.
- In Phase 2b, a commitment opening must fail or result in an invalid ciphertext or identity. Then $p_j = (j, \mathsf{c}_7)$.
- In Phase 3, $\vec{C}_i$ must have an invalid or duplicate ciphertext. If some member j releases an empty, invalid, or non-matching outer private key in Phase 6, then $p_j = (j, \mathsf{c}_4)$. Otherwise, i replays the permutations and decryptions of Phase 3. During the replay, if some member j did not correctly permute and decrypt her inputs, then $p_j = (j, \mathsf{c}_8)$. Otherwise, i must observe a member j whose commitment value decrypted either to an invalid ciphertext, in which case $p_j = (j, \mathsf{c}_9)$, or to a duplicate ciphertext, in which case $p_j = (j, \mathsf{c}_{10})$.
- In Phase 4, it could be that the inner ciphertext C'_j is not in $\vec{C}_N$. In this case, as in the previous one, if some member j releases an empty, invalid, or non-matching outer private key in Phase 6, then $p_j = (j, \mathsf{c}_4)$. Otherwise, i replays Phase 3 and during the replay must observe some member j who did not correctly permute and decrypt her inputs. Then $p_j = (j, \mathsf{c}_8)$. It could also be that a non-matching broadcast hash is received from j, in which case j must have sent an incorrect hash, and $p_j = (j, \mathsf{c}_{12})$.

Next suppose case (2) occurs. If some member j releases an empty, invalid, or non-matching outer private key in Phase 6, then $p_j = (j, \mathsf{c}_4)$. Otherwise, i replays the protocol. If any member j sent an invalid public key or an invalid commitment, then $p_j = (j, \mathsf{c}_5)$ or $p_j = (j, \mathsf{c}_6)$, respectively. If $k = 1$ and commitment opening failed or resulted in an invalid ciphertext for some j, then $p_j = (j, \mathsf{c}_7)$. If there were invalid or duplicate ciphertexts in $\vec{C}_k$, then i must observe a member j who either did not correctly permute and decrypt her inputs, in which case $p_j = (j, \mathsf{c}_8)$, or committed to a value that decrypted to an invalid or duplicate ciphertext, in which case $p_j = (j, \mathsf{c}_9)$ or $p_j = (j, \mathsf{c}_{10})$, respectively. If the inner ciphertext of member k is not included in $\vec{C}_N$, then there must be some member j who did not correctly permute and decrypt her inputs, and $p_j = (j, \mathsf{c}_8)$. Otherwise, k incorrectly set GO_k, and $p_j = (j, \mathsf{c}_{11})$ with $j = k$.

Finally, suppose case (3) occurs. An empty inner private key can only be justified by a $\text{GO}_k =$ FALSE for some k or a non-matching broadcast hash from some j. In either case we have already identified the p_j added by i. If an empty key from some j is not justified, then $p_j = (j, \mathsf{c}_3)$. If an invalid or non-matching inner private key is received from some j, then $p_j = (j, \mathsf{c}_2)$.

Thus we have shown that honest member i adds some proof p_j to BLAME_i whenever $\text{SUCCESS}_i =$ FALSE, and furthermore that any such p_j is a verifiable proof given log ℓ_i. □

Lemma 6. *An honest member j is not exposed after a run of* GMP-SHUFFLE*, except with negligible probability.*

Proof. Suppose that the adversary exposes an honest member j. That is, suppose that he produces a proof p_j and log ℓ_i such that $\text{VERIFY-PROOF-S}(p_j, \ell_i) = \text{TRUE}$. To pass the initial proof verification, it must be the case that $p_j = (c, j)$. To pass the log verification, it must be the case either that $c = \mathsf{c}_1$ or that all the $\vec{T}$ logs in the μ_{j6} of ℓ_i are complete and consistent.

Each message in ANONYMIZE-S identifies the sender and is signed by that sender. By the EUF-CMA property of the signature scheme, the adversary is not able to forge a signature under any honest member's key, except with negligible probability, and therefore any message signed by j in ℓ_i must have been sent by j. Furthermore, each message identifies the round and phase for which that message was sent. An honest member sends exactly one message during each phase of a given round. Therefore, every message in ℓ_i from j must have actually been sent during that round and phase by j.

Given these facts, we can go through each possible check and show that for each one the needed log evidence cannot exist. Whenever we refer to message $\mu_{k\phi}$, we are referring to the message that ℓ_i indicates was sent by member k in phase ϕ.

Suppose that $c = \mathsf{c}_1$. Then for the proof to verify, ℓ_i must contain either different copies of the same message for a given phase or an incomplete log $\vec{T}$ in a μ_{j6}. An honest j would never send such messages. Thus $c \neq \mathsf{c}_1$.

In each of the remaining cases, the log vectors $\vec{T}$ in the μ_{k6} were verified during log verification to be complete and consistent, and ℓ_i is augmented with all messages from all members during Phases 1–5. Thus we can assume that each message $\mu_{k\phi}$ sent or received by j during these phases appears with the same contents in ℓ_i.

Suppose that $c = \mathsf{c}_2$. Then it must be the case that μ_{j1} and μ_{j5} have non-matching I_j^{pub} and I_j^{sec}. j would never send such a pair, however. Thus $c \neq \mathsf{c}_2$.

Suppose that $c = \mathsf{c}_3$. Then j must have sent an empty inner key, which implies that j observed either a $\text{GO}_k = \text{FALSE}$ or a non-matching broadcast hash $\text{HASH}\{\vec{B}_k\}$. Therefore the μ_{k4} do not contain the evidence needed for VERIFY-PROOF-S to validate this check. Thus $c \neq \mathsf{c}_3$.

Suppose that $c = \mathsf{c}_4$. Then either j sent outer keys O_j^{pub} and O_j^{sec} that do not match, or j incorrectly sent an empty outer private key. j only ever sends matching outer keys, and so the former case cannot apply. If j sent an empty outer private key, it must have been the case that, for all μ_{k4}, the contained $\text{GO}_k = \text{TRUE}$ and $\text{HASH}\{\vec{B}_k\} = \text{HASH}\{\vec{B}_j\}$. Therefore the μ_{k4} do not contain the evidence needed for VERIFY-PROOF-S to validate this check. Thus $c \neq \mathsf{c}_4$.

Suppose that $c = \mathsf{c}_5$. Then j must have sent an invalid key in μ_{j1}. An honest j would never send an invalid key, though, and thus $c \neq \mathsf{c}_5$.

Suppose that $c = \mathsf{c}_6$. The j must have sent an invalid commitment in μ_{j2a}. An honest j would never send an invalid commitment, though, and thus $c \neq \mathsf{c}_6$.

Suppose that $c = \mathsf{c}_7$. Then either j's commitment opening in μ_{j2b} does not match the commitment in μ_{j2a}, or the value from the opening is not a valid ciphertext or identity. j always sends a matching commitment and opening, though, and j's committed value is always a valid ciphertext and her identity. Thus $c \neq \mathsf{c}_7$.

Suppose that $c = \mathsf{c}_8$. Then the messages in μ_{j3} must not be a permutation and decryption of the messages in $\mu_{(j-1)3}$ using the key O_j^{sec} released by j. However, j does correctly permute and decrypt during Phase 3 and only ever releases the correct key used in that decryption. Thus $c \neq \mathsf{c}_8$.

Suppose that $c = \mathsf{c}_9$. Then j must send a value C_j into the Phase 3 shuffle that results in an invalid ciphertext after some sequence of decryptions by the outer private keys released by all members. Those private keys are checked to match the outer public keys received by j, however, and j correctly forms C_j by encrypting m_j with the inner and outer public keys in sequence. Therefore it can never be that C_j results in an invalid ciphertext after decryption by some of the outer private keys, and $c \neq \mathsf{c}_9$.

Suppose that $c = \mathsf{c}_{10}$. Then it must be that for some ciphertext C_k, $k \neq j$, both C_k and C_j yield the same result after some number of sequential decryptions by the outer private keys. As we established above, the messages in Phases 1–5 of ℓ_i sent and received by j are those actually sent and received by j during the protocol run. Thus, if the adversary were able to produce a commitment to a value that is related to C_j in that some sequential decryptions yield the same result, then we could construct an adversary that violates the non-malleability of the commitment scheme (Dolev, Dwork, and Naor 2000). Thus $c \neq \texttt{c}_{10}$.

Suppose that $c = \texttt{c}_{11}$. Then it must be that j sent $\text{GO}_j = \text{FALSE}$ without justification. The justification needed would be receiving an invalid public key in Phase 1, receiving an invalid commitment in Phase 2a, receiving an invalid commitment opening or opening an invalid ciphertext or identity in Phase 2b, producing invalid or duplicate ciphertexts during Phase 3, or not receiving her own inner ciphertext C'_j at the end of Phase 3. However, each of these conditions is true in ℓ_i if it was true during the run from j's perspective. In particular, the inner ciphertext as determined by VERIFY-PROOF-S must be the inner ciphertext of j because the decryption keys are verified to match the public keys seen by j. j would only send $\text{GO}_j = \text{FALSE}$ if one of these conditions held, and thus $c \neq \texttt{c}_{11}$.

Suppose that $c = \texttt{c}_{12}$. Then it must be that the broadcast hash that j sent in Phase 4 does not match the hash of all broadcast messages up to that point. j sends the correct hash, however, and thus $c \neq \texttt{c}_{12}$.

Therefore, there is no value of c for which VERIFY-PROOF-S could output TRUE given ℓ_i, except with negligible probability, and the adversary cannot expose an honest member. □

Theorem 3. *The* GMP-SHUFFLE *protocol offers accountability.*

Proof. Following Lemma 5 we know that after a failed run of GMP-SHUFFLE for an honest member i, BLAME_i is non-empty. Additionally, every proof included in BLAME_i is verifiable given a log ℓ_i, hence, an honest member exposes every member she blames. Following Lemma 6 we know that an honest member j is not exposed after a run of GMP-SHUFFLE, except with negligible probability. Thus the GMP-SHUFFLE protocol offers accountability. □

5.4.2 The GMP-Bulk Protocol

Lemma 7. *If, after a complete run of* GMP-BULK, $\text{SUCCESS}_i = \text{FALSE}$ *for an honest member i, then* BLAME_i *is non-empty, and every proof it contains is verifiable given log ℓ_i.*

Proof. We will show that, whenever $\text{SUCCESS}_i = \text{FALSE}$, i adds a proof p_j to BLAME_i, and every proof it adds is verifiable. In fact, it will suffice to show that, whenever $\text{SUCCESS}_i = \text{FALSE}$, i adds a proof p_j to BLAME_i, because we first prove that any such p_j is verifiable.

In VERIFY-PROOF-B, proof verification of p_j (Step 1) always succeeds, because honest i always includes a valid check number and member identifier in p_j. Log verification of ℓ_i (Step 2) always succeeds because the protocol completes by assumption, and i adds all her messages to log ℓ_i. Finally, given complete log ℓ_i, the properties of that log that must hold for the proof verification decision (Step 3) to output TRUE on proof p_j are almost exactly the same properties that must hold for honest i to add p_j to BLAME_i. In fact, VERIFY-PROOF-B only verifies as true more proofs for a given log than would be created by i, as we show by considering each check separately:

- $p_j = (j, \texttt{c}_1)$: VERIFY-PROOF-B omits checking for $(j, \texttt{c}_{11}) \in \text{BLAME}_i^{s2}$ and otherwise makes the same log checks to verify p_j as ANONYMIZE-B does during blame to produce p_j.

- $p_j = (j, \texttt{c}_2)$: VERIFY-PROOF-B and ANONYMIZE-B use the same log checks for this p_j.

- $p_j = (j, \mathsf{c}_3)$: VERIFY-PROOF-B adds a check to make sure that the descriptors claimed by i are those received by j, but this check is always satisfied by the log of an honest i. All other checks are the same for this p_j.
- $p_j = (j, \mathsf{c}_4)$: VERIFY-PROOF-B omits checking that the blame shuffle succeeds and that $\vec{V}_i$ contains some $\text{GO}_k = \text{FALSE}$. Otherwise, it is the same as ANONYMIZE-B for this p_j.
- $p_j = (j, \mathsf{c}_5)$: VERIFY-PROOF-B omits checking that the blame shuffle succeeds, that $\vec{V}_i$ contains some $\text{GO}_k = \text{FALSE}$, and that the member with evidence of a bad key gets blamed first. Otherwise, it the same as ANONYMIZE-B for this p_j.
- $p_j = (j, \mathsf{c}_6)$: VERIFY-PROOF-B omits checking that the blame shuffle succeeds, that $\vec{V}_i$ contains some $\text{GO}_k = \text{FALSE}$, and that the member with equivocation evidence gets blamed first. Otherwise, it the same as ANONYMIZE-B for this p_j.

Thus, VERIFY-PROOF-B$(p_j, \ell_i) = \text{TRUE}$ for every $p_j \in \text{BLAME}_i$.

Therefore, we can simply show that, whenever the protocol fails for i, a proof is added to BLAME_i. In ANONYMIZE-B, $\text{SUCCESS}_i = \text{FALSE}$ upon protocol completion only in the following cases:

1. The blame shuffle fails.
2. The blame shuffle succeeds and outputs a valid accusation.
3. Some μ_{j4} contains $\text{GO}_j = \text{FALSE}$.

We consider each case and identify a proof p that is added to BLAME_i in each one.

In case (1), by Lemma 5, there exists a verifiable proof $(j, c) \in \text{BLAME}_i^{s2}$ given ℓ_i^{s2}. If $c = \mathsf{c}_{11}$ and evidence of ciphertext equivocation by k exists in μ_{j7}, then $p = (k, \mathsf{c}_1)$. Otherwise, $p = (j, \mathsf{c}_2)$. In case (2), $p = (j, \mathsf{c}_3)$. In case (3), $p = (j, \mathsf{c}_4)$ if μ_{j4} contains no verifiable proofs, $p = (k, \mathsf{c}_2)$ if μ_{j4} has a verifiable proof of k's misbehavior and k provides no justification in μ_{k3}, and $p = (\ell, \mathsf{c}_5)$ or $p = (\ell, \mathsf{c}_6)$ if μ_{j4} has a verifiable proof of k's misbehavior but k provide evidence against ℓ in μ_{k3}.

Thus, if GMP-BULK fails for i, BLAME_i contains a verifiable proof given ℓ_i and only contains such proofs. □

Lemma 8. *An honest member j is not exposed after a run of* GMP-BULK*, except with negligible probability.*

Proof. Suppose that the adversary exposes an honest member j. To pass the proof verification of VERIFY-PROOF-B, it must be the case that he produces a proof $p_j = (c, j)$. To pass the log verification, it must be the case the log ℓ_i is complete.

Each message in ANONYMIZE-B identifies the sender and is signed by that sender. By the assumption the signature scheme is EUF-CMA, the adversary is not able to forge a signature under any honest member's key, except with negligible probability, and therefore any message signed by j in ℓ_i must have been sent by j. Furthermore, each message identifies the round and phase for which that message was sent. An honest member sends at most one message during each phase of a given round. Therefore, every message in ℓ_i from j must have actually been sent during that round and phase by j.

Given these facts, we can go through each possible check and show that for each one the needed log evidence cannot exist. Whenever we refer to message $\mu_{k\phi}$, we are referring to the message that ℓ_i indicates was sent by member k in phase ϕ.

Suppose that $c = \mathtt{c}_1$. Then for the proof to verify, ℓ_i must contain different copies of the same message for Phase 4. An honest j always sends the same message to every member in any given phase and therefore such messages do not exist. Thus $c \neq \mathtt{c}_1$.

Suppose that $c = \mathtt{c}_2$. Then ANONYMIZE-S must have failed in Phase 3 or Phase 7.

If ANONYMIZE-S failed in Phase 3, then for the proof to verify member j must have not distributed a proof of another member's bad key or key equivocation, and there must be a verifiable $p_j \in \text{BLAME}_k^{s1}$ for some member k. However, if j intentionally causes a failure, then she always distributes an appropriate proof in μ_{j3}, and if she does not, then by Lemma 6 a verifiable proof blaming j cannot be produced, except with negligible probability.

If ANONYMIZE-S failed for i in Phase 7, then for the proof to verify member j must have not distributed a proof another's member equivocation in Phase 4, and $p_j \in \text{BLAME}_i^{s2}$ must be verifiable. However, if j causes a failure of the blame shuffle, then she always distributes a proof of equivocation in μ_{j7}, and if she does not, then by Lemma 6 a verifiable proof blaming j cannot be produced, except with negligible probability. Thus $c \neq \mathtt{c}_2$.

Suppose that $c = \mathtt{c}_3$. Then j must have sent an incorrect or empty ciphertext in Phase 4.

Observe that the hash of broadcast messages in ℓ_i is verified to be equal to the broadcast hash sent by j, and thus, by the second-preimage resistance property, it must be that the inner public keys and inner ciphertexts in ℓ_i are the same as those seen by j, except with negligible probability. The inner private keys are verified to match their public keys, and thus the descriptors computed by VERIFY-PROOF-B must match those seen by j.

An honest j would only send a non-empty ciphertext C'_{kj} if the pseudorandom bits from its decrypted seed yield the correct hash value. Given that the computed descriptors match those seen by j and that only one seed can encrypt to a given ciphertext, the accusation must not satisfy the validity checks in VERIFY-PROOF-B.

If j sends an empty ciphertext C'_{kj}, then it must be that due to a problem with descriptor d_k that she observed. That is, it must be that S_{kj} is not a valid ciphertext, s_{kj} is not a valid seed, or $\text{HASH}\{C_{kj}\} \neq H_{kj}$. If any of the above descriptor problems exist, then because the descriptors used in VERIFY-PROOF-B must match the ones seen by j, the accusation must not satisfy the validity checks in VERIFY-PROOF-B. Thus $c \neq \mathtt{c}_3$.

Suppose that $c = \mathtt{c}_4$. Then it must be that j sent $\text{GO}_j = \text{FALSE}$ in μ_{j4} without justification. The justification needed either would be evidence in μ_{j3} of a bad key or key equivocation in Phase 1a or would be a verifiable proof in μ_{j4} of misbehavior during ANONYMIZE-S in Phase 3.

If j sent $\text{GO}_j = \text{FALSE}$ in μ_{j4}, it must have been that the descriptor shuffle failed for j. If j intentionally caused this shuffle to fail, then j observed bad or non-matching keys and distributed the evidence in μ_{j3}. If j did not intentionally cause shuffle failure, then by Lemma 5, BLAME_j^{s1} contains a verifiable proof given ℓ_j^{s1}. Thus $c \neq \mathtt{c}_4$.

Suppose that $c = \mathtt{c}_5$. Then j must have sent an invalid key in μ_{j1a}. An honest j would never send an invalid key, though, and thus $c \neq \mathtt{c}_5$.

Suppose that $c = \mathtt{c}_6$. Then for the proof to verify, ℓ_i must contain different copies of the same message for Phase 1a. However, an honest j always sends the same message to every member in any given phase. Thus $c \neq \mathtt{c}_6$.

Therefore, there is no value of c for which VERIFY-PROOF-B could output TRUE given ℓ_i, except with negligible probability, and the adversary cannot expose an honest member. □

Theorem 4. *The* GMP-BULK *protocol offers accountability.*

Proof. Following Lemma 7 we know that after a failed run of GMP-BULK for an honest member i, BLAME_i is non-empty. Additionally, every proof included in BLAME_i is verifiable given a log ℓ_i, hence, an honest member exposes every member she blames. Following Lemma 8 we know that an honest member j is never exposed after a run of GMP-BULK, except with negligible probability. Thus the GMP-BULK protocol offers accountability. □

5.5 Anonymity

Definition 15. *A protocol maintains* anonymity *with k colluding members if, for all probabilistic polynomial-time adversaries, the advantage in the anonymity game with any k dishonest members is negligible.*

Note that the definition will only make sense for $0 \leq k \leq N-2$. We use the anonymity game described by Brickell and Shmatikov (2006a). The anonymity game is played between an adversary A and a challenger $C(b)$, where b denotes a hidden challenge bit. The adversary plays the roles of k dishonest members, while the challenger plays the role of the $N-k$ honest members.

The *anonymity game* works as follows:

1. As many times as A requests, $C(b)$ takes message inputs for the honest members from A and uses them to execute the protocol with A, giving him a copy of every message sent.
2. A chooses two honest participants α and β and two message inputs m_0^c and m_1^c. He also chooses message inputs m_h for each honest member h and sends his choices to $C(b)$.
3. $C(b)$ assigns $m_\alpha = m_b^c$ and $m_\beta = m_{\bar{b}}^c$.
4. A and $C(b)$ execute the protocol, during which $C(b)$ gives A a copy of every message sent.
5. As many times as A requests, $C(b)$ takes message inputs for the honest members and uses them to execute the protocol with A, giving him a copy of every message sent.
6. The adversary outputs a guess $\hat{b} \in \{0,1\}$ for the value of b.

The adversary's *advantage* in the anonymity game is equal to

$$\left| Pr\left[A^{C(0)} = 1\right] - Pr\left[A^{C(1)} = 1\right] \right|,$$

where the probability is taken over the randomness of both the adversary and of the challenger.

5.5.1 The GMP-Shuffle Protocol

We consider the anonymity game running GMP-SHUFFLE and show that the adversary's advantage in winning this game is negligible.

We begin by using any adversary A to construct *Game 0*, in which a new challenger C^0 randomly guesses whether a given honest user will release her outer private key during the final phases of the protocol. When C^0 guesses correctly, he behaves exactly as C would in the anonymity game and the game ends with the output of A. When C^0 guesses incorrectly, the game output is a random bit. C^0 guesses independently of A, and so we will be able to show that the game output's advantage in Game 0 is $1/2$ the advantage of A in the anonymity game.

Then we define *Game 1*, in which a further modified challenger C^1 creates the inner or outer ciphertexts of α by starting with a plaintext unrelated to the challenge message m_b. We will be able to show that advantage in Game 1 is negligibly close to the advantage in Game 0 by showing how a non-negligible change in advantage would allow us to distinguish encrypted messages with non-negligible probability.

Finally, we define *Game 2* by creating a challenger C^2 from C^1 in the same way that C^1 was created from C^0, except replacing α by β and m_b with $m_{\bar{b}}$ in the changes. We can show that the advantage changes negligibly from Game 1 to Game 2 using a similar argument as used from Game 0 to Game 1. It will be the case that the advantage in Game 2 must be 0 because the adversary sees the same distribution of messages from the challenger regardless of the challenge bit.

Let $h_1, h_2, \ldots, h_{N-k}$ be the honest users in the order they appear in the shuffle. Let Z^i indicate that the challenger C^i guesses that h_1 should release her outer private key at some point as part of ANONYMIZE-S, let G^i be a "game output" for Game i, and let F^i indicate whether or not the challenger failed in Game i. In the games and their associated random variables, the challenge bit b is a hidden input that we generally omit.

Game 0: In this game, A interacts with a challenger C^0 that sometimes fails. C^0 sets $Z^0 \in \{0, 1\}$ uniformly at random. C^0 behaves the same as C except in the following cases of the challenge shuffle, when his guess about which keys will be released proves to be incorrect:

1. In Phase 3 (Anonymization), $Z^0 = 0$ and the partial decryptions of the outer ciphertexts C_α and C_β with keys $I_1^{sec} \ldots, I_{h_1-1}^{sec}$ do not appear exactly once each in the ciphertext vector $\vec{C}_{h_1-1}$ sent to h_1. C^0 can check this by comparing to the partial ciphertexts created during Phase 2a.

2. In Phase 4 (Verification), $Z^0 = 0$ and either of the inner ciphertexts C'_α and C'_β is missing either from the copy of vector $\vec{C}_N$ sent to α or from the copy sent to β. Again, C^0 can notice this by comparing to inner ciphertexts created during Phase 2a.

3. In Phase 5 (Key Release and Decryption), $Z^0 = 1$ and member h_1 receives $\text{GO}_j = \text{TRUE}$ and $\text{HASH}\{\vec{B}_j\} = \text{HASH}\{\vec{B}_{h_1}\}$ for every member $j \neq h_1$, and $\text{GO}_{h_1} = \text{TRUE}$.

4. In Phase 6 (Blame), $Z^0 = 0$ and *i)* $\text{GO}_{h_1} = \text{FALSE}$, *ii)* h_1 received $\text{GO}_j = \text{FALSE}$ from any member j, or *iii)* h_1 received $\text{HASH}\{\vec{B}_j\} \neq \text{HASH}\{\vec{B}_{h_1}\}$ from any member j.

In each of these cases, $F^0 = 1$, C^0 terminates, and the game output G^0 is set to a uniformly random bit. In every other case, $F^0 = 0$, C^0 correctly executes ANONYMIZE-S on behalf of the honest users, and G^0 is set to the output bit of A.

Game 1: In this game, we further modify the challenger to define C^1, which replaces with unrelated ciphertexts the intermediate stages of the construction of the inner or outer ciphertext of α, depending on Z^1. That is, C^1 behaves the same as C^0, except

1. In Phase 2a,

Case 1: $Z^1 = 0$. A partially encrypted outer ciphertext for α is created and stored as $C''_\alpha = \{\{\alpha\}_{I_N^{pub}:I_1^{pub}}\}_{O_N^{pub}:O_{h_1}^{pub}}$, and the outer ciphertext is then created as $C_\alpha = \{C''_\alpha\}_{O_{h_1-1}^{pub}:O_1^{pub}}$. Also create $C'_\alpha = \{m_b\}_{I_N^{pub}:I_1^{pub}}$ for later use. The public keys used for each ciphertext of α are those received by α in Phase 1.

Case 2: $Z^1 = 1$. The inner ciphertext for α is created and stored as $C'_\alpha = \{\alpha\}_{I_N^{pub}:I_1^{pub}}$, and the outer ciphertext C_α is created from C'_α in the same way as C^0. Again, the public keys used for each ciphertext of α are those received by α in Phase 1.

The rest of the phase is executed in the same way as C^0.

2. In Phase 3, if $Z^1 = 0$ and both the stored ciphertext C''_α and the partial decryption of C_β by $I_N^{sec} \ldots, I_{h_1-1}^{sec}$ (which C^1 knows because it created C_β) appear exactly once each in the vector of ciphertexts $\vec{C}_{h_1-1}$ sent to h_1, then replace C''_α with $\{C'_\alpha\}_{O_N^{pub}:O_{h_1+1}^{pub}}$ for inclusion in the vector $\vec{C_{h_1}}$ sent to $h_1 + 1$, where the encryption uses the outer keys sent to α.

   In every other way, C^1 executes in the same way as C^0.

Game 2: This game is created from Game 1 using the same changes given in its definition, except replacing α with β and m_b with $m_{\bar{b}}$ everywhere.

The following lemma shows that Game 0 is a relevant starting point because its output's advantage is $1/2$ the advantage of A in the anonymity game:

Lemma 9.

$$\Delta(G^0) = \frac{1}{2}\left|Pr\left[A^{C(0)} = 1\right] - Pr\left[A^{C(1)} = 1\right]\right|,$$

where the probability is taken over the randomness of both the adversary and the challenger.

Proof. Let $T_{A,C}$ be the set of all possible game *transcripts*, that is, sequences of messages, between A and C. Let T_{A,C^0} be the set of transcripts between A and C^0. We claim that each member of $T_{A,C}$ and T_{A,C^0} falls into exactly one of following cases, which are nearly the same as the failure cases defining Game 0:

1. In Phase 3, the expected ciphertexts of α and β are not sent to h_1 exactly once each.

2. Case 1 does not occur, and in Phase 4, either of the inner ciphertexts C'_α and C'_β is missing from either the copy of vector $\vec{C}_N$ sent to α or the copy sent to β.

3. At the start of Phase 5, $\text{GO}_{h_1} = \text{TRUE}$, and h_1 receives from every member j $\text{GO}_j = \text{TRUE}$ and $\text{HASH}\{\vec{B}_j\} = \text{HASH}\{\vec{B}_{h_1}\}$.

4. Case 1 does not occur, Case 2 does not occur, and at the start of Phase 6 *i*) $\text{GO}_{h_1} = \text{FALSE}$, *ii*) h_1 received $\text{GO}_j = \text{FALSE}$ from some member j, or *iii*) $\text{HASH}\{\vec{B}_j\} \neq \text{HASH}\{\vec{B}_{h_1}\}$ from some member j.

Cases 1, 2, and 4 are mutually exclusive events because the latter of these cases are explicitly defined to occur only when the previous do not. Case 3 is disjoint from the other three cases because each of them either results in termination before Phase 5 or results in $\text{GO}_h = \text{FALSE}$ sent in Phase 5 from an honest node h. One of these cases always occurs because one of the following is true of the execution:

1. The challenger fails, which as mentioned above only happens in one of these cases.

2. After Phase 4, $\textsc{go}_{h_1} = \textsc{false}$ or h_1 received from some member j $\textsc{go}_j = \textsc{false}$ or $\textsc{hash}\{\vec{B}_j\} \neq \textsc{hash}\{\vec{B}_{h_1}\}$, which implies that one of Cases 1, 2, or 4 above occurred.

3. After Phase 4, $\textsc{go}_{h_1} = \textsc{true}$ and h_1 received from every member j $\textsc{go}_j = \textsc{true}$ and $\textsc{hash}\{\vec{B}_j\} = \textsc{hash}\{\vec{B}_{h_1}\}$, which implies that Case 3 above occurred.

Now consider members of $T_{A,C}$ that fall in Case 1 above. C^0 sends messages according to the same distribution as C up to the Phase 3 message to h_1. Whether or not Case 1 also applies to a transcript in T_{A,C^0} is determined by the messages up to this point. Thus the probability that Case 1 applies to a transcript in T_{A,C^0} is the same as for a transcript in $T_{A,C}$. Moreover, Z^0 is independent of these messages, and thus C^0 fails under the first failure case of Game 0 with probability $1/2$ among those T_{A,C^0} transcripts in Case 1. Among those same transcripts where C^0 does not fail under the first failure case, Z^0 must be 1 and $\textsc{go}_{h_1} = \textsc{false}$ if Phase 4 is reached, so the other failure cases of Game 0 don't apply and C^0 behaves the same as C throughout the transcript. Therefore, the distribution of T_{A,C^0} transcripts in Case 1 given that $F^0 = 0$ is the same as the distribution of $T_{A,C}$ transcripts in Case 1.

Next consider those $T_{A,C}$ transcripts not in Case 1. Because Case 1 does not apply, C and C^0 behave the same up through Phase 3. Whether or not Case 2 applies is determined by the end of Phase 3. Thus the probability that Case 2 applies to a T_{A,C^0} transcript is the same as for a $T_{A,C}$ transcript. Z^0 is again independent of all partial T_{A,C^0} transcripts in Case 2 up through Phase 3. Thus, C^0 fails under the second Game 0 failure case in transcripts in this case with probability $1/2$. Moreover, when C^0 does not fail under the second failure case but the transcript is in Case 2, $Z^0 = 1$ and either $\textsc{go}_h = \textsc{false}$ for one of $h \in \{\alpha, \beta\}$ or the hashes of the broadcast vectors of α and β don't match. Thus, the other failure cases of Game 0 don't apply, and so C^0 behaves the same as C throughout the transcript. Therefore, the distribution of T_{A,C^0} transcripts in Case 2 given that $F = 0$ is the same as the distribution of $T_{A,C}$ transcripts in Case 2.

Next consider those $T_{A,C}$ transcripts not in either Case 1 or Case 2. Because Case 1 and Case 2 don't apply, C and C^0 behave the same up through Phase 4. Whether or not Case 3 applies is determined by the end of Phase 4. Thus the probability that Case 3 applies to a T_{A,C^0} transcript is the same as for a $T_{A,C}$ transcript. Z^0 is again independent of all partial T_{A,C^0} transcripts in Case 3 up through Phase 4. Thus, C^0 fails under the third failure case in transcripts in this case with probability $1/2$. Moreover, when C^0 does not fail under the third failure case but the transcript is in Case 3, $Z^0 = 0$, $\textsc{go}_{h_1} = \textsc{true}$, h_1 receives $\textsc{go}_j = \textsc{true}$ from all j, and $\textsc{hash}\{\vec{B}_j\} = \textsc{hash}\{\vec{B}_{h_1}\}$ for all j. Thus, the fourth failure case of Game 0 does not apply. The first two failure cases can't apply to any transcript in Case 3, and so C^0 behaves the same as C throughout the transcript. Therefore, the distribution of T_{A,C^0} transcripts in Case 3 given that $F^0 = 0$ is the same as the distribution of $T_{A,C}$ transcripts in Case 3.

Finally, consider those $T_{A,C}$ transcripts not in Case 1, Case 2, or Case 3. Because Cases 1–3 don't apply, C and C^0 behave the same up through Phase 5. Whether or not Case 4 applies is determined by the end of Phase 5. Thus the probability that Case 4 applies to a T_{A,C^0} transcript is the same as for a $T_{A,C}$ transcript. Z^0 is again independent of all partial T_{A,C^0} transcripts in Case 4 up through Phase 5. Thus, C^0 fails under the fourth failure case in transcripts in this case with probability $1/2$. Moreover, the other failure cases don't apply to transcripts in Case 4, and so, when Case 4 applies but C^0 does not fail, C^0 behaves the same as C throughout the transcript. Therefore, the distribution of T_{A,C^0} transcripts in Case 4 given that $F^0 = 0$ is the same as the distribution of $T_{A,C}$ transcripts in Case 4.

Thus, because $F^0 = 0$ with probability $1/2$ for each of the above cases, $F^0 = 0$ with probability $1/2$ overall. In addition, because the distribution of T_{A,C^0} transcripts in each of the above cases conditional on $F^0 = 0$ is the same as the distribution of $T_{A,C}$ transcripts in the same cases, and the probability of each case is the same between $T_{A,C}$ and C_{A,C^0}, the conditional distribution of T_{A,C^0} given $F^0 = 0$ is the same as the distribution of $T_{A,C}$. The game output $G^0(b)$ is $A^{C^0(b)}$ if $F^0 = 0$ and is a uniformly random bit if $F^0 = 1$. Therefore,

$$\begin{aligned} Pr[G^0(b) = 1] &= Pr[F^0 = 0]Pr[G^0(b) = 1|F^0 = 0] + Pr[F^0 = 1]Pr[G^0(b) = 1|F^0 = 1] \\ &= \frac{1}{2}Pr[A^{C(b)} = 1] + \frac{1}{2} \cdot \frac{1}{2}, \end{aligned}$$

which proves the lemma. □

The next lemma shows that changing the ciphertexts between Game 0 and Game 1 can only change the advantage of the game output by a negligible amount.

Lemma 10. $\left|\Delta(G^1) - \Delta(G^0)\right|$ *is negligible.*

Proof. We prove the lemma by constructing a distinguisher $D(b)$ that has a non-negligible advantage in the distinguishing game if $|Pr[G^1(b) = 1] - Pr[G^0(b) = 1]|$ is non-negligible. Let b_D be the challenge bit in the distinguishing game. D interacts with the distinguishing-game challenger $C_D(b_D)$ and A to execute either Game 0 or Game 1, depending on b_D, as follows:

1. D simulates the challenger of the anonymity game $C(b)$ exactly up to Phase 1 of the challenge shuffle. Let Z denote the random guess about later key releases that D makes as part of the simulation.

2. For Phase 1, D generates encryption key pairs $(I_{h_i}^{sec}, I_{h_i}^{pub})$ and $(O_{h_i}^{sec}, O_{h_i}^{pub})$, $1 < i \leq N - k$. D obtains the public encryption key K_1^{pub} from C_D and generates the encryption key pair (K_2^{sec}, K_2^{pub}). Then,

   Case 1: $Z = 0$. D sets
   $$\begin{aligned} &O_{h_1}^{pub} = K_1^{pub} \text{ and} \\ &(I_{h_1}^{sec}, I_{h_1}^{pub}) = (K_2^{sec}, K_2^{pub}). \end{aligned}$$

   Case 2: $Z = 1$. D sets
   $$\begin{aligned} &I_{h_1}^{pub} = K_1^{pub} \text{ and} \\ &(O_{h_1}^{sec}, O_{h_1}^{pub}) = (K_2^{sec}, K_2^{pub}). \end{aligned}$$

   Then D broadcasts these public keys from the honest members as described in the protocol.

3. For Phase 2a,

   Case 1: $Z = 0$. D sets $C'_\alpha = \{m_b\}_{I_N^{pub}:I_1^{pub}}$, $m'_0 = \{C'_\alpha\}_{O_N^{pub}:O_{h_1+1}^{pub}}$ and $m'_1 = \{\{\alpha\}_{I_N^{pub}:I_1^{pub}}\}_{O_N^{pub}:O_{h_1+1}^{pub}}$, using the encryption keys of α. D submits (m'_0, m'_1) to C_D, and receives c_{b_D} as a response. D sets $C''_\alpha = c_{b_D}$. D then finishes the phase as C^1 would starting after it creates C''_α in Case 1.

Case 2: $Z = 1$. D sets $m'_0 = \{m_b\}_{I_N^{pub}:I_{h_1+1}^{pub}}$ and $m'_1 = \{\alpha\}_{I_N^{pub}:I_{h_1+1}^{pub}}$. D submits (m'_0, m'_1) to C_D and receives c_{b_D} as a response. D sets $C'_\alpha = \{c_{b_D}\}_{I_{h_1-1}^{pub}:I_1^{pub}}$. All encryption is done using the keys received by α. D then finishes the phase as C^1 would starting after it creates C'_α in Case 2.

4. Phase 2b is executed as described in the protocol.

5. For Phase 3, D first receives a ciphertext vector of $\vec{C}_{h_1-1}$ intended for h_1. Then

   Case 1: $Z = 0$, and both C''_α and the partial decryption of C_β by $I_N^{sec}, \ldots, I_{h_1-1}^{sec}$ (which D can check because that ciphertext was constructed by D alone) appear in $\vec{C}_{h_1-1}$ exactly once. Then D replaces C''_α with m'_0, decrypts the remaining ciphertexts using C_D, shuffles the vector, and sets $\vec{C}_{h_1}$ to the result. D then finishes the phase as described in the protocol starting with sending $\vec{C}_{h_1}$ to member $h_1 + 1$.

   Case 2: $Z = 0$, and C''_α or the partial decryption of C_β does not appear in $\vec{C}_{h_1-1}$ exactly once. In this case, D terminates the simulation and sets $G \in \{0, 1\}$ uniformly at random.

   Case 3: $Z = 1$. D has the private outer keys for all honest members, and therefore can execute this phase just as C^1 would.

   D executes this phase for other honest members h_i, $i > 1$, as described in the protocol.

6. D executes Phase 4 as C^1 would, which is possible because this phase uses no private keys. If C^1 terminates, D terminates the simulation and sets $G \in \{0, 1\}$ uniformly at random.

7. D executes Phase 5 as C^1 would. This is possible because if $Z = 0$ D has the inner private keys and if $Z = 1$ C^1 fails if inner private keys are required. If C^1 fails, D terminates the simulation and sets $G \in \{0, 1\}$ uniformly at random.

8. D executes Phase 6 as C^1 would. This is possible because if $Z = 0$ C^1 fails if outer keys are required and if $Z = 1$ D has the outer private keys. If C^1 fails, D terminates the simulation and sets $G \in \{0, 1\}$ uniformly at random.

9. As many times as requested, D takes messages for the honest members and executes the shuffle protocol with A.

10. If the simulation did not terminate prematurely, let $\hat{b}_A$ be the guess output by A and set $G = \hat{b}_A$. D outputs its guess in the distinguishing game as $\hat{b}_D = G$.

We claim that D simulates C^{b_D} with A, that is, that D effectively executes Game 0 or Game 1, depending on b_D. That D correctly simulates all steps of the anonymity game except the challenge shuffle (i.e. Steps 1–4, 6, and 7) follows because it is defined as doing so, and these steps are the same for C^0 and C^1. To show that the challenge shuffle (Step 5) is simulated correctly, we show that for each phase, D simulates C^{b_D}:

- Phase 1: Although one public key is determined by the challenger C_D, the end result is that D broadcasts inner and outer public keys for honest members that are generated using the cryptosystem's key generation algorithm, just as both C^0 and C^1 do.

- Phase 2a:

  Case 1: $b_D = 0$. The result of using the response from C_D to construct C_α is that α commits to $\{\{m_b\}_{I_N^{pub}:I_1^{pub}}\}_{O_N^{pub}:O_1^{pub}}$, where the keys used are those received by α, just as in C^0. The other honest members behave as they do in C^1 by definition, which is the same as in C^0.

  Case 2: $b_D = 1$. The result of using the response from C_D is that α commits to $\{\{\alpha\}_{I_N^{pub}:I_1^{pub}}\}_{O_N^{pub}:O_1^{pub}}$, where the keys used are those received by α, just as in C^1. The other honest members behave as they do in C^1 by definition.

- Phase 2b: D, C^0, and C^1 all execute this phase as described in the protocol.

- Phase 3:

  Case 1: $Z = 0$ and both C''_α and the partial decryption of C_β by $I_N^{sec}, \ldots, I_{h_1-1}^{sec}$ appear in $\vec{C}_{h_1-1}$ exactly once. For h_1, D replaces C''_α with $m'_0 = \{\{m_b\}_{I_N^{pub}:I_1^{pub}}\}_{O_N^{pub}:O_{h_1+1}^{pub}}$ when constructing $\vec{C}_{h_1}$, and the other ciphertexts are simply decrypted . This is just as both C^0 and C^1 would have done. For the other honest members, D executes them as described in the protocol, just as C^0 and C^1 would do.

  Case 2: $Z = 0$ and either C''_α or the partial decryption of C_β by $I_N^{sec}, \ldots, I_{h_1-1}^{sec}$ does not appear in $\vec{C}_{h_1-1}$ exactly once. D terminates the game, just as both C^0 and C^1 would do.

  Case 3: $Z = 1$. D executes just as C^1 would by definition, which is the same as C^0.

- Phases 4–6: D executes these phases just as C^1 would by definition, which is the same as C^0.

Thus D correctly simulates C^{b_D} for A. Moreover, observe that when C^{b_D} does not prematurely terminate, then D uses the output $\hat{b}_A$ of A for the game output G, and when C^{b_D} does terminate, then D randomly sets $G \in \{0, 1\}$. This is exactly how G^{b_D} is set. Therefore, the advantage of D in the distinguishing game is

$$\begin{aligned}\left|Pr\left[\hat{b}_D = 1 | b_D = 1\right] - Pr\left[\hat{b}_D = 1 | b_D = 0\right]\right| &= |Pr[G = 1 | b_D = 1] - Pr[G = 1 | b_D = 0]| \\ &= \left|Pr\left[G^1 = 1\right] - Pr\left[G^0 = 1\right]\right|.\end{aligned}$$

Therefore, because we assume that the cryptosystem is IND-CCA2, $Pr[G^1 = 1] - Pr[G^0 = 1]$ must be negligible. This implies the lemma. ☐

Game 1 is modified to create Game 2 by replacing some ciphertexts of β just as Game 0 was modified to create Game 1 by replacing ciphertexts of α. Thus for similar reasons as before, it holds that that the advantage of the game output changes by a negligible amount from Game 1 to Game 2.

Lemma 11. *$\left|\Delta(G^2) - \Delta(G^1)\right|$ is negligible.*

Proof. The proof is exactly the same as the proof for Lemma 10 except for the following changes:

1. Everywhere α is replaced by β, β is replaced by α, and m_b^c is replaced by $m_{\bar{b}}^c$.

2. The simulation claim is that D simulates C^{b_D+1} for A, rather than C^{b_D}. Thus the resulting execution is identical to either Game 2 or Game 1, rather than Game 1 or Game 0, and the output G is has the same distribution as G^{b_D+1}, rather than G^{b_D}.

□

We now show that when Game 2 does not fail, the adversary has the same view whether m_0^c belongs to α or β and therefore has no advantage in the output of Game 2. In doing so we view the challenger C^2 as invoking a subroutine C'^2 that just executes the challenge shuffle of the anonymity game. This view allows our results to be reused when proving the anonymity of the bulk protocol, which calls the shuffle as a subprotocol.

Specifically, we consider the simulation by C^2 of ANONYMIZE-S during the challenge run of the shuffle protocol as an invocation of C'^2. The inputs from C^2 to C'^2 are the challenge bit b, the challenge members α and β, the challenge messages m_0^c and m_1^c, the honest non-challenge messages $\{m_h\}_{h\in H\setminus\{\alpha,\beta\}}$, the round number n_R, the signing keys K, the member ordering τ, and fail flags $\{f_h = \text{FALSE}\}_{h\in H}$. Let I be a vector all of these inputs except b. Let the output of honest members from the challenge shuffle be $O = (O_{h_1}, \ldots, O_{h_{N-k}})$, where O_{h_i} is the output of h_i. C^2 fails if and only if C'^2 fails. Let F'^2 indicate that C'^2 fails. Let M be the transcript of messages between C'^2 and A during the challenge shuffle. When $F'^2 = 1$, O and M are defined to take a constant failure value.

The following lemma shows that changing the challenge bit b does not change the joint probability of challenger failure, shuffle messages, and honest members' shuffle outputs:

Lemma 12.

$$Pr[M = m \wedge O = o \wedge F'^2 = f | I = i \wedge b = 0] = Pr[M = m \wedge O = o \wedge F'^2 = f | I = i \wedge b = 1].$$

Proof. We consider the messages sent in each phase as well as the final output and show that they do not depend on b. In order to do this, we also track some of the internal variables and show that they are updated the same way regardless of b.

- Phase 1:
  - C'^2 sets guess Z^2 independently of b.
  - Each honest member h generates inner and outer keypairs (I_h^{sec}, I_h^{pub}) and (O_h^{sec}, O_h^{pub}) independently of b.
  - The message $\mu_{h1} = \{I_h^{pub}, O_h^{pub}, n_R, 1, h\}\text{SIG}_{u_h}$ sent by each honest member $h \in H$ is independent of b by the above.
  - The messages to h from other honest members are shown above to be independent of b.
  - The messages to h from A are independent of b because A uses the outputs of SETUP-S as well as the messages from honest users to generate its messages, both of which are shown above to be independent of b.
  - GO_h is set to FALSE if h receives invalid public keys. Thus by the above GO_h is independent of b.
- Phase 2a: This phase depends on Z^2, which has been shown to be independent of b.
  - *Case 1*: $Z^2 = 0$.
    - The partially decrypted outer ciphertexts $C''_h = \{\{h\}_{I_N^{pub}:I_1^{pub}}\}_{O_N^{pub}:O_{h_1}^{pub}}, h \in \{\alpha, \beta\}$ do not depend on b by the above.

* The outer ciphertexts $C_h = \{C''_h\}_{O^{pub}_{h_1-1}:O^{pub}_1}$, $h \in \{\alpha, \beta\}$, do not depend on b by the above.
* The inner ciphertexts $C'_h = \{d_h\}_{I^{pub}_N:I^{pub}_1}$, $h \in H\backslash\{\alpha, \beta\}$, do not depend on b by the above.
* The outer ciphertexts $C_h = \{C'_h\}_{O^{pub}_N:O^{pub}_1}$, $h \in H\backslash\{\alpha, \beta\}$, do not depend on b by the above.
* Note that the inner ciphertext $C'_\alpha = \{m_b\}_{I^{pub}_N:I^{pub}_1}$ *does* depend on b (and similarly for C'_β).

- *Case 2*: $Z^2 = 1$.
  * The inner ciphertexts $C'_h = \{h\}_{I^{pub}_N:I^{pub}_1}$, $h \in \{\alpha, \beta\}$, do not depend on b by the above.
  * The inner ciphertexts $C'_h = \{d_h\}_{I^{pub}_N:I^{pub}_1}$, $h \in H\backslash\{\alpha, \beta\}$, do not depend on b by the above.
  * The outer ciphertexts $C_h = \{C'_h\}_{O^{pub}_N:O^{pub}_1}$, $h \in H$, do not depend on b by the above.
- The commitments $X_h = \text{COMMIT}\{C_h, h\}$, $h \in H$, do not depend on b because h does not and C_h does not by the above.
- The message $\mu_{h2a} = \{X_h, n_R, \texttt{2a}, h\}\text{SIG}_{u_h}$ sent by each $h \in H$ does not depend on b by the above.
- The additional inputs to A since his last output are messages μ_{h2a}, $h \in H$, shown above to be independent of b. Thus the messages μ_{i2a}, $i \in D$, received by $h \in H$ are independent of b.
- The messages μ_{h2a}, $h \in H$, received by $h' \in H$ are shown above to be independent of b.
- GO_h is set to FALSE if h receives an invalid commitment, $h \in H$. Thus GO_h still does not depend on b by the above.

• Phase 2b:

- The message $\mu_{h2b} = \{\text{OPEN}\{X_h\}, n_R, \texttt{2b}, h\}\text{SIG}_{u_h}$ sent by each $h \in H$ does not depend on b by the above.
- The additional inputs to A since his last output are messages μ_{h2b}, $h \in H$, shown above to be independent of b. Thus the messages μ_{i2b}, $i \in D$, received by $h \in H$ are independent of b.
- The messages μ_{h2b}, $h \in H$, received by $h' \in H$ are shown above to be independent of b.
- GO_h is set to FALSE if h receives an invalid opening or an opening to an invalid ciphertext, $h \in H$. Thus GO_h still does not depend on b by the above.

• Phase 3:

- Whether C'^2 fails depends on Z^2, on the partially decrypted ciphertexts C''_α and C''_β, and on the ciphertexts h_1 receives during the shuffle. Z^2, C''_α, and C''_β are shown above to be independent of b. If $h_1 = 1$, then the received ciphertexts are in the openings of the message committments from the previous phases. These are shown above not to depend on b. If $h_1 > 1$, then these ciphertexts are from the adversary in this phase, and since his last output, the adversary has only received as additional input messages from honest users that are independent of b, as shown above. Thus the outputs of A continue to be independent of b. In either case, therefore, the probability that C'^2 fails is independent of b.
- $h \in H$ chooses a permutation π_h to apply to the elements of the ciphertext vector it receives in this phase. π_h is chosen independently of b.
- The behavior of h_1 depends on Z^2, which has been shown to be independent of b.
  * *Case 1*: $Z^2 = 0$. A message is only sent by h_1 if C'^2 does not fail, which itself only happens when C''_α and C''_β appear exactly once each among the received ciphertexts. In this case, h_1 replaces these by $\{C'_\alpha\}_{O_N:O_{h_1+1}}$ and $\{C'_\beta\}_{O_N:O_{h_1+1}}$, where the keys used are those received by the α and β, respectively. If the encryption keys received by α and β do not match, then α and β will send different broadcast hashes to h_1 in Phase 4, and C'^2 will fail by Phase 6. Assuming C'^2 does not fail, the replacements C'^2 makes for C''_α and C'_β are m_b and $m_{\bar{b}}$, respectively, multiply encrypted in the same way. C'^2 simply decrypts the rest of the received ciphertexts using its outer private key.

    The received ciphertexts are received from A, which has not received any messages since the last phase. Therefore the above shows that these ciphertexts are independent of b. The permutation π_{h_1} used in the vector $\vec{C}_{h_1}$ is uniformly random. Thus, regardless of b, $\vec{C}_{h_1}$ contains in a random order m_0 and m_1 encrypted in the same way as well as the decryptions of the rest of the received ciphertexts. Therefore, assuming C'^2 has not and will not fail, the message $\mu_{h_1 3}$ sent by h_1 is independent of b.
  * *Case 2*: $Z^2 = 1$. The message $\mu_{h_1 3}$ sent by h_1 depends on π_{h_1},$O^{sec}_{h_1}$, messages from the previous phases, and messages from the adversary in this phase. These are all shown above to be independent of b.
- The message μ_{i3} from member $i > h_1$ depends on the messages from previous phases and messages in this phase from members $j < i$. We have shown above that messages from previous phases are independent of b. We inductively assume that messages in this phase from $j < i$ are independent of b. For $i \in D$, the only additional inputs A has received since the last phase are μ_{j3}, $j < i$, and therefore its outputs continue to be independent of b. For $i \in H$, μ_{i3} contains the permutation and decryption of the ciphertexts received by i in $\mu_{(i-1)3}$. The permutation π_i and decryption key O^{sec}_i used are shown above to be independent of b. Therefore μ_{i3} is independent of b.
- GO_h, $h \in H$, may be set to FALSE depending on the ciphertexts in μ_{h3}. This message is shown above to be independent of b, and so GO_h remains independent of b.
- The messages μ_{h3}, $h \in H$ received by $h' \in H$ are shown above to be independent of b.

- Phase 4:

- C'^2 fails if $Z^2 = 0$ and either α or β received a vector $\vec{C}_N$ that didn't contain both inner ciphertexts C'_α and C'_β at least once. If encryption keys received by α and β match, then the set $\{C'_\alpha, C'_\beta\}$ contains m_0 and m_1 encrypted in the same way, and thus it does not depend on b. $\vec{C}_N$ and Z^2 are shown above to be independent of b. Therefore, if the encryption keys of α and β match, whether or not C'^2 fails is independent of b. If those encryption keys don't match, then C'^2 will fail in Phase 6. The keys are received in an earlier phase, and so it follows from above that whether or not they match is independent of b.
- GO_h, $h \in H\backslash\{\alpha, \beta\}$, is updated depending on the inner ciphertext C'_h, μ_{N3}, the fail flag f_h, and GO_h itself, all of which are shown above to be independent of b. Thus GO_h remains independent of b.
- The update to GO_h, $h \in \{\alpha, \beta\}$, depends on Z^2, which is shown above to be independent of b, as follows:
  * *Case 1*: $Z^2 = 0$. In this case, if the ciphertext vector $\vec{C}_N$ sent to both α and β does not contain both the inner ciphertexts of α and β, then C'^2 will fail. Assuming that C'^2 does not fail, both GO_α and GO_β get set to FALSE if the fail flag is $f_h = \text{TRUE}$, and otherwise keep any existing FALSE value or get a new value of TRUE. They thus remain independent of b.
  * *Case 2*: $Z^2 = 1$. For $h \in \{\alpha, \beta\}$, GO_h is updated depending on f_h, the message μ_{N3} received by h, on C'_h, and on GO_h itself. In this case, the inner ciphertext C'_h is shown above to be independent of b. Likewise, f_h, μ_{N3} and GO_h are shown above to be independent of b.
- The message $\mu_{h4} = \{\text{GO}_h, \text{HASH}\{\vec{B}\}, n_R, 4, h\}\text{SIG}_{u_h}$, $h \in H$, does not depend on b by the above.
- The additional inputs to A since his last output are messages μ_{h4}, $h \in H$, shown above to be independent of b. Thus the messages μ_{i4}, $i \in D$, received by $h \in H$ are independent of b.
- The messages μ_{h4}, $h \in H$, received by $h' \in H$ are shown above to be independent of b.

- Phase 5:
  - Whether C'^2 fails in this phase depends on Z^2 and on the messages sent and received by h_1. These are shown above to be independent of b, and thus failure in this phase is independent of b also.
  - The message μ_{h5} sent by $h \in H$ and several internal variables are set differently depending on the messages μ_{i4} sent and received by h, which are shown above to be independent of b, as follows:
    * *Case 1*: h receives all $\text{GO}_i = \text{TRUE}$ and $\text{HASH}\{\vec{B}_i\} = \text{HASH}\{\vec{B}_h\}$.
      · The message $\mu_{h5} = \{I_h^{sec}, n_R, 5, h\}\text{SIG}_{u_h}$ does not depend on b by the above.
      · SUCCESS_h depends on the messages sent and received up to and including this phase. These messages are shown above to be independent of b, and thus SUCCESS_h is also.

- $\cdot$ M'_h depends on SUCCESS_h and on the messages sent and received up to and including this phase. All of these are shown above to be independent of b, and thus M'_h is also.

  - $*$ *Case 2*: h receives some $\text{GO}_i = \text{FALSE}$ or $\text{HASH}\{\vec{B}_i\} \neq \text{HASH}\{\vec{B}_h\}$.
    - $\cdot$ The message $\mu_{h5} = \{0, \vec{S}, n_R, 5, h\}\text{SIG}_{u_h}$ does not depend on b by the above.
    - $\cdot$ SUCCESS_h is set to FALSE, and thus is independent of b.

- The additional inputs to A since his last output are messages μ_{h5}, $h \in H$, shown above to be independent of b. Thus the messages μ_{i5}, $i \in D$, received by $h \in H$ are independent of b.
- The messages μ_{h5}, $h \in H$, received by $h' \in H$ are shown above to be independent of b.

- Phase 6:
  - Whether C'^2 fails in this phase assuming the encryption keys of α and β match (a case already covered above) depends on Z^2 and on the messages sent and received by h_1. These are shown above to be independent of b, and thus failure in this phase under the matching-keys assumption is independent of b also.
  - The message μ_{h6} sent by $h \in H$ is created differently depending on SUCCESS_h and the messages sent and received before this phase. These are shown above to be independent of b, and so the relevant case is independent of b as well.
    - $*$ *Case 1*: $\text{SUCCESS}_h = \text{TRUE}$. The message $\mu_{h6} = \{\vec{T}, n_R, 6, h\}\text{SIG}_{u_h}$ sent by h depends only on messages sent and received in previous phases. They are shown above to be independent of b, and thus μ_{h6} is as well.
    - $*$ *Case 2*: $\text{SUCCESS}_h = \text{FALSE}$, and for all i $\text{GO}_i = \text{TRUE}$ and $\text{HASH}\{\vec{B}_i\} = \text{HASH}\{\vec{B}_h\}$. The message $\mu_{h6} = \{\vec{T}, n_R, 6, h\}\text{SIG}_{u_h}$ sent by h depends only on messages sent and received in previous phases. They are shown above to be independent of b, and thus μ_{h6} is as well.
    - $*$ *Case 3*: $\text{SUCCESS}_h = \text{FALSE}$, and for some i $\text{GO}_i = \text{FALSE}$ or $\text{HASH}\{\vec{B}_i\} \neq \text{HASH}\{\vec{B}_h\}$. The message $\mu_{h6} = \{O_h^{sec}, \pi_h, \vec{T}, n_R, 6, h\}\text{SIG}_{u_h}$ sent by h depends on messages sent and received in previous phases as well as some internal variables, all of which are shown above to be independent of b.
  - The additional inputs to A since his last output are messages μ_{h6}, $h \in H$, shown above to be independent of b. Thus the messages μ_{i6}, $i \in D$, received by $h \in H$ are independent of b.
  - The messages μ_{h6}, $h \in H$, received by $h' \in H$ are shown above to be independent of b.
  - The outputs and some internal variables are set differently depending on SUCCESS_h and the messages sent and received before this phase. These are shown above to be independent of b, and so the relevant case is independent of b as well.
    - $*$ *Case 1*: $\text{SUCCESS}_h = \text{TRUE}$. The output $O_h = (\text{SUCCESS}, M'_h)$ is shown above to be independent of b.
    - $*$ *Case 2*: $\text{SUCCESS}_h = \text{FALSE}$, and for all i $\text{GO}_i = \text{TRUE}$ and $\text{HASH}\{\vec{B}_i\} = \text{HASH}\{\vec{B}_h\}$.

- $\cdot$ BLAME_h is set based only on messages sent and received up to this point and thus by the above is independent of b.
- $\cdot$ Log ℓ_h includes the output of SETUP-S and all messages sent and received by h and thus is independent of b by the above.
- $\cdot$ The output $O_h = (\text{FAILURE}, \text{BLAME}_h, \ell_h)$ is shown above to be independent of b.

$*$ *Case 3*: $\text{SUCCESS}_h = \text{FALSE}$, and for some i $\text{GO}_i = \text{FALSE}$ or $\text{HASH}\{\vec{B}_i\} \neq \text{HASH}\{\vec{B}_h\}$.

- $\cdot$ BLAME_h is set based only on messages sent and received up to this point and thus by the above is independent of b.
- $\cdot$ Log ℓ_h includes the output of SETUP-S and all messages sent and received by h and thus is independent of b by the above.
- $\cdot$ The output $O_h = (\text{FAILURE}, \text{BLAME}_h, \ell_h)$ is shown above to be independent of b.

Finally, we are able to prove that the messages, outputs, and failures of C'^2 are independent of b. The above analysis shows that the probability of failure does not depend on b. This implies that the probability that the messages M and outputs O of honest members take their constant failure values independently of b as well. When C'^2 does not fail, the above analysis shows that all messages and outputs from honest members are determined independently of b. Thus

$$Pr[M = m \wedge O = o \wedge F'^2 = f | I = i \wedge b = 0] = Pr[M = m \wedge O = o \wedge F'^2 = f | I = i \wedge b = 1].$$

□

We use this independence from b of the challenge shuffle's messages, outputs, and failure to prove that the adversary has no advantage in Game 2.

Lemma 13. $\Delta(G^2) = 0$.

Proof. To prove this, we show that the steps of the anonymity game surrounding the challenge shuffle are independent of b and use the previous lemma for the challenge shuffle itself.

1. In Step 1, the protocol executions are independent of b.

2. In Step 2, the all messages to the adversary have been independent of b, and so the users and messages A sends to C^2 for the challenge run are independent of b.

3. In Step 3, the challenger should assign the challenge messages to the correct challenge users, depending on b. However, we have modified the challenger to create Game 2 such that this is not necessary, and so we can omit this step.

4. During the challenge run in Step 4, C^2 first executes SETUP-S using as input the honest members' long-term signing keys, which are independent of b, as are the previous messages to A, and so the output (n_R, K, L) of SETUP-S is independent of b. C^2 then calls C'^2 with inputs $I = (\alpha, \beta, m_0^c, m_1^c, \{m_h\}_{h \in H \setminus \{\alpha,\beta\}}, n_R, K, \tau)$ and b. I is determined by previous messages from A and the outputs of SETUP-S. These have been shown to be independent of b, and thus I is as well. We can then apply Lemma 12 to conclude that the joint distribution of shuffle failure and messages to A are independent of b.

5. If C'^2 didn't fail, which as shown occurs independently of b, then C^2 executes Step 5 of the anonymity game by executing additional protocol executions. These depend on messages from A, and all messages to A have been shown independent of b. Thus these executions are independent of b.

6. The adversary's guess $\hat{b}$ in Step 6 depends on the messages to A and the possible failure of C^2. These have been shown to be independent of b, and so $\hat{b}$ is independent of b.

$G^2(b)$ depends on F^2 and $\hat{b}$. These have been shown to be independent of b, and thus

$$Pr[G^2(1) = 1] = Pr[G^2(0) = 1].$$

□

Theorem 5. *The* GMP-SHUFFLE *protocol maintains anonymity with k colluding members for any $0 \le k \le N - 2$.*

Proof. Let A be a probabilistic polynomial-time adversary. Let the change in advantage between Games i and j be $\epsilon_{ij} = \left|\Delta(G^j) - \Delta(G^i)\right|$. By Lemma 9, the advantage of A in the anonymity game with GMP-SHUFFLE is $2\Delta(G^0) \le 2(\epsilon_{01} + \epsilon_{12} + \Delta(G^2))$. ϵ_{01} is negligible by Lemma 10, ϵ_{12} is negligible by Lemma 11, and $\Delta(G^2)$=0 by Lemma 13. Thus the advantage of A in the anonymity game with GMP-SHUFFLE is negligible. □

5.5.2 The GMP-Bulk Protocol

We show that the adversary's advantage in winning the anonymity game with GMP-BULK is negligible.

As in the shuffle anonymity proof (Section 5.5.1), we take an adversary A playing against the anonymity-game challenger C and construct a sequence of games by successively modifying the challenger. We will show how any non-negligible difference in the game's advantage between neighboring games will contradict assumed security properties of the cryptographic primitives. The final game will be information-theoretically secure, that is, the output advantage will be zero. We incorporate the anonymity proof for the shuffle by using that sequence of games (extended to GMP-BULK) as game subsequences modifying the challenger during the bulk protocol's shuffle phases.

We define *Game 0*, *Game 1*, and *Game 2* by changing the behavior of C during the message-descriptor shuffle in Phase 3. The changes are essentially the same as those made in Game 0, Game 1, and Game 2, respectively, in the shuffle anonymity proof (Section 5.5.1). We then similarly define *Game 3*, *Game 4*, and *Game 5* by applying the same sequence of changes to the blame shuffle in Phase 7. We replace the encrypted seeds sent in the message descriptors of α and β with unrelated ciphertexts to define *Game 6*. Finally, in *Game 7*, we replace the pseudorandom bit streams sent during data transmission with random streams.

As before, let $h_1, h_2, \ldots, h_{N-k}$ be the honest users in the order they appear in the shuffle. Let C^i be the challenger defined in Game i. Let Z_1^i and Z_2^i indicate that challenger C^i guesses that h_1 should release her outer private key at some point as part of the message descriptor shuffle (Phase 3) and the blame shuffle (Phase 7), respectively. Let F^i indicate whether or not the challenger failed in Game i. Let G^i indicate a "game output" for Game i. The challenge bit b is again an implicit input to the games' challengers and associated random variables.

Game 0: We create a challenger C^0 that sets $Z_1^0 \in \{0, 1\}$ uniformly at random as a guess about if h_1 should reveal an outer private key during the message-descriptor shuffle of the challenge run in the bulk anonymity game. That is, C^0 behaves the same as the anonymity-game challenger except that he fails if his guess proves to be wrong at certain points during Phase 3 of the challenge protocol run. These failure points are exactly the same (using Z_1^0 in place of Z^0) as those defining C^0 for Game 0 of the shuffle anonymity analysis (Section 5.5.1), and so we do not repeat them here. Again, when failure occurs, $F^0 = 1$, C^0 terminates, and the game output G^0 is set to a uniformly random bit. In every other case, $F^0 = 0$, C^0 behaves exactly as C would.

Game 1: We again reuse the changes described in the shuffle anonymity analysis. We create challenger C^1 by applying the changes that define C^1 for Game 1 of the shuffle analysis to the challenger C^0 defined above. These changes are made to the Phase 3 shuffle of the challenge run in the bulk anonymity game. Everywhere Z^1 appears in these changes, we instead use Z_1^1, and everywhere m_b^c appears, we instead use the shuffle input of α (which is a message descriptor). These changes effectively replace a ciphertext containing the message descriptor of α with one that contains a dummy message until it has been shuffled by the first honest member.

Game 2: As in the shuffle anonymity analysis, this game is created from Game 1 above in the same way that Game 1 itself was created from Game 0, except replacing Z_1^1 with Z_1^2, α with β, and the shuffle input of α with the shuffle input of β. This effectively replaces a ciphertext containing the message descriptor of β with one that contains a dummy message until it has been shuffled by the first honest member.

Games 3–5: These games further modify the challenger by adapting and applying the sequence of changes given in the shuffle anonymity analysis as was done to define Games 0–2 above. This time, however, we apply the changes to the blame shuffle (Phase 7) of the challenge protocol run. In addition, the guess bit is denoted Z_2^i, and the shuffle inputs to α and β are accusations rather than message descriptors.

Game 6: We define challenger C^6 from C^5 by changing the inputs to the message-descriptor shuffle of the challenge run. During the generation of message descriptors (Phase 2), we replace the encrypted seeds $S_{\alpha\beta}$ and $S_{\beta\alpha}$ with the encryption of new random seeds. Specifically,

1. For α, we replace the encrypted seed it creates for β in Case 1 of Phase 2 with an encryption of the new random seed $s'_{\alpha\beta}$. That is, we set $S_{\alpha\beta} = \{s'_{\alpha\beta}\}_{y_\beta}$, where the encryption key is among those α received in Phase 1a. Note that the original seed $s_{\alpha\beta}$ is still created and used to generate the ciphertext $C_{\alpha\beta}$.

2. For β, we replace the encrypted seed it creates for α in Case 1 of Phase 2 with an encryption of the new random seed $s'_{\beta\alpha}$. That is, we set $S_{\beta\alpha} = \{s'_{\beta\alpha}\}_{y_\alpha}$, where the encryption key is among those β received in Phase 1a. Again, note that the original seed $s_{\beta\alpha}$ is still created and otherwise used as before.

Then during data transmission (Phase 4), C^6 recognizes the seeds that match $s'_{\alpha\beta}$ and $s'_{\beta\alpha}$ among those received by β and α, respectively, and simply uses the original seeds to generate the necessary ciphertexts. More precisely, for α, in Case 2 of Phase 4, whenever a value $S_{i\alpha}$ received by α decrypts to a seed that the challenger recognizes is identical to $s'_{\beta\alpha}$, α sets $C_{i\alpha}$ to the ciphertext $C_{\beta\alpha}$ that was generated earlier from $s_{\beta\alpha}$. A similar action is taken for β, where this time the challenger looks for decrypted seeds matching $s'_{\alpha\beta}$ and uses $C_{\alpha\beta}$ for the ciphertext.

Game 7: We construct challenger C^7 from C^6 by replacing some pseudorandomness with true randomness during the challenge protocol run. For α and β, in Case 1 of Phase 2 (descriptor generation), the ciphertexts $C_{\alpha\beta}$ and $C_{\beta\alpha}$, respectively, are chosen uniformly at random rather than being generated pseudorandomly. Note that these random ciphertexts are then used in the computation of $C_{\alpha\alpha}$ and $C_{\beta\beta}$, respectively. Then in Case 2 of Phase 4 (data transmission), α and β use these random sequences as ciphertexts. That is, α sends the random $C_{\beta\alpha}$ generated in Phase 2 for every decrypted seed $s_{i\alpha}$ that matches $s'_{\beta\alpha}$. Similarly, β sends the random $C_{\alpha\beta}$ generated in Phase 2 for every decrypted seed $s_{i\beta}$ that matches $s'_{\alpha\beta}$.

The following lemma shows that, as in the shuffle proof, the output's advantage in Game 0 is $1/2$ the advantage of A in the anonymity game:

Lemma 14.

$$\Delta(G^0) = \frac{1}{2}\left|Pr\left[A^{C(0)} = 1\right] - Pr\left[A^{C(1)} = 1\right]\right|,$$

where the probability is taken over the randomness of both the adversary and the challenger.

Proof. The proof of this lemma is almost exactly the same as the proof of Lemma 9. We simply interpret each reference to a phase of the challenge shuffle as instead referring to a phase of the message-descriptor shuffle in the bulk protocol. We also replace Z^0 everywhere it appears with Z^0_1. □

The next lemma shows that, as in the shuffle proof, the ciphertext changes from Game 0 to Game 2 can only change the advantage of the game output by a negligible amount.

Lemma 15. $\left|\Delta(G^2) - \Delta(G^0)\right|$ *is negligible.*

Proof. Games 1 and 2 are constructed by making essentially the same changes to the challenger's behavior during the descriptor shuffle that were made in Games 1 and 2 of the shuffle anonymity analysis. Thus, the proof that these two sets of changes each only change the output advantage by a negligible amount is almost exactly the same as the proofs of Lemmas 10 and 11.

In these proofs, a distinguisher D is constructed that simulates either member of a pair of adjacent games for the adversary, depending on the hidden bit of the distinguishing game. The proofs show that this distinguisher converts a non-negligible change in the game output's advantage to a non-negligible advantage in the distinguishing game. Such an advantage would contradict the IND-CCA2 property of the cryptosystem.

We slightly modify the argument of that sort in the proof of Lemma 10 to prove that the output advantage changes negligibly between Game 0 and Game 1. We construct a distinguisher D^B that is the same as D in that proof with the following differences:

1. In Step 1 of D, D^B instead executes the anonymity game up to the challenge run of the bulk protocol (rather than the shuffle protocol).

2. D^B then executes Phase 1 and Phase 2 of bulk protocol exactly, ending up with the inputs of honest members to shuffle protocol m_{h_i} that are constructed during Phase 2.

3. D^B continues with Step 2 of D.

4. D^B continues with Steps 3–8 of D, replacing m_b with m_α everywhere.

5. After Step 8 of the distinguisher is finished, the message-descriptor shuffle (i.e. Phase 3) of the bulk protocol is over, and the D^B uses the outputs of the honest members to execute the rest of the bulk protocol (Phase 4 – Phase 7) as described in the protocol.

By applying the subsequent arguments of Lemma 10 to D^B (again substituting m_α for m_b in the arguments), we can show that the game output's advantage changes negligibly between Game 0 and Game 1.

We can adapt the distinguisher construction and subsequent arguments of Lemma 11 in the same way (except using β in place of α and $\bar{b}$ in place of b) to show that the game output's advantage changes negligibly between Game 1 and Game 2. □

Game 3 is created by applying the first game transformation of the shuffle proof to the blame shuffle in Game 2, that is, by having the challenger guess about the revelation of outer private keys. Thus, as in the shuffle proof, the game advantage decreases by a factor of $1/2$:

Lemma 16. $\Delta(G^3) = \frac{1}{2}\Delta(G^2)$*, where the probability is taken over the randomness of both the adversary and the challenger.*

Proof. As with Lemma 14, the proof of this lemma is almost exactly the same as the proof of Lemma 9. We apply that proof to this lemma by interpreting each reference to a phase of the challenge shuffle as instead referring to a phase of the blame shuffle in the challenge bulk round. As we are comparing Games 2 and 3 rather than the anonymity game and Game 0, everywhere they appear we replace C with C^2, $A^{C(b)}$ with $G^2(b)$, C^0 with C^3, and Z^0 with Z_2^3. In addition, the transcripts between A and C^2 (i.e. T_{A,C^2}) and between A and C^3 (i.e. T_{A,C^3}) may fall into one more case than the four given in that proof. The challenger may fail with an incorrect guess Z_1 during the descriptor shuffle. The proof of Lemma 14 shows that this failure occurs with probability $1/2$ in Game 0, and in Game 2 this failure continues to occur with probability $1/2$ and for the same reasons, namely that each transcript falls into exactly one of the four listed cases, and failure occurs in each case when the independently random bit Z_1^2 has a certain value. The proof of Lemma 9 can easily be modified to include this failure case, with the following modified conclusions:

1. Each transcript case occurs with the same probability for T_{A,C^2} and T_{A,C^3}.
2. Failure occurs in every case except for the added one (which always fails) with probability $1/2$.
3. The distribution of transcripts in T_{A,C^3} conditional on $F^3 = 0$ is, in every case except the added one, the same as the distribution of transcripts in the same case in T_{A,C^2}.

These imply that $F^3 = 0$ with probability $1/4$ and that the conditional distribution of T_{A,C^3} given $F^3 = 0$ is the same as the distribution of T_{A,C^2} given that $F^2 = 0$. The game outputs G^3 and G^2 are the adversary output when the challenger does not fail and are uniformly random bits otherwise. Thus

$$
\begin{aligned}
\left|Pr\left[G^3(0)=1\right] - Pr\left[G^3(1)=1\right]\right| &= \frac{1}{4}\left|Pr[G^3(0)=1|F^3=0] - Pr[G^3(1)=1|F^3=0]\right| \\
&= \frac{1}{4}\left|Pr[G^2(0)=1|F^2=0] - Pr[G^2(1)=1|F^2=0]\right| \\
&= \frac{1}{2}\left|Pr[G^2(0)=1] - Pr[G^2(1)=1]\right|
\end{aligned}
$$

□

Changing ciphertexts from the challenger from Game 3 to Game 5 has only a negligible effect on the output advantage, as in the analogous game transitions of the shuffle proof:

Lemma 17. $\left|\Delta(G^5) - \Delta(G^3)\right|$ *is negligible.*

Proof. This lemma can be proven using the arguments of Lemma 15 applied to the blame shuffle rather than the descriptor shuffle. Those construct distinguishers and show that they convert a non-negligible change in the game output between Games 3 and 4 or between Games 4 and 5 into a non-negligible advantage in the IND-CCA2 game. This would contradict the IND-CCA2 property of the cryptosystem. □

Game 6 is created from Game 5 by changing some PRNG seeds that are then encrypted and sent by the challenger. By the IND-CCA2 property of the encryption scheme, this can only have a negligible effect on the output advantage:

Lemma 18. $\left|\Delta(G^6) - \Delta(G^5)\right|$ *is negligible.*

Proof. To prove this lemma, we consider the two ciphertext changes in sequence: *i*) $\{s_{\alpha\beta}\}_{y_\beta}$ gets replaced by $\{s'_{\alpha\beta}\}_{y_\beta}$ and *ii*) $\{s_{\beta\alpha}\}_{y_\alpha}$ gets replaced by $\{s'_{\beta\alpha}\}_{y_\alpha}$. For each change, we can construct a distinguisher that converts a non-negligible change in the game-output distribution into a non-negligible advantage in the distinguishing game.

Let Game 5a refer to the game that results from just the ciphertext replacement in (*i*). Let C_D be the challenger in the distinguishing game and b_D be the challenge bit. We construct a distinguisher D that simulates either Game 5 or Game 5a, depending on b_D, as follows:

1. D simulates the anonymity-game challenger C^5 up to the challenge run of the bulk protocol.

2. To begin Phase 1 of the bulk protocol (key generation), D obtains the public encryption key K_D from C_D and sets $y_\beta = K_D$. D generates the encryption key pairs (x_h, y_h) for all other honest users. Then D continues with the rest of Phase 1a (session-key generation) followed by Phase 1b (key verification), acting as C^5 would.

3. D executes Phase 2 (descriptor generation) for α as follows:

   Case 1: If key verification is successful (Case 1 of Phase 2), D executes the phase for α as C^5 would up to the point at which $S_{\alpha\beta}$ is assigned. At this point, D randomly chooses a new seed $s'_{\alpha\beta}$, submits $(s_{\alpha\beta}, s'_{\alpha\beta})$ to C_D, receives c_{b_D} as a response, and sets $S_{\alpha\beta} = c_{b_D}$. D executes the rest of the phase for α as C^5 would.

   Case 2: If key verification fails (Case 2 of Phase 2), D executes the phase for α as C^5 would.

   Case 3: This case will never execute for α because α has message m_b to send.

   D executes Phase 2 for the other honest members as C^5 would.

4. D executes Phase 3 as C^5 would.

5. D executes Phase 4 (data transmission) for β as follows:

   Case 1: If the descriptor shuffle failed (Case 1 of Phase 4), D executes the phase for β as C^5 would.

   Case 2: Otherwise the descriptors were successfully received (Case 2 of Phase 2). For each encrypted seed $S_{i\beta}$ received by β in a descriptor, if $S_{i\beta}$ matches the encrypted seed $S_{\alpha\beta} = c_{b_D}$ created by α for β, then D sets $s_{i\beta}$ to the seed $s_{\alpha\beta}$ chosen by α in Phase 2, rather than obtaining it by decrypting $S_{i\beta}$. Otherwise, D sends $S_{i\beta}$ to C_D for decryption, receiving s in response. If $s = s'_{\alpha\beta}$, then set $s_{i\beta} = s_{\alpha\beta}$, and otherwise set $s_{i\beta} = s$. D completes the phase as C^5 would.

   D executes this phase for the other honest members as C^5 would.

6. D executes Phase 5 (acknowledgement submission) and Phase 6 (message recovery) as C^5 would.

7. D executes Phase 7 (blame) as C^5 would. It will never be required for D to produce the random bits used to produce $S_{\alpha\beta}$, which it would be unable to do, because β only sends ciphertexts with correct hashes for slots with the descriptors of honest members.

8. D executes the rest of the anonymity game after the challenge run as C^5 would.

9. D uses G as its guess $\hat{b}_D$.

We observe that, except with negligible probability, D simulates C^5 if $b_D = 0$ (i.e. if $c_{b_D} = \{s_{\alpha\beta}\}_{y_\beta}$), and D simulates C^{5a} if $b_D = 1$. Note that, depending on b_D, D creates a message descriptor for α that contains as a seed for β either the encryption of $s_{\alpha\beta}$ or of $s'_{\alpha\beta}$. Moreover, if $b_D = 1$, D correctly uses $s_{\alpha\beta}$ for all encrypted seeds received by β that match $s'_{\alpha\beta}$, and, if $b_D = 0$, the probability that β receives an encryption of $s'_{\alpha\beta}$ and (incorrectly) uses $s_{\alpha\beta}$ as the decryption is negligible because $s'_{\alpha\beta}$ is never used in the simulation up to that point and is chosen independently at random. In addition, the ciphertexts sent to the decryption oracle never match the forbidden text $c_{b_D} = S_{\alpha\beta}$ because in those cases the decryptions are copied directly from the seed created by α for β. In every other way, C^5 and C^{5a} act the same, and D simulates their behavior.

The output of D is the game output $G(b)$, where b is the challenge bit of the simulated anonymity game. $G(b)$ is set exactly as it is by the simulated challenger except with negligible probability, and thus the advantage of D is negligibly close to the change in $G(b)$ for any b. That is,

$$\left|Pr[\hat{b}_D = 1|b_D = 0] - Pr[\hat{b}_D = 1|b_D = 1]\right| - \left|Pr[G^5(b) = 1] - Pr[G^{5a}(b) = 1]\right|$$

is negligible. Because the advantage in the distinguishing game is negligible by the IND-CCA2 property of the cryptosystem, the change in the output distribution between Game 5 and Game 5a for a given value of b must be negligible. This implies that the change in the output advantage is also negligible.

Applying ciphertext replacement (*ii*) to Game 5a results in Game 6. Essentially the same argument as above (simply swapping α and β everywhere) shows that the output advantage changes negligibly as a result of this replacement.

Thus the output advantage changes negligibly between Game 5 and Game 6. $\square$

We create Game 7 from Game 6 by replacing some pseudorandom streams with random streams. By the pseudorandomness of the PRNG, doing so has a negligible effect on the output advantage:

Lemma 19. *$\left|\Delta(G^7) - \Delta(G^6)\right|$ is negligible.*

Proof. Consider the changes made to C^6 in the following sequence: *i*) β chooses the ciphertext $C_{\beta\alpha}$ in Phase 2 randomly instead of pseudorandomly, and α uses that ciphertext in Phase 4; and *ii*) α chooses the ciphertext $C_{\alpha\beta}$ in Phase 2 randomly instead of pseudorandomly, and β uses that ciphertext in Phase 4. Let Game 6a be the game defined by applying (*i*) to Game 6. Game 7 is then (*ii*) applied to Game 6a. We can show that the game output changes negligibly for each pair in this short sequence by constructing a distinguisher that converts a change in the game output probability to the same advantage in the pseudorandomness game.

Let C_R be the challenger in the pseudorandomness game, and let b_R be its challenge bit. Distinguisher D interacts with C_R to simulate either Game 6 or Game 6a for the adversary, depending on b_R. Let D behave as follows:

1. D executes the anonymity game as C^6 would up to the challenge run of the bulk protocol.
2. D executes Phase 1 as C^6 would.
3. In Phase 2, D receives r from C_R. For member β, D sets $C_{\beta\alpha} = r$ in Case 1 and otherwise executes the phase for β as C^6 would. D executes Phase 2 for all other honest members as C^6 would.
4. D executes Phase 3 as C^6 would.
5. In Phase 4, for member α, D sets $C_{j\alpha} = r$ for all decrypted seeds $s_{i\alpha}$ that are identical to the seed $s'_{\beta\alpha}$ generated by β. D otherwise executes Phase 4 for β as C^6 would. D executes Phase 4 for all other honest members as C^6 would.
6. D executes Phase 5, Phase 6, and Phase 7 as C^6 would.
7. D executes the rest of the anonymity game after the challenge run.
8. D uses the game output of the simulated challenger as guess $\hat{b}_R$.

We observe that if $b_R = 0$ (i.e. r is pseudorandomly generated by C_R from an unknown random seed s), then D simulates Game 6, and if $b_R = 1$, then D simulates Game 6a. In particular, D can execute the blame phase without knowing the seed that is used to generate r, if any, because the encrypted seed included in the descriptor d_β is already an unrelated seed $s'_{\beta\alpha}$. Also, the challenger creates the encrypted seeds in both games, and so any accusation can be correctly generated, although because β only generates accusations for slots with its own descriptor d_β, and α always produces correct ciphertexts when using d_β, it should in fact never be the case that β generates an accusation involving the ciphertexts changed between Game 6 and Game 6a.

The guess bit $\hat{b}_R$ of D is thus G^6 when $b_R = 0$ and G^{6a} when $b_R = 1$. Therefore if the difference between $Pr[G^6(b) = 1]$ and $Pr[G^{6a}(b) = 1]$ were non-negligible for some b, then D could achieve a non-negligible advantage in the pseudorandomness game. This would contradict pseudorandomness, and thus the change in the output advantage from Game 6 to Game 6a is negligible.

A nearly identical argument, simply swapping α and β everywhere, shows that there is a negligible change in the game advantage from Game 6a to Game 7 as well. Thus, the change in the game advantage from Game 6 to Game 7 is negligible. □

By Game 7, the adversary has the same view whether m_0 belongs to α or β, and thus there is no advantage in the game output. In order to show this, we follow the approach of Lemmas 12 and 13, and we view the challenger C^7 as calling a subroutine C'^7 to execute ANONYMIZE-B during the challenge run. This allows a natural decomposition of the proof, and it also us to express the fact that in addition to the messages to the adversary, the outputs of the bulk protocol are independent of b. Thus if, for example, the members decide later to come to a consensus about the results of the bulk protocol, that information won't break anonymity. C'^7 takes as input the challenge bit b and $I = (n_R, n_{R_1}, n_{R_2}, K, \tau, \alpha, \beta, m_0^c, m_1^c, \{m_h\}_{h \in H \setminus \{\alpha,\beta\}})$. C'^7 either fails or returns output $O = (O_{h_1}, \ldots, O_{N-k})$, where O_h is the output of ANONYMIZE-B for member h. C^7 fails if and only if C'^7 fails.

In addition, we view C'^7 as executing the descriptor and blame shuffles by calling as a subroutine the challenger C'^2 as defined in Section 5.5.1 for use in Lemma 12. C'^7 uses as inputs to C'^2 the same K, α, β, and τ that itself received. It uses n_{R_1} as the round nonce input for the descriptor shuffle (i.e. Phase 3) and n_{R_2} as the round nonce input for the blame shuffle (i.e. Phase 7). The member messages and fail flags are determined from its own inputs as described in the bulk protocol description. For the descriptor shuffle, we denote by $m_0^{c_1}$ and $m_1^{c_1}$ the challenge messages, by m_h^1 the non-challenge messages, and by f_h^1 the fail flags. For the blame shuffle, we denote by $m_0^{c_2}$ and $m_1^{c_2}$ the challenge messages, by m_h^2 the non-challenge messages, and by f_h^2 the fail flags. We denote by $O^1 = (O^1_{h_1}, \ldots, O^1_{h_{N-k}})$ the output of the descriptor shuffle and by $O^2 = (O^2_{h_1}, \ldots, O^2_{h_{N-k}})$ the output of the blame shuffle. C'^7 fails if and only if one of the two invocations of C'^2 fails.

Let M be the transcript of all messages between members during the protocol. Let F'^7 be the event that C'^7 fails. When $F'^7 = 1$, O and M are defined to take a constant failure value. The following lemma shows that changing b does not change the joint distribution of M, O, and F'^7.

Lemma 20.

$$Pr[M = m \wedge O = o \wedge F'^7 = f | I = i \wedge b = 0] = Pr[M = m \wedge O = o \wedge F'^7 = f | I = i \wedge b = 1].$$

Proof. To prove this, we track the dependence on b of messages from C'^7 to A, internal variables of C'^7, and outputs of C'^7. This analysis will show that the messages and outputs of C'^7 follow the same distribution regardless of b. In order to do this despite the dependence of some variables on b, we will consider two parallel executions of the challenge round, one in which $b = 0$ and one in which $b = 1$. The messages, variables, and outputs that do not depend on b will indeed be the same in the two executions. The variables that do depend on b may have different states between the two executions, but the probability of those paired states will be the same.

We consider these executions step-by-step as follows:

- Phase 1a:
  - Encryption keys (x_h, y_h) are generated independently of b.
  - The message μ_{h1a} from $h \in H$ is independent of b by the above.
  - The additional inputs to A since his last output are messages μ_{h1a}, $h \in H$, shown above to be independent of b. Thus the messages μ_{i1a}, $i \in D$, received by $h \in H$ are independent of b.
  - The messages μ_{h1a}, $h \in H$, received by $h' \in H$ are shown above to be independent of b.

- Phase 1b:
  - The message $\mu_{h1b} = \{\vec{K}_h^e, n_R, \texttt{1b}, h\}\textsc{sig}_{u_h}$ from $h \in H$ contain keys received from other members and thus is independent of b by the above.
  - The additional inputs to A since his last output are messages μ_{h1b}, $h \in H$, shown above to be independent of b. Thus the messages μ_{i1b}, $i \in D$, received by $h \in H$ are independent of b.
  - The messages μ_{h1b}, $h \in H$, received by $h' \in H$ are shown above to be independent of b.

- Phase 2: We consider several cases for how challenge members form descriptors. These cases depend on the keys that honest members received in the previous phases, and thus by the above the applicable case does not depend on b.
  - *Case 1*: All honest members received valid and matching keys in the previous phases. In this case, the descriptors d_α and d_β do depend on b, and so we compare their generation when $b = 0$ and when $b = 1$. We observe that the descriptor for the challenge member $h \in \{\alpha, \beta\}$ assigned m_0 is created the same regardless of whether h is α or β. A seed is chosen uniformly at random for each member i, it is encrypted to produce S_{hi} using the same set of keys (as assumed for this case), and the randomness of the encryption is saved as R_{hi}. Ciphertexts C_{hi} are produced for all $i \in G \backslash \{\alpha, \beta\}$ using the PRNG with the seed generated for i. The ciphertext $C_{h\alpha}$ is chosen randomly, and $C_{h\beta}$ is chosen such that the XOR of all ciphertexts yields m_0. The descriptor is then created from the encrypted seeds, hashes of the ciphertexts, and the length of m_0. To emphasize that the creation of the descriptor depends on the message rather than its owner, we use the additional notation of $s_{m_0 i}$ for the seeds, $R_{m_0 i}$ for the encryption randomness, $C_{m_0 i}$ for the ciphertexts, and d_{m_0} for the descriptor. The descriptor for the challenge member assigned m_1 is similarly generated, and we use similar user-independent notation for it and its components. Thus, for specific d_{m_0} and d_{m_1}, the probability that $d_\alpha = d_{m_0}$ and $d_\beta = d_{m_1}$ when $b = 0$ is the same as the probability that $d_\alpha = d_{m_1}$ and $d_\beta = d_{m_0}$ when $b = 1$. We thus let the former occur in the execution under consideration for $b = 0$ and the latter occur in the execution for $b = 1$.
  - *Case 2*: Some honest member h received an invalid key or non-matching keys in the previous rounds. In this case, C'^7 will use $f_h^1 = \texttt{TRUE}$ as an input to C'^2 and thus cause the shuffle to fail. If $Z_1^7 = 0$, the challenger has guessed wrong, and the challenger will fail. Assuming the challenger does not fail, $Z_1^7 = 1$, and so the descriptors of α and β are never needed (the dummy message is preserved throughout the shuffle). Thus we assume that C^7 does not create them at all.
  - Member $h \in H \backslash \{\alpha, \beta\}$ creates her descriptor d_h in a way that only depends on her input message and the keys she received in the previous rounds. It is shown above that neither of these depends on b, and so her descriptor does not depend on b.

- Phase 3:
  - Each $h \in H$ sets the fail flag f_h^1 for the shuffle in this phase based on keys received in previous rounds, and thus it is independent of b by the above.

- C'^7 calls C'^2. The inputs to C'^2 are challenge users α and β, challenge messages $m_0^{c_1} = d_{m_0}$ and $m_1^{c_1} = d_{m_1}$, non-challenge messages $m_h^1 = d_h$ for $h \in H \backslash \{\alpha, \beta\}$, round number n_{R_1}, signing keys K, member ordering τ, fail flags f_h^1, and challenge bit b. As shown above, all of the inputs to C'^2 except b are independent of b. Thus with I set to all those inputs except b we can apply Lemma 12. We conclude that C'^7 fails in this step with probability independent of b, that the messages sent are independent of b, and that the output O^1 is independent of b.
- The message $\mu_{h3} = \{p', n_R, 3, h\}\textsc{sig}_{u_h}$ from $h \in H$ contains evidence of invalid or non-matching keys if any are received. Thus it depends only on messages received in previous rounds and is independent of b by the above.
- The additional inputs to A since his last output are messages μ_{h3}, $h \in H$, shown above to be independent of b. Thus the messages μ_{i3}, $i \in D$, received by $h \in H$ are independent of b.
- The messages μ_{h3}, $h \in H$, received by $h' \in H$ are shown above to be independent of b.

• Phase 4:

- For each member $h \in H$, we consider two cases for the message she sends. Which case applies depends on the shuffle output O_h^1. O_h^1 is shown above to be independent of b, and so the relevant case is independent of b.
  * *Case 1*: $O_h^1 = (\textsc{failure}, \textsc{blame}_h^{s_1}, \ell_h^{s_1})$. h sends message $\mu_{h4} = \{\textsc{false}, \textsc{blame}_h^{s_1}, \ell_h^{s_1}, n_R, 4, h\}\textsc{sig}_{u_h}$, which is independent of b by the above.
  * *Case 2* $O_h^1 = (\textsc{success}, M_h'^{s_1})$. h sends message $\mu_{h4} = \{\textsc{true}, C'_{\pi(1)h}, \ldots, C'_{\pi(N)h}, n_R, 4, h\}\textsc{sig}_{u_h}$. For $h \in H \backslash \{\alpha, \beta\}$, $C'_{\pi(i)h}$ is computed from the descriptors and keys received in earlier rounds, which are shown above to be independent of b. For $h \in \{\alpha, \beta\}$, h uses as its ciphertext the value $C_{m_0 h}$ generated in Phase 2 for descriptors containing the encryption of a seed matching the seed that is encrypted in d_{m_0}. h does similarly for descriptors with seeds matching the one in d_{m_1}. Otherwise, h computes ciphertexts $C'_{\pi(i)h}$ from the descriptors and keys received in earlier rounds. $C_{m_0 h}$ and $C_{m_1 h}$ are created above independently of b, and the previous messages received by h are shown above to be independent of b. Thus, the message μ_{h4} from h is independent of b.
- The additional inputs to A since his last output are messages μ_{h4}, $h \in H$, shown above to be independent of b. Thus the messages μ_{i4}, $i \in D$, received by $h \in H$ are independent of b.
- The messages μ_{h4}, $h \in H$, received by $h' \in H$ are shown above to be independent of b.

• Phase 5:

- The message $\mu_{h5} = \{\vec{V}_h, n_R, 5, h\}\textsc{sig}_{u_h}$ sent by $h \in H$ depends on the descriptors d_i obtained as an output of the shuffle, on $\textsc{go}_i$ received from each member i, and on the ciphertexts C'_{ij} received from each member i. These messages and outputs are shown above to be independent of b, and thus μ_{h5} is independent of b as well.

- The additional inputs to A since his last output are messages μ_{h5}, $h \in H$, shown above to be independent of b. Thus the messages μ_{i5}, $i \in D$, received by $h \in H$ are independent of b.
- The messages μ_{h5}, $h \in H$, received by $h' \in H$ are shown above to be independent of b.

- Phase 6: Member $h \in H$ creates message m'_i using the $\textsc{go}_j$ and the ciphertexts C'_{ij} received from each member j. Each of these is shown above to be independent of b, and so m'_i is also independent of b.

- Phase 7:
  - Each $h \in H$ sets the fail flag f_h^2 for the shuffle in this phase based on ciphertexts received in the messages μ_{j4} and μ_{j5} from every j. These messages are shown above to be independent of b, and thus each f_h^2 is independent of b.
  - C'^7 calls C'^2. The inputs to C'^2 from C'^7 are challenge users α and β, challenge messages $m_0^{c_2}$ and $m_1^{c_2}$ (to be specified), non-challenge messages $m_h^2 = A_h$ for $h \in H\backslash\{\alpha, \beta\}$, round number n_{R_2}, signing keys K, member ordering τ, fail flags f_h^2, and challenge bit b. We observe here that α, β, n_{R_2}, K, τ, and the f_h^2 are shown above to be independent of b. We consider two separate cases for the blame shuffle in order to show that all failures, messages, and outputs of the shuffle are independent of b. Which case applies depends only on the f_h^2 and thus is independent of b.
    * *Case 1*: $f_h^2 = \textsc{false}$ for all $h \in H$.
      It is shown above that all inputs to C'^2 are independent of b except $m_0^{c_2}$, $m_1^{c_2}$, $m_h^2 = A_h$, $h \in H\backslash\{\alpha, \beta\}$, and b itself. Each accusation A_h, $h \in H\backslash\{\alpha, \beta\}$, depends on the descriptor d_h, the shuffle output O_h^1 and the contents of the μ_{j4} received by h from all j. These are shown above to be independent of b, and so A_h is independent of b as well.
      We claim that the accusation A_{m_0} created by the member $h_0 \in \{\alpha, \beta\}$ that is assigned m_0 is created the same regardless of h_0. After showing this, we will be able to apply Lemma 12 to prove the anonymity of the shuffle. A_{m_0} depends on $O_{h_0}^1$, $\vec{V}_{h_0}$, and d_{h_0}.
      If $O_h^1 = (\textsc{failure}, \textsc{blame}_h^{s_1}, \ell_h^{s_1})$ for some $h \in \{\alpha, \beta\}$, then neither α nor β creates an accusation, and $A_{m_0} = 0^{\Lambda_a}$ regardless of h_0.
      Now suppose that $O_h^1 = (\textsc{success}, M_h'^{s_1})$ for all $h \in \{\alpha, \beta\}$. We observe that, although C'^2 does not strictly execute $\textsc{anonymize-s}$, after Phase 3 of the shuffle that challenger does simply execute $\textsc{anonymize-s}$ for each $h \in H$, assuming that he does not fail. If C'^2 had failed during the descriptor shuffle, of course, we would not have reached this phase, and therefore we can assume that he did not. In addition, the outcome is the same as if the entire $\textsc{gmp-shuffle}$ had been run because all $f_h^2 = \textsc{false}$ by assumption and the parameters K, τ, and n_{R_1} used for $\textsc{anonymize-s}$ are generated by $\textsc{setup-b}$ in the same way that $\textsc{setup-s}$ generates them. Thus, we observe that the proof of Lemma 2 applies to the descriptor shuffle. We are therefore guaranteed that the outputs $(\textsc{success}, M_h'^{s_1})$, $h \in \{\alpha, \beta\}$, are identical. We can then assume that $O_\alpha^1 = (\textsc{success}, M_\alpha'^{s_1}) = O_\beta^1$.
      If α receives a μ_{j4} with $\textsc{go}_j = \textsc{false}$, then β must as well. Otherwise, the equivocation would have been discovered and the shuffle deliberately failed by all honest

members, contradicting our assumption for this case. The accusation for both members in this case is empty, and so we can say that $A_{m_0} = 0^{\Lambda_a}$ regardless of h_0.
If α and β receive $\text{GO}_j = \text{TRUE}$ in all μ_{j4}, then any incorrect ciphertexts α receives in μ_{j4} must also be received by β. Otherwise, again, all honest users would have noticed the equivocation and caused the blame shuffle to fail, contradicting the case assumption. Thus, because α and β have the same sequence of descriptors, and because equality of hashes implies equality of the preimages by second-preimage resistance, we can conclude that α and β must receive all the same C'_{ij}. As stated above, accusation A_{m_0} depends only on d_{m_0}, $O^1_{h_0}$ and the GO_j and C_{ij} contained in each μ_{j4} received by h_0. We have shown that in this case all of these are equal for α and β, and thus A_{m_0} is indeed created the same regardless of h_0.
The above arguments apply to the accusation A_{m_1} created by the user $h_1 \in \{\alpha, \beta\}$ that is assigned m_1. Therefore, with $m_0^{c_2} = A_{m_0}$, $m_1^{c_2} = A_{m_1}$, and I as the set of all inputs from C'^7 to C'^2 except b, we can apply Lemma 12. We conclude that C'^2 fails during the blame shuffle with probability independent of b, that the messages sent to A are independent of b, and that the output O^2 is independent of b.

* *Case 2*: $f_h^2 = 1$ for some $h \in H$.
  In this case, we simply view C^7 as calling C'^2 with $A_h = h$ for all $h \in H$. Because the shuffle will fail, the challenger will fail if Z^2 is set to 0. Otherwise, $Z^2 = 1$, and the shuffle effectively uses h as the input message for each $h \in H$. In this case, b has no effect on the messages of each user and therefore no effect on the shuffle. Thus, C'^2 fails during the blame shuffle with probability independent of b, the messages sent to A during the shuffle are independent of b, and the shuffle output O^2 is independent of b.

- The message $\mu_{h7} = \{p', n_R, 7, h\}\text{SIG}_{u_h}$ sent by $h \in H$ depends on f_h^2 and the messages μ_{i5} and μ_{i4} received by h. These are shown above to be independent of b, and thus μ_{h7} is independent of b as well.
- The additional inputs to A since his last output are messages sent during the blame shuffle and μ_{h7}, $h \in H$, all of which are shown above to be independent of b. Thus the messages μ_{i7}, $i \in D$, received by $h \in H$ are independent of b.
- The messages μ_{h7}, $h \in H$, received by $h' \in H$ are shown above to be independent of b.
- For each $h \in H$, SUCCESS_h and BLAME_h are set differently in several different cases. Which case applies depends on O_h^2 and μ_{h5}, which are shown above to be independent of b. Thus which case is applied is also independent of b. For each case, SUCCESS_h and BLAME_h depend at most on $\text{BLAME}_h^{s_2}$; on the messages μ_{i3}, μ_{i4}, and μ_{i7} sent and received by h; on the blame-shuffle output O_h^2; and on the descriptor-shuffle output O_h^1. These are all shown above to be independent of b, and thus SUCCESS_h and BLAME_h are independent of b as well.
- Output messages M'_h, $h \in H$, are created depending on SUCCESS_h and the messages μ_{i4} sent and received by h. These are shown above to be independent of b, and thus M'_h is independent of b as well.
- Log ℓ_h, $h \in H$, depends on SUCCESS_h, the output of SETUP-B, all messages sent and received by h, and the shuffle outputs O_h^1 and O_h^2. These are all shown above to be independent of b, and thus ℓ_h is independent of b as well.

– The output O_h of GMP-BULK, $h \in H$, depends on SUCCESS$_h$, M'_h, BLAME$_h$, and ℓ_k. These are shown above to be independent of b, and thus O_h is independent of b as well.

We have thus shown that, given input $I = i$, for every execution of C'^7 when $b = 0$ there is an execution when $b = 1$ that occurs with the same probability and for which (*i*) F'^7 is the same, (*ii*) M is the same, and (*iii*) O is the same. □

Lemma 21. $\Delta(G^7) = 0$.

Proof. To prove this, we show that the steps of the anonymity game surrounding the challenge run are independent of b and use the previous lemma for the challenge run itself.

1. In Step 1, pre-challenge rounds of the bulk protocol are executed, which do not depend on b.
2. In Step 2, A sends C^7 the challenge participants α and β, the challenge messages m_0^c and m_1^c, and the non-challenge messages m_h, $h \in H \backslash \{\alpha, \beta\}$, which must be independent of b because all previous inputs to A were shown above to be independent of b.
3. Step 3 of the anonymity game is for the challenger to assign the messages of the challenge users according to b. However, we leave these variables undefined, as we have modified the challenger to create Game 7 such that they are not necessary.
4. The challenge run is executed during Step 4. We observe that C^7 first executes SETUP-B. This protocol takes only the long-term signing keys as input, and therefore its output $(n_R, n_{R_1}, n_{R_2}, K, \tau)$ is independent of b. Next C^7 calls C'^7 with inputs b and $I = (n_R, n_{R_1}, n_{R_2}, K, \tau, \alpha, \beta, m_0^c, m_1^c, \{m_h\}_{h \in H \backslash \{\alpha, \beta\}})$. I has been shown to be independent of b. Therefore, by applying Lemma 20, we can conclude that C^7 fails independently of b, and if it does not fail any messages M to A and outputs O are also independent of b.
5. In Step 5, the challenger executes further rounds of the protocol. The adversary's inputs up to this point have been shown to be independent of b, and thus these executions do not depend on b.
6. In Step 6, A outputs guess $\hat{b}$. All inputs to the adversary have been shown to be independent of b, and thus $\hat{b}$ is independent of b.

The game output $G^7(b)$ only depends on F^7 and $\hat{b}$. These have both been shown to be independent of b, and therefore

$$Pr[G^7(1) = 1] = Pr[G^7(0) = 1].$$

□

Taken together, the preceding lemmas show that the adversary has a negligible advantage in the anonymity game:

Theorem 6. *The* GMP-BULK *protocol maintains anonymity with k colluding members for any* $0 \leq k \leq N - 2$.

Proof. Let A be a probabilistic polynomial-time adversary. We denote the change in advantage between games i and j as $\epsilon_{ij} = \left|\Delta(G^j) - \Delta(G^i)\right|$. Using Lemmas 14 and 16, the advantage of A in the anonymity game with GMP-BULK is at most $2\left(\epsilon_{02} + 2\left(\epsilon_{35} + \epsilon_{56} + \epsilon_{67} + \Delta(G^7)\right)\right)$. By Lemma 21 this is $2\epsilon_{02} + 4\epsilon_{35} + 4\epsilon_{56} + 4\epsilon_{67}$. This quantity is negligible by Lemmas 15, 17, 18, and 19. □

6 Related Work

DISSENT's shuffle protocol builds directly on an anonymous data collection protocol by Brickell and Shmatikov (Brickell and Shmatikov 2006b), adding DoS resistance via new go/no-go and blame phases. DISSENT's bulk protocol is similarly inspired by DC-nets (Chaum 1988), which are computationally efficient and provide unconditional anonymity. DC-nets traditionally require nondeterministic "reservation" schemes to allocate the anonymous channel's communication bandwidth, however, and are difficult to protect against anonymous DoS attacks by malicious group members. Strategies exist to strengthen DC-nets against DoS attacks (Waidner and Pfitzmann 1989; Golle and Juels 2004), or to form new groups when an attack is detected (Sirer et al. 2004). DISSENT's use of a shuffle protocol to set up a *deterministic* DC-nets instance, however, cleanly avoids these DoS vulnerabilities while providing the additional guarantee that each member sends *exactly* one message per protocol run, a useful property for holding votes or assigning 1-to-1 pseudonyms.

Mix networks (Chaum 1981) offer high-latency but practical anonymous communication, and can be adapted to group broadcast (Perng, Reiter, and Wang 2006). Unfortunately, for many mix-network designs, anonymity is vulnerable to traffic analysis (Serjantov, Dingledine, and Syverson 2003) and performance is vulnerable to DoS attacks (Dingledine and Syverson 2002; Iwanik, Klonowski, and Kutylowski 2004). Cryptographically-verifiable mixes (Neff 2001; Furukawa and Sako 2001; Adida 2006) are a possible solution to DoS attacks and a potential replacement for our shuffle protocol. These algorithms require exotic and complex cryptography, however, bringing efficiency costs and implementation and verification challenges.

Low-latency designs can provide fast and efficient communication supporting a wide variety of applications, but they typically provide much weaker anonymity than DISSENT. For example, onion routing (Goldschlag, Reed, and Syverson 1999; Dingledine, Mathewson, and Syverson 2004), a well-known and practical approach to general anonymous communication on the Internet, is vulnerable to traffic analysis by adversaries who can observe streams going into and out of the network (Syverson, Tsudik, Reed, and Landwehr 2000). Similarly, Crowds (Reiter and Rubin 1999) is vulnerable to statistical traffic analysis when an attacker can monitor many points across the network. Herbivore (Goel, Robson, Polte, and Sirer 2003) provides unconditional anonymity, but only within a small subgroup of the total group of participants. k-anonymous transmission protocols (von Ahn, Bortz, and Hopper 2003) provide anonymity only when most members of a group are honest.

We thus observe a tradeoff between security, efficiency, and possible applications. Furthermore, many cryptographic attacks have been discovered against specific anonymity protocols. These protocols are often complex and contain subtle flaws in design, security proofs, or security definitions.

For example, many attacks have been identified against mix-network schemes, some against schemes that offered proofs of security. A simple yet powerful attack against one scheme (Park, Itoh, and Kurosawa 1994) trivially breaks an honest member's anonymity if an attacker can create a ciphertext related to that member's ciphertext (Pfitzmann 1994; Pfitzmann and Pfizmann 1990). An attack on the integrity of a scheme claimed to be probably secure (Jakobsson 1998) was given by Mitomo and Kurosawa (2000). A corrupted mix server can alter intermediate ciphertexts, affecting the corresponding output messages, without being detected. Several attacks on the anonymity and robustness of another scheme (Golle, Zhong, Boneh, Jakobsson, and Juels 2002) claimed secure were presented by Wikström (2003). These attacks exploited previously identified (Pfitzmann 1994; Pfitzmann and Pfizmann 1990; Desmedt and Kurosawa 2000) general design flaws as well as the ability of mix servers to use incorrect and specially-prepared inputs. Abe and Imai (2003) described two anonymity attacks on mix-net designs (Jakobsson and Juels 2001; Golle, Zhong, Boneh, Jakobsson, and Juels 2002), possible when members collude with a server and even with completely-honest mix servers. Later, the authors pointed out (Abe and Imai 2006) that some flaws are related to weak security definitions. Even newly proposed schemes still succumb to previous attacks. A recent work of Khazaei, Terelius, and Wikström (2012) points out flaws in the design of Allepuz and Castello (2010) that facilitate attacks against anonymity and integrity, some of which are based on previously-described attacks (Pfitzmann 1994).

These attacks show that obtaining a provably secure anonymous communication protocol is a surprisingly complex task. It requires a considerable amount of effort and careful attention to every design detail of a protocol. Indeed, only relatively recently has a framework for rigorous security proofs been available for mix networks (Wikström 2004).

7 Conclusion and Future Work

DISSENT is a novel, provably secure protocol for anonymous and accountable group communication. DISSENT allows a well-defined group of participants to exchange variable-length messages anonymously, while resisting the traffic analysis and anonymous DoS attacks effective against mix-networks, DC-nets, and onion routing. DISSENT improves upon previous shuffled-send primitives by adding accountability—the ability to trace misbehaving nodes—and by eliminating the message padding requirements of earlier schemes. DISSENT guarantees anonymity, integrity, and accountability, and has been shown practical for anonymous communication within moderate-size groups.

We have presented an improved version of this protocol that fixes several flaws in the original design. We have precisely defined its security properties and provided detailed security proofs.

Future work includes exploring ways to achieve scalability in order to accommodate large groups, as well as interactivity to make DISSENT suitable for latency-sensitive applications.

APPENDIX

Here we describe in more detail some of the security flaws discovered in the DISSENT protocol of Corrigan-Gibbs and Ford (2010). Flaws were discovered affecting each of the desired security criteria: integrity, anonymity, and accountability. We also briefly mention the technique we adopted to fix each problem. By following a rigorous proof methodology for the improved protocol, we can have high confidence that these fixes have not not introduced problems of their own. Note that the terminology and notation used here is that of Corrigan-Gibbs and Ford (2010).

Anonymity

- Ciphertext replay attack in shuffle

  Flaw: The adversary can replay a ciphertext C_i of some user i from an earlier run of the shuffle by submitting C_i as his own ciphertext. Then the adversary looks for the "inner" ciphertext C_i' that appeared at the end of the anonymization phase (Phase 3) in both this run and the earlier run. The adversary can conclude that the message contained in that inner ciphertext, which was successfully decrypted in the earlier run, were sent by i.

Fix: New "outer" encryption keys are generated in each run of the shuffle.

- Message descriptor replay attack in the bulk protocol

  Flaw: The adversary can replay the message descriptor d_i of some user i received in an earlier run of the bulk protocol by submitting it as his own descriptor. d_i contains an encrypted seed for i that does not generate a ciphertext with a hash matching the included hash. In the previous run, user i was looking for a slot with descriptor matching d_i and used a precomputed ciphertext for it instead of using the included seed. In this run, i is not looking for it, and because the hash of the ciphertext won't match the one included in d_i, i will send an empty ciphertext. This identifies i as the owner of the message revealed during the slot containing d_i in the previous run.

  Fix: New encryption keys for the seeds in the message descriptors are generated in each run of the bulk protocol.

- Ciphertext equivocation attack in the bulk protocol

  Flaw: The adversary can target user i as the suspected owner of a slot $\pi(j)$ by sending an incorrect ciphertext $\hat{C}_{jk}$ to i in Phase 3 and sending correct ciphertexts to all other members. Then if a valid accusation comes out of the blame phase (Phase 5), i must be the owner of the slot, that is, $i = j$.

  Fix: Rebroadcast the ciphertexts before the blame shuffle, and then have users that observe ciphertext equivocation "break" the blame shuffle and then send evidence of equivocation to exonerate themselves and expose the equivocation member.

- Adversary copies encrypted seeds during the bulk protocol

  Flaw: An adversary in the last position of the shuffle can copy the ciphertext containing a message descriptor into his own slot. An honest member only looks for one message descriptor matching her own, and therefore the owner of the copied descriptor will use the encrypted seed in the second slot containing her descriptor, the ciphertext won't match the hash, and so she will send an empty ciphertext. This identifies herself as the owner of the slot containing the first copy of the descriptor, which does have its message revealed.

  Relatedly, it appears technically possible for an adversary to create a wholly new descriptor that contains the encrypted seed that a slot owner creates for herself in her own descriptor. IND-CCA2 doesn't appear to have a type of non-malleability that would prevent this kind of copying. Thus simply looking for all copies of a member's descriptor isn't enough, as the adversary could potentially target a member by copying out her encrypted seed from her encrypted message descriptor into a totally different descriptor. The member who uses different ciphertexts for the same seeds is the owner of the (original non-modified) descriptors.

  Fix: Have members look for all copies of their encrypted seed, and use the same precomputed ciphertext in each of those slots.

Accountability

- Ciphertext duplication attack in the shuffle

Flaw: An adversary in the first position of shuffle can use as his own ciphertext submission the ciphertext that an honest member submits into the shuffle. The shuffle fails when duplicate ciphertexts are observed, and both the honest and dishonest members are exposed. This violates accountability, which prohibits exposing honest members.

Fix: Members must first commit publicly to their ciphertext submission using non-malleable commitments and including their identity (e.g. their shuffle position) in the commitment.

- Equivocation in proceeding to blame in the shuffle

  Flaw: If all $\text{GO}_i = \text{TRUE}$ in the verification phase (Phase 4), but dishonest j pretends to honest k that j received $\text{GO}_i = \text{FALSE}$ from i by only sending blame data in the last phase (i.e. executing Phase 5b), while sending his private key w_j to all other members (i.e. executing Phase 5a with respect to them), then it is not clear if liveness assumption implies that k can eventually get enough blame data from the other members (who see everything go correctly, proceed to Phase 5a, and finish the protocol) to expose a faulty member.

  Fix: The key release and blame phases (Phase 5a and 5b) are now unconditionally run in sequence. A member must justify in the blame phase not sending out aprivate key in the key-release phase with enough evidence to expose another member.

Integrity

- Ciphertext equivocation attack in the bulk protocol

  Flaw: The adversary can send a bad ciphertext to just one member, who, if not the owner, will never receive a valid accusation and so will complete successfully without all honest members' messages.

  Fix: As described earlier as the fix to an anonymity attack, we rebroadcast the ciphertexts before the blame shuffle, and then we have users that observe ciphertext equivocation "break" the blame shuffle and then send evidence of equivocation to exonerate themselves and expose the equivocating member.

References

Abe, M. and H. Imai (2003). Flaws in some robust optimistic mix-nets. In *ACISP*.

Abe, M. and H. Imai (2006). Flaws in robust optimistic mix-nets and stronger security notions. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*.

Adida, B. (2006). *Advances in cryptographic voting systems*. Ph. D. thesis, Cambridge, MA, USA.

Allepuz, J. P. and S. G. Castello (2010). Universally verifiable efficient re-encryption mixnet. In *Electronic Voting*. GI.

Bellare, M., A. Desai, D. Pointcheval, and P. Rogaway (1998). Relations among notions of security for public-key encryption schemes. *Advances in Cryptology —CRYPTO '98*.

Brickell, J. and V. Shmatikov (2006a). Efficient anonymity-preserving data collection. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Brickell, J. and V. Shmatikov (2006b). Efficient anonymity-preserving data collection.

Castro, M. and B. Liskov (1999, February). Practical byzantine fault tolerance. In *OSDI*.

Chaum, D. (1981, February). Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*.

Chaum, D. (1988, January). The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*.

Clarke, I., O. Sandberg, B. Wiley, and T. W. Hong (2000, July). Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability*.

Corrigan-Gibbs, H. and B. Ford (2010). Dissent: Accountable anonymous group messaging. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*.

Davenport, D. (2002, April). Anonymity on the Internet: why the price may be too high. *Communications of the ACM 45*(4), 33–35.

Desmedt, Y. and K. Kurosawa (2000). How to break a practical mix and design a new one. In *EUROCRYPT*.

Dingledine, R., N. Mathewson, and P. Syverson (2004). Tor: the second-generation onion router. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*.

Dingledine, R. and P. Syverson (2002, March). Reliable MIX cascade networks through reputation. In *Financial Cryptography*.

Dolev, D., C. Dwork, and M. Naor (2000). Nonmalleable cryptography. *SIAM J. Comput.*.

Douceur, J. R. (2002, March). The Sybil attack. In *1st International Workshop on Peer-to-Peer Systems*.

Furukawa, J. and K. Sako (2001, August). An efficient scheme for proving a shuffle. In *CRYPTO*.

Goel, S., M. Robson, M. Polte, and E. G. Sirer (2003, February). Herbivore: A Scalable and Efficient Protocol for Anonymous Communication. Technical Report 2003-1890, Cornell University.

Goldschlag, D., M. Reed, and P. Syverson (1999, February). Onion routing for anonymous and private internet connections. *Communications of the ACM 42*(2), 39–41.

Goldwasser, S., S. Micali, and R. L. Rivest (1995). A digital signature scheme secure against adaptive chosen-message attacks.

Golle, P. and A. Juels (2004, May). Dining cryptographers revisited. *Eurocrypt*.

Golle, P., S. Zhong, D. Boneh, M. Jakobsson, and A. Juels (2002). Optimistic mixing for exit-polls. In *Asiacrypt 2002, LNCS 2501*.

Haeberlen, A., P. Kouznetsov, and P. Druschel (2007, October). PeerReview: Practical accountability for distributed systems. In *21st SOSP*.

Iwanik, J., M. Klonowski, and M. Kutylowski (2004, September). DUO-Onions and Hydra-Onions — failure and adversary resistant onion protocols. In *IFIP CMS*.

Jakobsson, M. (1998). Flash mixing. EUROCRYPT.

Jakobsson, M. and A. Juels (2001). An optimally robust hybrid mix network. In *In Principles of Distributed Computing - PODC 01*.

Khazaei, S., B. Terelius, and D. Wikström (2012). Cryptanalysis of a universally verifiable efficient re-encryption mixnet. In *International conference on Electronic Voting Technology/Workshop on Trustworthy Elections*.

Lamport, L. (1998). The part-time parliament. *TOCS*.

Mitomo, M. and K. Kurosawa (2000). Attack for flash MIX. In *In Advances in Cryptology - ASIACRYPT 2000, LNCS*, pp. 192–204. Springer-Verlag.

Neff, C. A. (2001, November). A verifiable secret shuffle and its application to e-voting. In *ACM CCS*.

Park, C., K. Itoh, and K. Kurosawa (1994). Efficient anonymous channel and all/nothing election scheme. In *EUROCRYPT*.

Perng, G., M. Reiter, and C. Wang (2006). M2: Multicasting mixes for efficient and anonymous communication. In *26th ICDCS*, pp. 59–59.

Pfitzmann, B. (1994). Breaking an efficient anonymous channel. In *EUROCRYPT*.

Pfitzmann, B. and A. Pfizmann (1990). How to break the direct RSA-implementation of mixes. In *EUROCRYPT*.

Reiter, M. K. and A. D. Rubin (1999). Anonymous web transactions with crowds. *Communications of the ACM 42*(2), 32–48.

Rogaway, P. and T. Shrimpton (2004). Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *Fast Software Encryption - FSE'04*.

Serjantov, A., R. Dingledine, and P. Syverson (2003). From a trickle to a flood: Active attacks on several mix types. *Information Hiding*.

Sirer, E. G. et al. (2004, September). Eluding carnivores: File sharing with strong anonymity. In *ACM SIGOPS EW*.

Stein, E. (2003). Queers anonymous: Lesbians, gay men, free speech, and cyberspace. *Harvard Civil Rights-Civil Liberties Law Review*.

Stinson, D. R. (2005). *Cryptography: Theory and Practice, Third Edition (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC.

Stone, B. and M. Richtel (2007). The hand that controls the sock puppet could get slapped. *New York Times*.

Syverson, P., G. Tsudik, M. Reed, and C. Landwehr (2000, July). Towards an Analysis of Onion Routing Security. In *Design Issues in Anonymity and Unobservability*.

Teich, A., M. S. Frankel, R. Kling, and Y. Lee (1999, May). Anonymous communication policies for the Internet: Results and recommendations of the AAAS conference. *Information Society*.

von Ahn, L., A. Bortz, and N. J. Hopper (2003). k-anonymous message transmission. In *10th CCS*.

Waidner, M. and B. Pfitzmann (1989, April). The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability. In *Eurocrypt*.

Wallace, J. D. (1999, December). Nameless in cyberspace: Anonymity on the internet. Cato Briefing Paper No. 54.

Wikström, D. (2003). Five practical attacks for "optimistic mixing for exit-polls". In *Selected Areas in Cryptography*.

Wikström, D. (2004). A universally composable mix-net. In *TCC*.

Yale Law Journal (1961, June). The constitutional right to anonymity: Free speech, disclosure and the devil. *Yale Law Journal 70*(7), 1084–1128.