Security Analysis of Accountable Anonymity in Dissent

EWA SYTA, HENRY CORRIGAN-GIBBS, SHU-CHUN WENG, DAVID WOLINSKY,
BRYAN FORD, Yale University
AARON JOHNSON, U.S. Naval Research Laboratory

Users often wish to communicate anonymously on the Internet, for example in group discussion or instant messaging forums. Existing solutions are vulnerable to misbehaving users, however, who may abuse their anonymity to disrupt communication. Dining Cryptographers Networks (DC-nets) leave groups vulnerable to denial-of-service and Sybil attacks, mix networks are difficult to protect against traffic analysis, and accountable voting schemes are unsuited to general anonymous messaging.

DISSENT is the first general protocol offering provable anonymity and accountability for moderate-size groups, while efficiently handling unbalanced communication demands among users. We present an improved and hardened DISSENT protocol, define its precise security properties, and offer rigorous proofs of these properties. The improved protocol systematically addresses the delicate balance between provably *hiding* the identities of well-behaved users, while provably *revealing* the identities of disruptive users, a challenging task because many forms of misbehavior are inherently undetectable. The new protocol also addresses several non-trivial attacks on the original DISSENT protocol stemming from subtle design flaws.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms: Algorithms, Security

Additional Key Words and Phrases: Anonymous communication, accountable anonymity, provable security

ACM Reference Format:
Ewa Syta, Aaron Johnson, Henry Corrigan-Gibbs, Shu-Chun Weng, David Wolinsky, and Bryan Ford. 2013. Security Analysis of Accountable Anonymous Group Communication in Dissent. *ACM Trans. Info. Syst. Sec.* 0, 0, Article 0 (January 2013), 30 pages.
DOI : http://dx.doi.org/10.1145/0000000.0000000

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 enabling people and groups with controversial or unpopular views to communicate and organize without fear of personal reprisal.Yet anonymity makes it difficult to trace or exclude misbehaving participants. Online protocols providing stronger anonymity, such as mix-networks [Chaum 1981; Adida 2006], onion routing [Goldschlag et al. 1999; Dingledine et al. 2004a], 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, within a well-defined *group* of participants whose membership is closed and known to its members [Corrigan-Gibbs and Ford 2010]. DISSENT enables members of such a group to send *anonymous* messages – either to each other, to the whole

The work of Ewa Syta, Henry Corrigan-Gibbs, Shu-Chun Weng, David Wolinsky, and Bryan Ford was supported by the Defense Advanced Research Projects Agency (DARPA) and SPAWAR Systems Center Pacific, Contract No. N66001-11-C-4018. Aaron Johnson 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.

DOI : http://dx.doi.org/10.1145/0000000.0000000

group, or to a non-member – such that the receiver knows that *some* member sent the message, but no one knows *which* member. DISSENT also holds members *accountable* – not by compromising their anonymity, but rather by ensuring that communication resources are allocated fairly among all communicating members, and that any disruption results in the identification of some malicious member during a "blame" process. Misbehaving 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 a group's deliberations.

DISSENT builds on the sender-verifiable shuffle of Brickell and Shmatikov [2006], combining a similar shuffle scheme with DC-net techniques for efficient bulk communication. DISSENT uses only readily available cryptographic primitives and handles arbitrarily large messages and unbalanced loads efficiently. Each member sends *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 et al. 1999; Clarke et al. 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 2.

DISSENT was introduced by Corrigan-Gibbs and Ford [2010], who sketched the basic protocol and informal security arguments, described practical usage considerations, and experimentally evaluated the performance of a prototype implementation. This paper revisits and substantially modifies the DISSENT protocol, to offer a precise formal definition and exposition of the protocol and a rigorous analysis of its security properties. Though the overall structure and function remains similar, the new protocol re-formulates and heavily revises the original to address flaws our formal analysis revealed, and to provide a modular framework for defining and rigorously reasoning about the DISSENT protocol's nontrivial composition of verifiable shuffle and DC-nets techniques. While our primary focus is on hardening the DISSENT protocol through rigorous formal analysis, some of the techniques we develop may be of independent interest, such as our methods of modular reasoning and ensuring accountability throughout complex protocols, including capabilities to identify and *prove* the source of any disruption attempt without compromising other security properties.

For this improved protocol we are able to offer proofs of all three key security properties: integrity, accountability, and anonymity. Obtaining a provably secure protocol 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 2). Section 3.4 discusses in greater detail the discovered flaws and the resulting changes to the protocol. The full version of this paper [Syta et al. 2013] details the discovered flaws and their fixes, and also includes full details of protocols, properties, and proofs.

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

2. RELATED WORK

DISSENT's shuffle protocol builds on an anonymous data collection protocol by Brickell and Shmatikov [2006], adding accountability via new go/no-go and blame phases. DISSENT's bulk protocol is inspired by DC-nets [Chaum 1988], an information coding approach to anonymity.

Mix networks [Chaum 1981] offer high-latency but practical anonymous communication, and can be adapted to group broadcast [Perng et al. 2006]. Unfortunately, for many mix-network designs, anonymity is vulnerable to traffic analysis [Serjantov et al. 2003] and performance is vulnerable to active disruption [Dingledine and Syverson 2002; Iwanik et al. 2004]. Cryptographically-verifiable mixes [Neff 2001; Furukawa and Sako 2001; Adida 2006] are a possible solution to disruption at-

tacks and a potential alternative to our shuffle protocol. However, verifiable shuffles alone generally verify only a shuffle's *correctness* (i.e., that it is a permutation), and not its *randomness* (i.e., that it ensures anonymity). All existing techniques of which we are aware to assure a shuffle's randomness and anonymity, in the presence of compromised members, require passing a batch of messages through a series of independent shuffles, as in DISSENT or mix-networks [Dingledine et al. 2004b].

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 et al. 1999; Dingledine et al. 2004a], 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 et al. 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 et al. 2003] provides unconditional anonymity, but only within a small subgroup of all participants. k-anonymous transmission protocols [von Ahn et al. 2003] provide anonymity only when most members of a group are honest.

We thus observe tradeoffs among security, efficiency, and possible applications. Further, 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 mix network designs have claimed to provide anonymity [Park et al. 1994; Jakobsson 1998; Jakobsson and Juels 2001; Golle et al. 2002; 2002; Allepuz and Castello 2010], only to be broken in later work [Pfitzmann and Pfizmann 1990; Pfitzmann 1994; Mitomo and Kurosawa 2000; Abe and Imai 2003; Wikström 2003; Khazaei et al. 2012b] or to have weaknesses identified in their security definitions [Abe and Imai 2006]. DISSENT has been designed to avoid these and other common flaws [Desmedt and Kurosawa 2000]. However, they 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 recently has a framework for rigorous security proofs been available for mix networks [Wikström 2004].

Finally, there are several ways in which anonymity protocols have provided some notion of accountability. In general, they may offer accountability either for protocol violations or for undesirable content or behavior [Feigenbaum et al. 2011]. DISSENT and other protocols based on DC-nets [Waidner and Pfitzmann 1989; Golle and Juels 2004], and verifiable shuffles [Neff 2003; Khazaei et al. 2012a; Bayer and Groth 2012] aim to hold users accountable for protocol violations. Each client remains anonymous unless he misbehaves by breaking the rules of the protocol. In contrast, some other anonymity protocols [von Ahn et al. 2006; Diaz and Preneel 2007; Backes et al. 2014] attempt to unmask a client's identity if the client's actions or the contents of his messages are unacceptable or unpopular, when a set of explicitly or implicitly defined parties agrees to.

3. INFORMAL PROTOCOL OVERVIEW

DISSENT is designed to be used in a group setting. Each member i of a group is associated with a long-term public signature key pair (u_i, v_i), where u_i is the private signing key and v_i is the public verification key. We assume the signature key pair represents each member's public identity, and that members cannot easily obtain such identities. This assumption makes DISSENT's accountability property enforceable, so that an exposed misbehaving member cannot trivially leave and rejoin the group under a new (public) identity. Members can obtain such identities from trusted certification authorities, or agree among themselves on a static set of group members and corresponding signature keys. Specific approaches to group formulation, however, are out of scope of this paper.

DISSENT provides a *shuffled send* communication primitive that ensures sender anonymity among the 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 to all members. 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 pro-

tocol provides anonymous communication for fixed-length messages, and the GMP-BULK protocol builds on this to provide efficient anonymous communication of arbitrary-length messages.

3.1. The General Messaging Protocol

A Group Messaging Protocol GMP is a 3-tuple of the following algorithms: SETUP(v_i), ANONYMIZE($m_i, K, n_R, \tau, k_h, f_i$) and VERIFY-PROOF($p_j, \ell_i$). All group members collectively run the SETUP and ANONYMIZE algorithms on their own inputs, while anyone, including users other than group members, can independently run the VERIFY-PROOF algorithm.

SETUP takes a member's public verification key v_i as input and outputs one or more session nonces n_R, a set K of all members' verification keys, a well-known ordering of members τ, a hash key k_h 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' signature key pairs or create a binding between a user's identity and his signature key pair; rather, it uses long-term verification keys submitted by each member and allows group members to agree on the set K of verification keys for a particular protocol run.

ANONYMIZE takes a message m_i, a set K of members' verification keys, one or more round nonces n_R, an ordering of members τ, a hash key k_h, and optionally a flag f_i 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. The goal of ANONYMIZE is to broadcast anonymously the set of messages submitted by group members. If a protocol run succeeds from a given member's perspective, then she outputs the anonymized messages. Otherwise, the protocol run fails and the group member produces a set of *blame* proofs that verifiably reveal at least one misbehaving member responsible for causing the failure. In the security properties to be defined below we will demand that ANONYMIZE always *either* succeeds completely *or* produces a valid blame proof on failure; this guarantee of *accountability* is both one of DISSENT's key points of novelty and the source of some of the most difficult technical challenges this paper addresses.

VERIFY-PROOF takes a proof p_j of a member j's misbehavior and a log ℓ_i as input, and outputs either TRUE if p_j indeed proves that j misbehaved given the protocol history represented by log ℓ_i, or FALSE otherwise. Any third party can use VERIFY-PROOF to check a proof of j's misbehavior.

3.2. The GMP-Shuffle Protocol

The GMP-SHUFFLE protocol enables the anonymous exchange of equally sized messages. GMP-SHUFFLE builds on the protocol of Brickell and Shmatikov [2006], which provides cryptographically strong anonymity, and improves it by adding *go/no-go* and *blame* phases to trace and hold accountable any group member disrupting the protocol.

GMP-SHUFFLE consists of three algorithms: SETUP-S, ANONYMIZE-S, and VERIFY-PROOF-S. All group members run SETUP-S to agree on common parameters for ANONYMIZE-S. During ANONYMIZE-S, members first establish ephemeral inner and outer encryption keys, then each member doubly onion-encrypts his secret message using the inner and outer public keys of all members. After collectively shuffling all encrypted messages and removing the outer layers of encryption, members verify that the resulting set includes each member's inner encryption of their message and no member observed any failures thus far. If all steps are performed correctly, members reveal their inner private keys, allowing each member to recover the full set of secret messages and successfully complete the protocol. However, if any member observes misbehavior at any step of the protocol, the protocol fails for that member. Following a failure, members perform a blame procedure whose goal is to identify at least one culprit member and to produce a verifiable proof of his misbehavior. To facilitate the blame process, all members always exchange their protocol logs, and those who did not reveal their inner private keys share their outer private keys, allowing each member to trace the protocol's execution. Although members reveal full logs, each member's anonymity is protected since honest members never reveal both private keys. Therefore, a member can always perform the blame procedure and produce proofs of misbehavior regardless of how the protocol completed for

other members. Afterwards, anyone can run VERIFY-PROOF-S to validate purported proof(s) of any member's misbehavior in ANONYMIZE-S.

Section 6 details the GMP-SHUFFLE protocol and Section 8 proves its security.

3.3. The GMP-Bulk Protocol

The GMP-BULK protocol uses ideas from DC-nets to transmit variable-length messages anonymously, but leverages the GMP-SHUFFLE protocol to prearrange the DC-nets transmission schedule, guaranteeing each member exactly one message slot per round. GMP-BULK also reuses GMP-SHUFFLE to broadcast anonymous *accusations*, to blame a culprit who may have caused a protocol failure.

Like GMP-SHUFFLE, GMP-BULK consists of three algorithms, SETUP-B, ANONYMIZE-B, and VERIFY-PROOF-B. All members use SETUP-B to agree on common parameters for any given protocol round and VERIFY-PROOF-B to verify proofs of misbehavior produced in ANONYMIZE-B. During ANONYMIZE-B, each member creates and anonymously broadcasts via ANONYMIZE-S a *message descriptor*, which defines pseudorandom sequences all *other* members must send in a subsequent DC-nets exchange, such that XORing all sequences together yields a permuted set of secret messages. Cryptographic hashes in the message descriptors enable members to verify the correctness of each others' bulk transmissions, ensuring message integrity and accountability throughout. A successful protocol run allows a member to recover all secret messages of honest members. If any member observes a failure at any step, however, he prepares and shares with other members an accusation naming the culprit member. All members, including those who did not observe any failures, participate in the blame phase to give each member an opportunity to broadcast an anonymous accusation via ANONYMIZE-S (anonymity is needed because some accusations can only be formed by the owner of a corrupted message) and distribute evidence to support the accusation. After validating accusations, members who experienced failures perform the blame procedure to find at least one faulty member and produce a proof of his misbehavior, exposing the culprit member.

The GMP-BULK protocol is detailed in Section 7 and Section 8 proves its security.

3.4. Comparison to the Original DISSENT Protocol

In analyzing the original DISSENT protocol we identified several attacks, which this paper fixes. Anonymity could be broken by replaying protocol inputs in subsequent rounds, by providing incorrect ciphertexts to some members at certain points and correct ones to the rest, or by copying ciphertexts at other points. Accountability for disruption could be avoided by copying protocol inputs from honest members, and dishonest members could falsely accuse honest ones by rearranging valid signed messages to create phony logs. Finally, through equivocation a dishonest member could cause some honest members to terminate successfully and skip the blame process, while other honest members observe failure but are unable to terminate the protocol with a valid blame proof.

To fix these flaws, we made several non-trivial modifications to the original protocol. To prevent replay attacks we added key generation steps (GMP-SHUFFLE Phase 1 and GMP-BULK Phases 1a and 1b). To prevent equivocation attacks, where a member sends different versions of a message instead of broadcasting it to all members, we added rebroadcast steps (GMP-BULK Phase 5), and have members intentionally cause intermediate protocol failures (GMP-BULK Phases 3 and 7) when equivocation is observed. We add non-malleable commitments (GMP-SHUFFLE Phases 2a and 2b) 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.

The protocol changes result add a communication overhead of four broadcasts to GMP-SHUFFLE. For GMP-BULK, the overhead is four broadcasts plus the messages exchanged as a part of ANONYMIZE-A in Phase 7, which is now non-optional. The most significant cost in practice comes from always running ANONYMIZE-S in GMP-BULK Phase 7, due to its serialized structure, which we found to be dominant in related experiments [Corrigan-Gibbs et al. 2013].

4. SECURITY MODEL AND DEFINITIONS

We model the adversary with a probabilistic polynomial-time Turing machine A. We allow him to control a fixed subset of k group members. We call the members that he controls *dishonest* and the members that he does not control *honest*. We suppose that the members communicate using non-private but authenticated channels. That is, a message that appears to i to be from member j is guaranteed to be from j, but the adversary can observe all such messages when they are sent. We also give the adversary access to member outputs.

The security properties we wish the protocol to satisfy are *integrity*, *accountability*, and *anonymity*. The definitions we give of these are precise versions of the notions used by Corrigan-Gibbs and Ford [2010]. We express these properties as games between the adversary A and a challenger C or $C(b)$, where $b \in \{0, 1\}$ will be a hidden bit input to algorithm C. We denote the adversary's output from this game with A^C. For all games, C executes the protocol with A by running the protocol algorithm for each honest member and allowing A to act as the dishonest members. When any message is sent from an honest member, C also sends a copy of the message with its source and destination to A. C also sends protocol outputs of honest members to the adversary. Our definitions are round-based and allow the adversary to execute arbitrary sequential executions of the protocol. There is an implicit initial step of all games in which the challenger generates long-term signature key pairs (u_i, v_i) for each honest member i. In general, our definitions require that the adversary "win" the security games with *negligible* probability, that is, with probability that goes to zero with the security parameter asymptotically faster than any inverse polynomial. Output probabilities are taken over the randomness of both the adversary and challenger.

The *integrity game* for protocol GMP is as follows:

(1) As many times as A requests, C takes message inputs for the honest members from A and uses them to execute GMP with A.
(2) A and C execute a challenge run of GMP for which C takes message inputs for the honest members from A.
(3) C outputs 1 if, at any time after the start of the challenge round, (*i*) an honest member i outputs $(\text{SUCCESS}, M_i')$ such that M_i' does not contain exactly N messages or does not include the multiset of input messages from the honest members, or (*ii*) two honest members i and j produce outputs $(\text{SUCCESS}, M_i')$ and $(\text{SUCCESS}, M_j')$ such that M_i' and M_j' contain different messages or contain a different ordering of the messages. Otherwise, once all honest members complete, C outputs 0.

Definition 4.1. A protocol offers *integrity* if the challenger output in the integrity game is 1 with negligible probability.

The *accountability game* for protocol GMP is as follows:

(1) As many times as A requests, C takes message inputs for the honest members from A and uses them to execute GMP with A.
(2) A and C execute a challenge run of GMP for which C takes message inputs for the honest members from A.
(3) As many times as A requests, C takes message inputs for the honest members from A and uses them to execute GMP with A.
(4) C outputs 1 if (*i*) at the end of the challenge run an honest member i produces an output of $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$, where BLAME_i is empty or contains $p_j \in \text{BLAME}_i$ such that $\text{VERIFY-PROOF}(p_j, \ell_i) \neq \text{TRUE}$, or (*ii*) at any time after the challenge run starts A sends C $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$ such that $p_j \in \text{BLAME}_i$ for honest j, $\text{VERIFY-PROOF}(p_j, \ell_i) = \text{TRUE}$, and the output of SETUP in ℓ_i includes the nonce of the challenge round and assigns the long-term verification key v_j to j. Otherwise C outputs 0 once all protocol runs are completed.

Definition 4.2. A protocol offers *accountability* if the challenger output in the accountability game is 1 with negligible probability.

We use the *anonymity game* described by Brickell and Shmatikov [2006]. Note that this definition will only make sense for an adversary of size $0 \leq k \leq N - 2$.

(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.
(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 $h \notin \{\alpha, \beta\}$ and sends them to $C(b)$.
(3) $C(b)$ assigns $m_\alpha = m_b^c$ and $m_\beta = m_{1-b}^c$.
(4) A and $C(b)$ execute the protocol.
(5) 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.
(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 $\left|Pr\left[A^{C(0)} = 1\right] - Pr\left[A^{C(1)} = 1\right]\right|$.

Definition 4.3. A protocol maintains *anonymity* if the advantage in the anonymity game is negligible.

We note that this definition implies anonymity among all honest users and not just pairs, because for any pair of assignments of honest messages to honest users, we can turn one assignment into the other via a sequence of pairwise swaps, each of which are guaranteed by the definition to change the adversary's output distribution by a negligible amount. We observe that these properties do not imply that the protocol completes for all members, and, in fact, we cannot guarantee that DISSENT 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 et al. 2007] should be applicable via a wrapper protocol to ensure liveness. Moreover, we note that when every member follows the protocol, not only does it complete but it succeeds.

5. TECHNICAL PRELIMINARIES

5.1. Definitions

Member i *broadcasts* a message by sending it to all other members. A dishonest member might *equivocate* during a broadcast by sending different messages to different members. A run of GMP *succeeds* for member i if the ANONYMIZE algorithm terminates with output $(\text{SUCCESS}, M_i')$, and it *fails* if the ANONYMIZE algorithm terminates with output $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$.

5.2. Cryptographic Primitives and Security Assumptions

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

Hash functions: We use a standard definition [Stinson 2005] of a *keyed hash function* and will denote the hash of message m using key k_h as $\text{HASH}_{k_h}\{m\}$. We assume that the hash function used is collision resistant [Rogaway and Shrimpton 2004].

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: $c = \{\ldots\{m\}_{y_1}^{R_1}\ldots\}_{y_N}^{R_N}$. We omit R when an encryption's random inputs need not be saved.We assume that the underlying public-key cryptosystem provides indistinguishable ciphertexts against a chosen-ciphertext attack, that is, that the cryptosystem is IND-CCA2 secure [Bellare et al. 1998]. We also 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 describe a ciphertext as *invalid* when it can be recognized with no private information that decryption would result in an error. This includes as an important special case the value $\perp$, which is the output upon a decryption error.

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 if 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 provides existential unforgeability under an adaptive chosen message attack, that is, that it is EUF-CMA secure [Goldwasser et al. 1995].

Pseudorandom Number Generator: We use a standard definition [Stinson 2005] of a *pseudo-random 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 $\text{PRNG}\{L, s\}$.

Non-interactive Commitments: We use a *non-interactive commitment* that is concurrent non-malleable [Pandey et al. 2008]. 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. We note that minor protocol modifications would allow interactive commitments instead.

SETUP Consensus: We assume that the protocol used by SETUP produces a consensus output in the presence of the adversary. The adversary signals honest members to begin a SETUP round and can repeat with additional rounds. Each honest member i uses verification key v_i as the input. Our assumption is that, with probability 1 for every round, honest members that terminate produce the same output and that the output includes *i*) nonces never used in a previous round, *ii*) the key v_i for honest member i and some key for each dishonest member, and *iii*) a uniformly random hash key.

6. GMP-SHUFFLE

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, τ, and k_h as inputs to ANONYMIZE-S. This algorithm also takes a fail flag f_i 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. Figure 1 shows the normal execution (solid lines) and failure-handling execution (dashed lines) of ANONYMIZE-S.

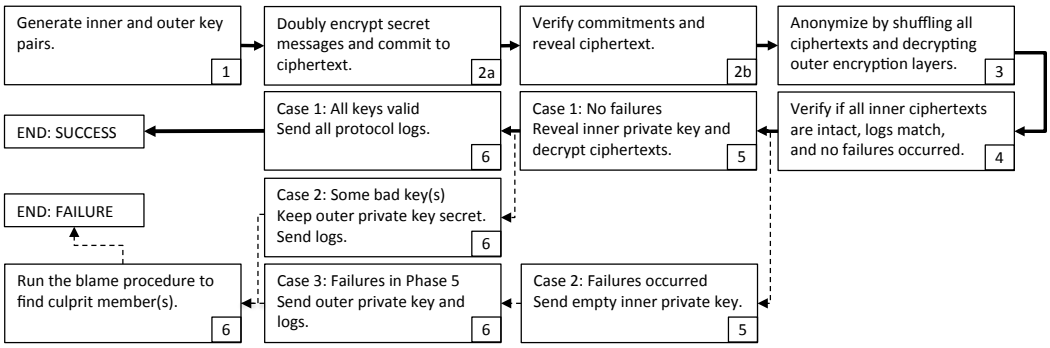

Fig. 1. Flow of the ANONYMIZE-S algorithm

6.1. The Setup-S Algorithm

$\text{SETUP-S}(v_i)$ takes each member's verification key v_i as input and outputs a session nonce n_R, a list K of all members' verification keys, an ordering of members τ, a fixed message length L, and a hash key k_h. As described in Section 3.1, this algorithm can be implemented using tools such as a standard consensus protocol.

6.2. The Anonymize-S Algorithm

The purpose of ANONYMIZE-S$(m_i, K, n_R, \tau, k_h, f_i)$ when run by each member in a group on the collective input messages M is to produce anonymized messages M'.

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 8, 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. A complete log ℓ_i includes all messages member i sent and received within SETUP-S and ANONYMIZE-S as defined in the protocol description.

Table I contains the checks applied by a member 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.

Table I. A list of checks a member applies within the GMP-SHUFFLE protocol.

Check 1 ($\mathtt{c}_1$):	Incomplete or inconsistent log.	Check 7 ($\mathtt{c}_7$):	Incorrect commitment or invalid ciphertext or identity in Phase 2b.
Check 2 ($\mathtt{c}_2$):	Mismatched inner key pair in Phase 5.	Check 8 ($\mathtt{c}_8$):	Incorrect set of permuted ciphertexts after decryption in Phase 3.
Check 3 ($\mathtt{c}_3$):	Empty inner key in Phase 5 without a GO$_k$ = FALSE or a bad HASH$_{k_h}\{\vec{B}\}$.	Check 9 ($\mathtt{c}_9$):	Invalid ciphertext(s) after decryption in Phase 3.
Check 4 ($\mathtt{c}_4$):	Mismatched or empty outer private key in Phase 6 without justification.	Check 10 ($\mathtt{c}_{10}$):	Duplicate ciphertext(s) after decryption in Phase 3.
Check 5 ($\mathtt{c}_5$):	Invalid public key in Phase 1.	Check 11 ($\mathtt{c}_{11}$):	Incorrect GO$_j$ in Phase 4.
Check 6 ($\mathtt{c}_6$):	Invalid commitment in Phase 2a.	Check 12 ($\mathtt{c}_{12}$):	Bad HASH$_{k_h}\{\vec{B}\}$ in Phase 4.

Algorithm description. ANONYMIZE-S$(m_i, K, n_R, \tau, k_h, f_i)$

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 not, 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} creating $C_i' = \{m_i\}_{I_N^{pub}:I_1^{pub}}$. 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 new key for j to allow the protocol to go forward. Now member i calculates $X_i = \text{COMMIT}\{C_i, i\}$ and broadcasts $\mu_{i2a} = \{X_i, n_R, 2\mathtt{a}, i\}\text{SIG}_{u_i}$. After receiving such a message from every other member, member i verifies that they include valid commitments, and if not GO$_i$ is set to FALSE.

Phase 2b: Data Submission. Member i sends to member 1 an opening of her commitment: $\mu_{i2b} = \{\text{OPEN}\{X_i\}, n_R, 2\mathtt{b}, i\}\text{SIG}_{u_i}$. Member 1 verifies that each μ_{i2b} successfully opens X_i and results in a valid ciphertext and position 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 $\mu_{13} = \{\vec{C}_1, n_R, 3, 1\}\text{SIG}_{u_1}$ to member 2. 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 forms 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 $f_i = \text{TRUE}$ then member i always sets $\text{GO}_i = \text{FALSE}$. If $f_i = \text{FALSE}$ 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 $\vec{C}_N$. Member i broadcasts $\mu_{i4} = \{\text{GO}_i, \text{HASH}_{k_h}\{\vec{B}\}, n_R, 4, i\}\text{SIG}_{u_i}$.

Phase 5: Key Release and Decryption.

Case 1. If $\text{GO}_j = \text{TRUE}$ and $\text{HASH}_{k_h}\{\vec{B}_j\} = \text{HASH}_{k_h}\{\vec{B}_i\}$ for every member j, then member i broadcasts her inner private key in $\mu_{i5} = \{I_i^{sec}, n_R, 5, i\}\text{SIG}_{u_i}$. 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 there is at least one empty key or if any key pair fails the verification, then i sets $\text{SUCCESS}_i = \text{FALSE}$. 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 $\text{GO}_j = \text{FALSE}$ or $\text{HASH}_{k_h}\{\vec{B}_j\} \neq \text{HASH}_{k_h}\{\vec{B}_i\}$ for any member j, then member i sends to all members an empty string instead of I_i^{sec}. Member i broadcasts $\mu_{i5} = \{0, n_R, 5, i\}\text{SIG}_{u_i}$ and sets SUCCESS_i to FALSE.

Phase 6: Blame.

Case 1. $\text{SUCCESS}_i = \text{TRUE}$. 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, broadcasts $\mu_{i6} = \{\vec{T}, n_R, 6, i\}\text{SIG}_{u_i}$, and outputs $(\text{SUCCESS}, M'_i)$, which completes the protocol.

Case 2. $\text{SUCCESS}_i = \text{FALSE}$ and for every member j, $\text{GO}_j = \text{TRUE}$ and $\text{HASH}_{k_h}\{\vec{B}_j\} = \text{HASH}_{k_h}\{\vec{B}_i\}$. Member i keeps her outer private key O_i^{sec} secret and broadcasts $\mu_{i6} = \{0, \vec{T}, n_R, 6, i\}\text{SIG}_{u_i}$, which contains an empty string instead of her key and a vector $\vec{T}$ of all signed messages she sent and received in Phases 1–5.

Case 3. $\text{SUCCESS}_i = \text{FALSE}$ and for any member j, $\text{GO}_j = \text{FALSE}$ or $\text{HASH}_{k_h}\{\vec{B}_j\} \neq \text{HASH}_{k_h}\{\vec{B}_i\}$. Member i broadcasts $\mu_{i6} = \{O_i^{sec}, \pi_i, \vec{T}, n_R, 6, i\}\text{SIG}_{u_i}$, 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.

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 μ_{j6} from every other member j, member i discards any message in $\vec{T}$ that is not properly signed or does not have the correct round or phase number. Member i then checks that each member j's $\vec{T}$ 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 $\vec{T}$ logs of other members.

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. 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 $p_j = (j, \mathsf{c}_1)$, where c_1 indicates the failed check number, and adds p_j to BLAME_i. Then, member i outputs $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$, which concludes the protocol.

Otherwise, member i uses the messages in the $\vec{T}$ logs to complete her view of Phases 1–5 of the execution and thus can proceed to examine the rest of the protocol. Member i begins by examining the release of inner and outer private keys. She adds (j, c_2) to BLAME_i for every member j whose

revealed I_j^{sec} does not match I_j^{pub}. Then, for every member j who sent an empty inner private key without any $\text{GO}_k = \text{FALSE}$ or incorrect broadcast hash, member i adds $(j, \mathsf{c_3})$ to BLAME_i. For every member j who revealed her outer private key in Phase 6, member i checks if O_j^{sec} is valid and corresponds to O_j^{pub}. In addition, for every member j who sent an empty outer private key, member i verifies that it is justified in that every $\text{GO}_k = \text{TRUE}$ and all broadcast hashes matched in Phase 4. 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 adds $(j, \mathsf{c_4})$ to BLAME_i.

Member i next replays the protocol from the perspective of every member j. She creates a proof p_j and adds it to blame BLAME_i for every member j who

— sends an invalid public key in Phase 1 ($p_j = (j, \mathsf{c_5})$);
— sends an invalid commitment in Phase 2a ($p_j = (j, \mathsf{c_6})$);
— sends an opening that does not successfully open her commitment or does not result in a valid ciphertext and identity j in Phase 2b ($p_j = (j, \mathsf{c_7})$);
— in the case that matching outer key pairs are received from all members, does not send a valid permutation of the decrypted valid ciphertexts in Phase 3 ($p_j = (j, \mathsf{c_8})$), sends a ciphertext that produces duplicate or invalid ciphertexts after decryption ($p_j = (j, \mathsf{c_9})$), or sends an outer ciphertext which at any point decrypts to the same ciphertext as another member's outer ciphertext ($p_j = (j, \mathsf{c_{10}})$; or
— improperly reports in Phase 4 $\text{GO}_j = \text{FALSE}$ based on the messages sent to j in Phases 1–5 in the case that matching outer key pairs are received from all members ($p_j = (j, \mathsf{c_{11}})$), or sends an incorrect $\text{HASH}_{k_h}\{\vec{B}_j\}$ ($p_j = (j, \mathsf{c_{12}})$).

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

6.3. Verify-Proof-S Algorithm

VERIFY-PROOF-S(p_j, ℓ_i) verifies a member j's misbehavior. VERIFY-PROOF-S takes as input a proof p_j and a log ℓ_i of a protocol run. VERIFY-PROOF-S outputs TRUE if p_j is a verifiable proof based on ℓ_i and FALSE otherwise.

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 p_j is invalid, output FALSE.

Step 2: Log verification. Examine ℓ_i to ensure that all included messages are properly signed, contain a correct round nonce given the execution of SETUP-S, and contain 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, which is clear from the descriptions of those algorithms, then output FALSE and stop. Otherwise, verify that the logs of all sent and received messages revealed by each member j in Phase 6 are complete and consistent. That is, verify that the $\vec{T}$ in every μ_{j6} contains all messages sent and received in Phases 1–5 such that the messages are properly signed, include correct phase and round numbers, and the contents of the corresponding messages match for every member j. If any $\vec{T}$ is incomplete or inconsistent and $c \neq \mathsf{c_1}$, then output FALSE and stop. If $c \neq \mathsf{c_1}$, augment ℓ_i with those messages in the $\vec{T}$ logs not seen by i.

Step 3: Proof verification decision. Examine log ℓ_i to verify that it contains a record of a check c failed by member j. If yes, output 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.

7. GMP-BULK

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. The fail flag f_i is always set to FALSE for any execution of ANONYMIZE-B. Figure 2 shows the normal execution (solid lines) and failure-handling execution (dashed lines) of ANONYMIZE-B.

7.1. The Setup-B Algorithm

SETUP-B(v_i) takes each member's verification key v_i as input, and outputs a session nonce n_R identifying a run of ANONYMIZE-B, session nonces n_{R_1} and n_{R_2} identifying runs of ANONYMIZE-S in Phase 3 and Phase 7 of ANONYMIZE-B respectively, a list K of members' verification keys , an ordering of members τ, and a hash key k_h. As described in Section 3.1, this algorithm can be implemented using tools such as a standard consensus protocol.

7.2. The Anonymize-B Algorithm

ANONYMIZE-B$(m_i, K, n_R, n_{R_1}, n_{R_2}, \tau, k_h)$ takes a message m_i of variable length L and the output of SETUP-B as input. The algorithm operates in phases. Member i sends at most one unique message $\mu_{i\phi}$ in phase ϕ. If a protocol run succeeds, then member i outputs $(\text{SUCCESS}, M_i')$, where, as we show in Section 8, M_i' consists of N messages including every message submitted by an honest member. If a protocol run fails, then member i produces $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$. 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. Table II contains the checks applied by a member during the protocol, listed in the order they are applied. Each check is associated with a check number that ANONYMIZE-B uses to form a proof of a particular misbehavior and VERIFY-PROOF-B uses to confirm a record of that misbehavior.

Table II. The list of checks a member applies within the GMP-BULK protocol.

Check 1 ($\mathtt{c}_1$):	Equivocation in Phase 4 or Phase 5.	Check 4 ($\mathtt{c}_4$):	Unverifiable proof included in Phase 4.
Check 2 ($\mathtt{c}_2$):	Failure of ANONYMIZE-S in Phase 3 or Phase 7 without justification.	Check 5 ($\mathtt{c}_5$):	Invalid public key sent in Phase 1a.
Check 3 ($\mathtt{c}_3$):	Empty or incorrect ciphertext(s) in Phase 4.	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.

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

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 y_j from every member j, member i notifies other members about the set of keys she receives. Member i creates a vector $\vec{K}_i^e = \{\mu_{11a}, \ldots, \mu_{N1a}\}$ 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, and sets L_i to the length of message m_i, or $L_i = 0$ if i has no message to send.

Case 1. Successful key verification. Member i verifies each set of public keys received in Phase 1b to ensure she has the same set of valid public keys. If every $\vec{K}_j^e$ contains the same set of valid public keys, then for each member j, i chooses a random seed s_{ij} and for each $j \neq i$ generates L_i pseudorandom bits from s_{ij} to obtain ciphertext $C_{ij} = \text{PRNG}\{L_i, s_{ij}\}$, 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 $C_{ii} = C_{i1} \oplus \ldots \oplus C_{i(i-1)} \oplus m_i \oplus C_{i(i+1)} \oplus \ldots \oplus C_{iN}$. Member i computes hashes $H_{ij} = \text{HASH}_{k_h}\{C_{ij}\}$, encrypts each seed s_{ij} with j's public key to form $S_{ij} = \{s_{ij}\}_{y_j}^{R_{ij}}$, and collects the H_{ij} and S_{ij} into vectors $\vec{H}_i = (H_{i1}, \ldots, H_{iN})$ and $\vec{S}_i = (S_{i1}, \ldots, S_{iN})$. Member i forms a message descriptor $d_i = \{L_i, \vec{H}_i, \vec{S}_i\}$, which has a fixed length Λ_d.

Case 2. Failed key verification. If any $\vec{K}_j^e$ contains a non-matching set of keys or any $\vec{K}_j^e$ contains an invalid key, then member i creates an empty message descriptor $d_i = 0^{\Lambda_d}$ of length Λ_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 $d_i = \{L_i, \vec{H}_i, \vec{S}_i\}$ of length Λ_d.

Phase 3: Message Descriptor Shuffle. Each member i runs the ANONYMIZE-S protocol described in Section 6 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 and distributes it to other members. If member j sent an invalid key, then member i sets $p' = (j, \mathsf{c_5}, \mu_{j1a})$, where $\mathsf{c_5}$ 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 $p' = 0$. Then 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 shares her blame set $\text{BLAME}_j^{s_1}$ and log $\ell_j^{s_1}$, sets $\text{GO}_j = \text{FALSE}$, and broadcasts $\mu_{j4} = \{\text{GO}_j, \text{BLAME}_j^{s_1}, \ell_j^{s_1}, n_R, 4, j\}\text{SIG}_{u_j}$.

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 $\text{HASH}_{k_h}\{C_{ij}\}$ 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, H_{ij} is not a valid hash value, or $\text{HASH}_{k_h}\{C_{ij}\} \neq H_{ij}$, then j sets C'_{ij} to an empty ciphertext, $C'_{ij} = 0$. Member j now broadcasts each C'_{ij} in π-shuffled order $\mu_{j4} = \{\text{GO}_j, C'_{\pi(1)j}, \ldots, C'_{\pi(N)j}, n_R, 4, j\}\text{SIG}_{u_j}$.

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} that includes $\text{GO}_j = \text{FALSE}$ into a vector $\vec{V}_k$.

Case 2. If $\text{GO}_j = \text{TRUE}$ for every member j, then member k checks each C'_{ij} she receives from every member j against the corresponding H_{ij} from descriptor d_i. If C'_{ij} is empty or $\text{HASH}_{k_h}\{C'_{ij}\} \neq H_{ij}$, then message slot $\pi(i)$ was corrupted and member k reports this fact to other group members. Member k adds each message μ_{j4} that contains an empty or incorrect C'_{ij} to a vector $\vec{V}_k$.

Case 3. If $\text{GO}_j = \text{TRUE}$ for every member j and all ciphertexts are correct, then member i 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 $m'_i = C'_{i1} \oplus \ldots \oplus C'_{iN}$.

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 ANONYMIZE-S. However, before running it, members look for evidence of equivocation in the previous two phases. 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 invalid signature or incorrect member identifier, round or phase number, then member i 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 protocol to distribute the 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 $p'_i = (j, \mathsf{c}_1, \mu_{j4}, \mu'_{j4})$. 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 $p'_i = 0$. Member i then broadcasts $\mu_{i7} = \{p'_i, n_R, 7, i\}\text{SIG}_{u_i}$.

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: $O_k = (\text{FAILURE}, \text{BLAME}_k^{s_2}, \ell_k^{s_2})$. Member k sets $\text{SUCCESS}_k = \text{FALSE}$. Then k considers every $p_i = (i, c) \in \text{BLAME}_k^{s_2}$. If $c \neq \mathsf{c}_{11}$, then i could not have justifiably caused the blame shuffle to fail, and so k adds (i, c_2) to BLAME_k. Otherwise $c = \mathsf{c}_{11}$, 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 with correct phase and round numbers) for some member j, then k adds (j, c_1) to BLAME_k. Otherwise, k adds (i, c_2) to BLAME_k.

Case 2: $O_k = (\text{SUCCESS}, M_k^{s_2})$ and $\vec{V}_k = \{\}$. Member k sets $\text{SUCCESS}_k = \text{TRUE}$.

Case 3: $O_k = (\text{SUCCESS}, M_k^{s_2})$ and $\vec{V}_k$ includes ciphertexts. Member k checks the validity of every accusation $A_i = (j, \pi(i), s_{ij}, R_{ij})$ in $M_k^{s_2}$. 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 $\text{HASH}_{k_h}\{\text{PRNG}\{L_i, s_{ij}\}\}$, where L_i is also obtained from d_i. If the accusation is valid, then member k adds (j, c_3) to BLAME_k. If $M_k^{s_2}$ includes no valid accusation targeting an incorrect ciphertext received by k, then k sets $\text{SUCCESS}_k = \text{TRUE}$, otherwise, k sets $\text{SUCCESS}_k = \text{FALSE}$.

Case 4: $O_k = (\text{SUCCESS}, M_k^{s_2})$ 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 $\ell_i^{s_1}$ contains n_{R_1} as the round number that is the result of SETUP-B and that VERIFY-PROOF-S$(p_j, \ell_i^{s_1}) = \text{TRUE}$ for some $p_j \in \text{BLAME}_i^{s_1}$. If not, then member k blames i by adding (i, c_4) to BLAME_k. If so, then k considers every $p_j \in \text{BLAME}_i^{s_1}$ such that VERIFY-PROOF-S$(p_j, \ell_i^{s_1}) = \text{TRUE}$. If $p_j \neq (j, \mathsf{c}_{11})$, then member k adds (j, c_2) to BLAME_k. If $p_j = (j, \mathsf{c}_{11})$, 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 or two different versions of y_ℓ (in properly signed messages with correct phase and round numbers), then member k adds (ℓ, c_5) or (ℓ, c_6) to BLAME_k, respectively. Otherwise, k adds (j, 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 that includes 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 $(\text{FAILURE}, \text{BLAME}_k, \ell_k)$.

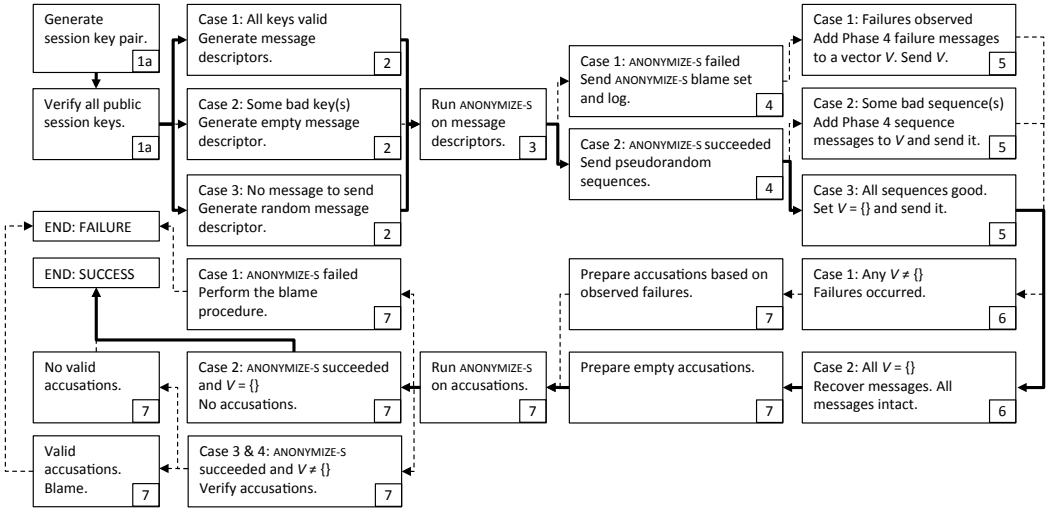

Fig. 2. Flow of the ANONYMIZE-B algorithm

7.3. Verify-Proof-B Algorithm

VERIFY-PROOF-B(p_j, ℓ_i) is used to verify member misbehavior. It takes as input a proof p_j and a log ℓ_i. The proof p_j should be a tuple (j, c), where j is a member's identifier and c indicates the failed check. The 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. VERIFY-PROOF-B outputs TRUE if ℓ_i shows that j indeed failed check c 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 p_j is invalid, then output FALSE and stop.

Step 2: Log verification. Examine ℓ_i, and discard all messages that are not properly signed, do not contain a correct round nonce given the execution of SETUP-B, or do not contain a correct phase number. 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, which is clear from the descriptions of those algorithms, as well as the output of ANONYMIZE-S in Phases 3 and 7, then output FALSE.

Step 3: Proof verification decision. Examine log ℓ_i to verify that it contains a record of a check c failed by member j. If yes, output TRUE, and FALSE otherwise.

8. PROOFS

In this section we prove that DISSENT satisfies the definitions of integrity, accountability, and anonymity given in Section 4. We generally organize proofs as a sequence of games [Shoup 2004].

8.1. Notation and Definitions

Let G be the set of all members participating in the protocol and H be the set of honest members. A group member i *blames* member j if $p_j \in \text{BLAME}_i$ upon a protocol failure resulting in $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$. This p_j is a *verifiable proof* of j's misbehavior given ℓ_i if VERIFY-PROOF$(p_j, \ell_i) = \text{TRUE}$. We say that i *exposes* j if member i produces a verifiable proof for j given a log ℓ_i in which SETUP outputs the long-term signature verification key v_j for j.

8.2. Integrity

THEOREM 8.1. *The* GMP-SHUFFLE *protocol offers integrity.*

PROOF. We consider the modified integrity game in which C outputs 0 upon observing a hash collision. Such an observation occurs when C receives in Phase 6 different inputs to the Phase 4 hash of broadcast messages that have the same hash value. Modifying the game in this way can only change the probability that C outputs 1 by a negligible amount by reduction to the game defining hash collision resistance.

For C to output 1 an honest member i must succeed during the challenge run. 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 $\text{HASH}_{k_h}\{\vec{B}_j\} = \text{HASH}_{k_h}\{\vec{B}_i\}$ 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 hash collisions are not observed, member i is in possession of the same $\vec{C}_N$ and inner public keys as every other honest member j. 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. The probability that C outputs 1 is thus zero, which implies that the probability that C outputs 1 in the original game is negligible. □

THEOREM 8.2. *The* GMP-BULK *protocol offers integrity.*

PROOF. We consider the modified integrity game in which C outputs 0 when the shuffle in Phase 3 or 7 fails to provide integrity or when a hash collision is observed for a hash in a message descriptor. By reduction to the integrity game with GMP-SHUFFLE and Theorem 8.1, the shuffles fail to provide integrity with negligible probability. Then by reduction to the game defining collision resistance, different ciphertexts with the same hash are seen by C with negligible probability. Thus this game modification changes the output distribution only negligibly.

Suppose that C outputs 1. There must exist 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 $\text{HASH}_{k_h}\{C_{jk}'\} = H_{jk}$ for all ciphertexts received by i in Phase 4 or no valid accusation is received in Phase 7 for any ciphertext such that $\text{HASH}_{k_h}\{C_{jk}'\} \neq H_{jk}$.

Every honest member for whom the descriptor shuffle is successful 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, 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.

Therefore C cannot output 1 in the modified game, and the probability that C outputs 1 in the original game is negligible. □

8.3. Accountability

We use the following sequence of games to prove that the GMP-SHUFFLE and GMP-BULK protocols offer accountability:

Game 0: Adversary A and challenger C^0 play the accountability game.

Game 1: A interacts with a challenger C^1 that is the same as C^0 except that it outputs 0 when a signature forgery or hash collision is observed.

Game 2: A interacts with a challenger C^2 that is the same as C^1 except that it outputs 0 when, at the end of the challenge run, an honest member i produces an output of $(\text{FAILURE}, \text{BLAME}_i, \ell_i)$ with empty BLAME_i or containing $p_j \in \text{BLAME}_i$ such that $\text{VERIFY-PROOF}(p_j, \ell_i) \neq \text{TRUE}$.

Game 3: A interacts with a challenger C^3 that always outputs 0.

It will be shown for both protocols that the output distribution of challenger C changes negligibly between each sequential pair of games. This holds for Game 0 and Game 1 quite directly from the security properties of the signature scheme and hash function. The main argument that it holds for Game 1 and Game 2 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 produces a verifiable proof. The argument for Game 2 and Game 3 relies on the fact that 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 cannot be exposed.

8.3.1. The GMP-Shuffle Protocol. Here we denote by C^i the output of the challenger in Game i when running GMP-SHUFFLE.

LEMMA 8.3. $\left|Pr[C^2 = 1] - Pr[C^0 = 1]\right|$ *is negligible.*

PROOF. Any forged signature observed by C^0 could be used to win the signature game that defines the EUF-CMA property. Therefore, by the assumption that the signature scheme is EUF-CMA secure, forged signatures must occur with negligible probability. Similarly, any observed hash collision could be used to win the game that defines the collision resistance of the hash function, and so one must occur with negligible probability. Thus the output distributions of C^0 and C^1 are negligibly close.

Challengers C^1 and C^2 differ only when an honest member fails but produces either no proof of misbehavior or a proof that doesn't verify. We will show that this never happens, that is, that whenever $\text{SUCCESS}_i = \text{FALSE}$ for honest member i, 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 logs of exactly those executions 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 $(j, \mathsf{c_1})$ 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 $\text{GO}_k = \text{FALSE}$ 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 $\text{SUCCESS}_i = \text{FALSE}$, and furthermore that any such p_j is a verifiable proof given log ℓ_i. Therefore the output distributions of C^1 and C^2 are identical. □

LEMMA 8.4. $\left|Pr[C^3 = 1] - Pr[C^2 = 1]\right|$ *is negligible.*

PROOF. The output of C^2 and C^3 is different only when an honest member is exposed but signature forgeries and hash collisions are not observed. Suppose that this is true, where A produces a proof p_j and log ℓ_i such that VERIFY-PROOF-S$(p_j, \ell_i) =$ TRUE and SETUP-S outputs the challenge-run nonce and assigns j her long-term signature verification key. 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. Furthermore, each message identifies the round and phase for which that message was sent. Therefore we can

assume that each message $\mu_{k\phi}$ sent by j in any phase of the challenge run appears with the same contents in ℓ_i.

Given these facts, we can show that for each of the 12 checks the needed log evidence cannot exist. An honest member j sends the same message to every member in any given phase. Thus, $c \neq \mathtt{c_1}$. An honest j always sends a correct message according to the protocol specification. Thus, $c \notin \{\mathtt{c_2}, \mathtt{c_5}, \mathtt{c_6}, \mathtt{c_7}, \mathtt{c_{12}}\}$. In certain cases the messages sent by j depend on his perception of the protocol execution based on the behavior of other members. An honest j always sends a correct message that reflects his perception of the protocol execution, as indicated in j's $\vec{T}$, which is consistent with ℓ_i as established before. Thus, $c \notin \{\mathtt{c_3}, \mathtt{c_4}, \mathtt{c_8}, \mathtt{c_9}, \mathtt{c_{11}}\}$. Lastly, if $c = \mathtt{c_{10}}$ then it must be the case that the adversary was able to produce a different commitment to a value that at some point in the shuffle is equal to that of j. This happens with at most negligible probability because the concurrent non-malleability of commitment guarantees that a simulator *not* given any honest commitments must be able to produce such a commitment, which would imply the ability to guess the encryption of one of the honest member's inputs, violating the IND-CCA2 security of encryption.

Therefore, there is no value of c for which VERIFY-PROOF-S could output TRUE given ℓ_i, except with negligible probability, and so the output distributions of C^2 and C^3 differ by a negligible amount. □

THEOREM 8.5. *The* GMP-SHUFFLE *protocol offers accountability.*

PROOF. Lemmas 8.3 and 8.4 show that the challenger output distribution in the accountability game is negligibly close to the challenger output in Game 3. C^3 always outputs 0, and thus the probability that C outputs 1 in the accountability game is negligible. □

8.3.2. The GMP-Bulk Protocol. Here we denote by C^i the output of the challenger in Game i when running GMP-BULK.

LEMMA 8.6. $\left|Pr[C^2 = 1] - Pr[C^0 = 1]\right|$ *is negligible.*

PROOF. Any forged signature or hash collision observed by C^0 could be used to win the security game that defines EUF-CMA or collision resistance, respectively. Thus, those security assumptions imply that such events must occur with negligible probability, and so the output distributions of C^0 and C^1 are negligibly close.

Challengers C^1 and C^2 differ only when an honest member fails but produces either no verifiable proof of misbehavior or a proof that doesn't verify. We will show that this happens with negligible probability by first showing that any proof produced by an honest member is verifiable and second by showing that some proof is always produced by an honest member upon failure except with negligible probability.

In VERIFY-PROOF-B, proof verification of p_j (Step 1) always succeeds for honest member i because 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. By examining each check, we can see that for each one VERIFY-PROOF-B requires either exactly the same conditions on ℓ_i or strictly fewer to output true for p_j as are needed for ANONYMIZE-B to create p_j. Thus, VERIFY-PROOF-B$(p_j, \ell_i) =$ 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, $\text{SUCCESS}_i =$ FALSE upon protocol completion only these cases: (1) the blame shuffle fails, (2) the blame shuffle succeeds and outputs a valid accusation, (3) some μ_{j4} contains $\text{GO}_j =$ FALSE. We consider each case and identify a proof p that is added to BLAME_i in each one.

In case (1), by reduction to the accountability game with GMP-SHUFFLE and Theorem 8.5, there must exist a verifiable proof $(j, c) \in \text{BLAME}_i^{s_2}$ given $\ell_i^{s_2}$ except with negligible probability. 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 provides evidence against ℓ in μ_{k3}.

Thus, if GMP-BULK fails for honest member i, BLAME_i is non-empty and only contains verifiable proofs given ℓ_i, except with negligible probability. This implies that the different between the output distributions of C^1 and C^2 is negligible. □

LEMMA 8.7. $\left|Pr[C^3 = 1] - Pr[C^2 = 1]\right|$ *is negligible.*

PROOF. The output of C^2 and C^3 is different only when an honest member is exposed but signature forgeries and hash collisions are not observed. Suppose this is true for honest member j, and the adversary produces a proof p_j and log ℓ_i such that VERIFY-PROOF-B(p_j, ℓ_i) = TRUE and SETUP-S outputs the challenge-run nonce and assigns j her long-term signature verification key.

To pass the initial proof verification of VERIFY-PROOF-B (Step 1), it must be the case that $p_j = (c, j)$. To pass the log verification (Step 2), the log ℓ_i must be complete. Each message in that log identifies the sender and is signed by that sender. Because signature forgeries are not observed and each message includes round and phase numbers, ℓ_i must include the message j sent in each phase of the challenge run.

Given these facts, we can show that for each of the 6 proof-verification checks (Step 3) the needed log evidence cannot exist with non-negligible probability. Honest member j sends the same message to every member in any given phase. Thus, $c \notin \{\mathsf{c}_1, \mathsf{c}_6\}$. $c \neq \mathsf{c}_2$, because if j causes a shuffle failure she provides a justification, and if she does not cause a failure, shuffle accountability (Theorem 8.5) implies that she cannot be exposed except with negligible probability, by reduction to the accountability game with GMP-SHUFFLE. $c \neq \mathsf{c}_3$, because j never sends incorrect ciphertexts, each empty ciphertext is justified by an incorrect message descriptor, and in both cases the message descriptors in ℓ_i are verified to be the ones received by j using the $\vec{T}$ log sent by j at the end of the descriptor shuffle. $c \neq \mathsf{c}_4$, because in case of GO_j = FALSE, j either sends the justifying evidence of equivocation or the verifiable proof of shuffle misbehavior that Theorem 8.5 and a reduction to the accountability game with GMP-SHUFFLE guarantees must exist, except with negligible probability. Lastly, an honest j always sends a valid key, and thus $c \neq \mathsf{c}_5$.

Therefore, there is no value of c for which VERIFY-PROOF-B could output TRUE on proof p_j given ℓ_i, except with negligible probability. This implies that C^2 and C^3 have negligible difference between their output distributions. □

THEOREM 8.8. *The* GMP-BULK *protocol offers accountability.*

PROOF. Lemmas 8.6 and 8.7 show that the challenger output distribution in the accountability game is negligibly close to the challenger output in Game 3. C^3 always outputs 0, and thus the probability that C outputs 1 in the accountability game is negligible. □

8.4. Anonymity

We prove that GMP-SHUFFLE and GMP-BULK maintain anonymity by sequentially modifying the original anonymity-game challenger C so that in Game i adversary A plays the anonymity game with challenger C^i. For Game i we define a "game output" G^i that is closely related to the adversary's output. Let $\Delta(G^i)$ 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 generally omit the challenge bit b that is technically an input to the game outputs, the challengers, and random variables that we define as a function of the challengers. Let $\overline{b}$ be the complement of bit b: $\overline{b} = 1 - b$. Let $h_1, h_2, \ldots, h_{N-k}$ be the honest users in the order they appear in the member permutation τ produced by SETUP.

8.4.1. The GMP-Shuffle Protocol. We show that the adversary's advantage in winning the anonymity game with GMP-SHUFFLE is negligible by proving that the game's advantage changes negligibly between neighboring games and is zero in the final game.

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, and let F^i indicate whether or not the challenger failed in Game i.

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 differs from C 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 $\text{HASH}_{k_h}\{\vec{B}_j\} = \text{HASH}_{k_h}\{\vec{B}_{h_1}\}$ 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 $\text{HASH}_{k_h}\{\vec{B}_j\} \neq \text{HASH}_{k_h}\{\vec{B}_{h_1}\}$ 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. This is also the result if C^0 observes a hash collision. 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 negligibly close to $1/2$ the advantage of A in the anonymity game:

LEMMA 8.9. *$\Delta(G^0)$ is negligibly close to* $(1/2)\left|Pr\left[A^{C(0)} = 1\right] - Pr\left[A^{C(1)} = 1\right]\right|$.

PROOF. By reduction to the game defining collision resistance, a hash collision is observed with negligible probability. With no hash collisions, C^0 acts like C except that for any execution either $Z^0 = 0$ or $Z^0 = 1$ causes failure. Z^0 is chosen independently of the rest of the game. Thus, the

probability of failure is negligibly close to $1/2$, and the game executions have the same distribution given no failure. When C^0 fails, G^0 is random, and otherwise G^0 is set to the output of A. □

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

PROOF. We prove the lemma by reduction to the IND-CCA2 game, that is, by constructing a distinguisher $D(b)$ that has a non-negligible advantage in the IND-CCA2 game if $|Pr[G^1(b) = 1] - Pr[G^0(b) = 1]|$ is non-negligible. Let b_D be the challenge bit in the IND-CCA2 game. D interacts with the IND-CCA2-game challenger $C_D(b_D)$ and A. D executes the anonymity game just as C^0 does, except replacing certain key generation, encryption, and decryption operations by interaction with C_D.

In the case that $Z = 0$, D gets a public encryption key from C_D and uses it as the outer public key $O^{pub}_{h_1}$. Then, for user α during data submission (Phase 2) D sets $m'^c_0 = (\{\{m^c_b\}_{I^{pub}_N : I^{pub}_1}\}_{O^{pub}_N : O^{pub}_{h_1+1}}$, $m'^c_1 = \{\{\alpha\}_{I^{pub}_N : I^{pub}_1}\}_{O^{pub}_N : O^{pub}_{h_1+1}})$, sends them to C_D, and receives in return the encryption c_{b_D} of $m'^c_{b_D}$. D uses $\{c_{b_D}\}_{O^{pub}_{h_1-1} : O^{pub}_1}$ as ciphertext C_α. During the shuffle decryption (Phase 3), D can decrypt all ciphertexts at h_1 except c_{b_D} by sending them to C_D. D can recognize c_{b_D} and replace it with m'^c_0. In the case that $Z = 1$, D gets a public key from C_D and uses it as the inner public key $I^{pub}_{h_1}$. D has the outer private key of h_1 in this case to do decryptions during Phase 3.

For the rest of the protocol, either D guessed right when setting Z and has the private keys needed by the protocol, or D fails. Either way D can finish the protocol as C^0 would. D then uses game output G as guess $\hat{b}_D$.

If $b_D = 0$, the message m^c_b is contained in the ciphertext submitted by α in Phase 3, and D simulates Game 0 in the rest of the protocol as well. If $b_D = 1$, a dummy ciphertext is submitted by α in Phase 3, and D simulates Game 1. D uses the game output as $\hat{b}_D$, and thus differences in G^0 and G^1 are reflected in $\hat{b}_D$. That is, for any b,

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

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

PROOF. We can reduce distinguishing Games 1 and 2 to winning the IND-CCA2 game in the same way as in Lemma 8.10 by using the same essential distinguisher but replacing α with β and m^c_b with $m^c_{\bar{b}}$. □

We now show that when Game 2 does not fail, the adversary has the same view whether m^c_0 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^c_0 and m^c_1, 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 8.12.

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

PROOF. We can track the internal states and sent messages to observe that nearly all internal variables are independent of b, and, more importantly, that whenever the challenger fails, sends a message, or creates an output the action depends on internal variables in a way that is independent of b. This is easy to see in the case that $Z^2 = 1$ because b is simply never used.

The case when $Z^2 = 0$ is less obvious. We note that the failure cases either don't depend on the input messages of α and β or are symmetric between α and β and thus apply regardless of b. Now assume that C'^2 does not fail. α and β actually commit to dummy messages, and the challenge messages don't appear until after anonymization by h_1 (Phase 3). Then they appear together, in a random position, and must be encrypted using the same set of keys or the challenger would fail, given that any observed hash collision automatically causes failure. Therefore, they appear in the same way regardless of b. After this, the only time α and β act differently depending on b is when detecting their inner ciphertexts during verification (Phase 4). However, because C'^2 does not fail, both inner ciphertexts must appear in the final shuffles received by both α and β, and the lack of hash collisions guarantees that the inner ciphertexts encrypt the challenge messages using the same keys. Thus, either both α and β detect their inner ciphertext or neither does, regardless of b. □

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 8.13. $\Delta(G^2) = 0$.

PROOF. The steps of the anonymity game before the challenge run (i.e. Steps 1–3) do not use the challenge bit b. Therefore the input I from C^2 to C'^2 that the challenger uses during the challenge run is independent of b, and so by Lemma 8.12 the failure, message transcript, and outputs of C'^2 are independent of b. This implies that the subsequent steps of the anonymity game (i.e. Steps 5–6) are independent of b, because they only depend on the previous messages among members. The game output G^2 depends on the failure of C^2 and the output $\hat{b}$ of A in Step 6. We have shown that these both are independent of b, and thus G^2 is independent of b. □

THEOREM 8.14. *The* GMP-SHUFFLE *protocol maintains anonymity with k colluding members for any $0 \leq k \leq N - 2$.*

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 8.9, the advantage of A in the anonymity game with GMP-SHUFFLE is negligibly close to $2\Delta(G^0) \leq 2(\epsilon_{01} + \epsilon_{12} + \Delta(G^2))$. ϵ_{01} is negligible by Lemma 8.10, ϵ_{12} is negligible by Lemma 8.11, and $\Delta(G^2)$=0 by Lemma 8.13. Thus the advantage of A in the anonymity game with GMP-SHUFFLE is negligible. □

8.4.2. The GMP-Bulk Protocol. We show that the adversary's advantage in winning the anonymity game with GMP-BULK is negligible by proving that the game's advantage changes negligibly between neighboring games and is zero in the final game. 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.

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.

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 (Phase 3) of the challenge run in the bulk anonymity game. C^0 fails if his guess proves to be wrong or if a hash collision is observed during the shuffle. Otherwise he behaves the same as the anonymity-game challenger. 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 8.4.1), 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 negligibly close to $1/2$ the advantage of A in the anonymity game:

LEMMA 8.15. *$\Delta(G^0)$ is negligibly close to $(1/2)\left|Pr\left[A^{C(0)}=1\right]-Pr\left[A^{C(1)}=1\right]\right|$.*

PROOF. This lemma holds for almost exactly the same reason as Lemma 8.9. The game between A and C^0 executes the same as that between A and C except for failure upon an observed hash collision and that, for every other execution of the latter, the corresponding execution of the former fails for one of $Z_1^0=0$ and $Z_1^0=1$. Collisions occur with negligible probability due to hash collision resistance. Z_1^0 is chosen independently of the protocol, and thus C^0 fails otherwise with probability $1/2$. G^0 is set uniformly at random when C^0 fails. This implies the lemma. □

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

PROOF. To prove that the output advantage changes negligibly between Game 0 and Game 1, we can adapt the proof of Lemma 8.10. That proof constructs a distinguisher D that turns a non-negligible change in the output of the anonymity game to a non-negligible advantage in the IND-CCA2 game, contradicting the IND-CCA2 assumption. The change needed to that proof is for the distinguisher to play the anonymity game with GMP-BULK instead of GMP-SHUFFLE and for the challenge shuffle steps to instead be executed during the descriptor shuffle (Phase 3) of the bulk challenge run. Note that the input of member $h \in H$ used by D thus becomes the descriptor d_h.

We adapt the distinguisher construction and subsequent arguments of Lemma 8.11 in the same way to show that the game output's advantage changes negligibly from Game 1 to Game 2. □

Game 3 is created by applying the first game transformation of the shuffle proof to the blame shuffle in Game 2. Thus, as in the shuffle proof, the game advantage decreases by a factor negligibly close to $1/2$:

LEMMA 8.17. *$\Delta(G^3)$ is negligibly close to $\frac{1}{2}\Delta(G^2)$.*

PROOF. The proof of this lemma is the same as the proof of Lemma 8.15 except for some small changes. First, the guess Z_2^3 about key release as well as failure upon an observed hash collision are done with respect to the blame shuffle (Phase 7). Second, in this case, we begin with the existing possibility that the challenger fails, specifically during the descriptor shuffle (Phase 3) upon observing a hash collision or making a wrong guess about key release. C^3 fails for this reason exactly when C^2 does, and the probability is negligibly close to $1/2$ in both. The game output is uniformly random in this failure case, and therefore we must observe that the new failure does not affect the game advantage in that case. Thus, with these changes, the earlier proof applies to the current lemma. □

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

PROOF. This lemma can be proven using the arguments of Lemma 8.16 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 8.19. $\left|\Delta(G^6) - \Delta(G^5)\right|$ *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 IND-CCA2 game.

Let Game 5a be the game resulting from just the ciphertext replacement in (*i*). We construct a distinguisher D that converts a change in output between Games 5 and 5a into an advantage in the IND-CCA2 game. Let C_D be the challenger in the IND-CCA2 game and b_D be the challenge bit.

D executes all steps of the anonymity game as C^5 would except during the challenge run. During that run, D obtains a public encryption key from C_D and uses it as the key y_β during key generation (Phase 1a). Then, during descriptor generation (Phase 2), if key verification is successful (Case 1), D creates $S_{\alpha\beta}$ by randomly choosing a new seed $s'_{\alpha\beta}$, submitting $(s_{\alpha\beta}, s'_{\alpha\beta})$ to C_D, receiving c_{b_D} as a response, and setting $S_{\alpha\beta} = c_{b_D}$. During data transmission (Phase 4), if the descriptors were successfully received (Case 2), then 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 D sets $s_{i\beta} = s_{\alpha\beta}$, and otherwise it sets $s_{i\beta} = s$. D otherwise executes the challenge run as C^5 would. Note that D will never be required 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. Finally, D uses the game output 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 C^{5a} if $b_D = 1$. 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.

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 the distribution of $G(b)$ for any b. Because the advantage in the IND-CCA2 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. □

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 8.20. $\left|\Delta(G^7) - \Delta(G^6)\right|$ *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 distribution changes negligibly for each pair in this short sequence by constructing a distinguisher D 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. D executes the anonymity game as C^6 does except for a couple of changes during the challenge round. During the descriptor generation of this round (Phase 2), if β received valid keys (Case 1) then D takes r from C_R and sets $C_{\beta\alpha} = r$. Then during data transmission (Phase 4) of 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. Finally, 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), 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 $s'_{\beta\alpha}$.

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 $\left|Pr[G^6(b) = 1] - Pr[G^{6a}(b) = 1]\right|$ 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 8.12 and 8.13, 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 will not 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 8.4.1 for use in Lemma 8.12. 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 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 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 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 8.21.

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

PROOF. We can pair the executions of C'^7 and A with $b = 0$ and those with $b = 1$ such that each member of a given pair occurs with the same probability and has the same failure event F'^7, message transcript M, and output O. In fact, most protocol operations are completely independent of b, but in GMP-BULK members do create message descriptors, ciphertexts, and accusations differently depending on their input message. Thus we must show that these differences do not affect challenger failure, messages, or outputs in Game 7.

We observe that when key equivocation does not occur in Phase 1, the descriptor for message m_0 is created the same way regardless of the owner. In particular, whichever $h \in \{\alpha, \beta\}$ owns m_0, effectively the ciphertext $C_{h\alpha}$ is chosen at random and $C_{h\beta}$ is the XOR of m_0 and all other ciphertexts. When key equivocation does occur in Phase 1, an honest member h sets $f_h^1 = 1$, and so C'^7 fails when $Z_1^7 = 0$ and uses only dummy messages in the descriptor shuffle when $Z_1^7 = 1$. Either resulting execution is independent of b. The same holds for the m_1 descriptor, and so formation of descriptors d_{m_0} and d_{m_1} by the owners of m_0 and m_1, respectively, has the same probability regardless of b. Thus we can apply Lemma 8.12 to the descriptor shuffle (Phase 3) with challenge messages $m_0^{c_1} = d_{m_0}$ and $m_1^{c_1} = d_{m_1}$ to show that its execution is independent of b.

Furthermore, we can see that during data transmission (Phase 4) α sends the same ciphertext for each seed matching her seed in d_{m_0}, and β sends the same ciphertext for each seed matching hers in d_{m_0}. A similar observation applies for m_1. Thus the generation of ciphertexts by α and β does not depend on b.

Finally, we note that accusations a_0 and a_1 for corrupted slots containing d_{m_0} and d_{m_1}, respectively, are formed the same regardless of b. To see this, observe that accusations are formed depending on the descriptor shuffle output O^1, the received ciphertexts, and the acknowledgements V_k. A successful output O^1 must be the same for α and β by the same reasoning as in the proof of shuffle integrity (cf. Theorem 8.1). Thus, assuming no ciphertext equivocation, whichever $h \in \{\alpha, \beta\}$ owns m_0 will create an accusation a_0 for it in the same way, and the same is true for a_1 and m_1. Thus we can apply Lemma 8.12 to the blame shuffle (Phase 7) with challenge messages $m_0^{c_2} = a_0$ and $m_1^{c_2} = a_1$ to show its execution and output is independent of b. Ciphertext equivocation results in honest member h setting $f_h^2 = 1$, and so either $Z = 0$ and C'^7 will fail during the blame shuffle or $Z = 1$ and dummy messages instead of accusations are sent. Either outcome is independent of b. Thus an execution of the blame shuffle when $b = 0$ has the same probability as the same execution when $b = 1$ but with any accusations for d_{m_0} and d_{m_1} swapped between α and β.

The other parts of the executions do not depend on b. Thus the failures, messages, and outputs of C'^7 are independent of b. □

LEMMA 8.22. $\Delta(G^7) = 0$.

PROOF. The steps of the anonymity game before the challenge run do not use the challenge bit b. Therefore the input I from C^7 to C'^7 during the challenge run is independent of b, and so Lemma 8.21 shows that the failures and message transcript of C'^7 are independent of b. This implies that the subsequent steps of the anonymity game, including the output $\hat{b}$ of A, are independent of b. The game output G^7 depends only on F^7 and $\hat{b}$, and thus it is independent of b as well. □

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

THEOREM 8.23. *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 $\epsilon_{ij} = \left|\Delta(G^j) - \Delta(G^i)\right|$. By Lemmas 8.15 and 8.17, the advantage of A in the anonymity game with GMP-BULK is negligibly close to $2\left(\epsilon_{02} + 2\left(\epsilon_{35} + \epsilon_{56} + \epsilon_{67} + \Delta(G^7)\right)\right)$. By Lemma 8.22 this is $2\epsilon_{02} + 4\epsilon_{35} + 4\epsilon_{56} + 4\epsilon_{67}$. This quantity is negligible by Lemmas 8.16, 8.18, 8.19, and 8.20. □

9. CONCLUSION AND FUTURE WORK

DISSENT is a practical protocol for anonymous and accountable group communication that allows a well-defined group of participants to efficiently exchange variable-length messages, while resisting traffic analysis and disruption attacks effective against mix-networks, DC-nets, and onion routing.

We have presented an improved version of this protocol that fixes several flaws in the original design. In addition, we have expressed the protocol in a modular framework that allows its compo-

nents to be easily reused and analyzed. We have precisely defined its security properties and have given rigorous proofs that the improved protocol satisfies these properties.

Recent additional work on DISSENT has resulted in two new systems. Dissent in Numbers [Wolinsky et al. 2012] accommodates anonymity sets sizes of thousands of nodes by offloading the protocols computational burden to a small set of servers. Verdict [Corrigan-Gibbs et al. 2013] extends this client/server architecture by requiring clients to prove (in zero-knowledge) the well-formedness of messages they submit to the servers, thus preventing the certain disruption attacks which affected prior DC-net-based systems. Both protocols, however, lack rigorous proof of security. Therefore, future work includes a thorough security analysis of the scalable DISSENT and Verdict. Performing a rigorous security analysis of a complex protocol is a time-consuming and error-prone task. Hence, we would like to take advantage of formal verification methods for cryptographic protocols [Meadows 2003], especially for an exhaustive case analysis. We also wish to express and verify DISSENT's security properties in the universally composable (UC) framework [Canetti 2001].

REFERENCES

Masayuki Abe and Hideki Imai. 2003. Flaws in some robust optimistic mix-nets. In *ACISP*.

Masayuki Abe and Hideki Imai. 2006. Flaws in Robust Optimistic Mix-Nets and Stronger Security Notions. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* (2006).

Ben Adida. 2006. *Advances in cryptographic voting systems*. Ph.D. Dissertation. Cambridge, MA, USA.

Jordi Puiggali Allepuz and Sandra Guasch Castello. 2010. Universally verifiable efficient re-encryption mixnet. In *Electronic Voting*.

Michael Backes, Jeremy Clark, Aniket Kate, Milivoj Simeonovski, and Peter Druschel. 2014. BackRef: Accountability in Anonymous Communication Networks. In *ACNS*.

Stephanie Bayer and Jens Groth. 2012. Efficient zero-knowledge argument for correctness of a shuffle. In *EUROCRYPT*.

Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. 1998. Relations among Notions of Security for Public-Key Encryption Schemes. In *CRYPTO*.

Justin Brickell and Vitaly Shmatikov. 2006. Efficient anonymity-preserving data collection. In *SIGKDD*.

R. Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *FOCS*.

Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *OSDI*.

David Chaum. 1981. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM* (1981).

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

Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. 2000. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Workshop on Design Issues in Anonymity and Unobservability*.

Henry Corrigan-Gibbs and Bryan Ford. 2010. Dissent: accountable anonymous group messaging. In *CCS*.

Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. 2013. Proactively Accountable Anonymous Messaging in Verdict. In *USENIX Security Symposium*.

Yvo Desmedt and Kaoru Kurosawa. 2000. How to break a practical MIX and design a new one. In *EUROCRYPT*.

Claudia Diaz and Bart Preneel. 2007. Accountable anonymous communication. In *Security, Privacy, and Trust in Modern Data Management*. Springer, 239–253.

Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004a. Tor: the second-generation onion router. In *USENIX Security Symposium*.

Roger Dingledine, Vitaly Shmatikov, and Paul Syverson. 2004b. Synchronous Batching: From Cascades to Free Routes. In *WPET*.

Roger Dingledine and Paul Syverson. 2002. Reliable MIX Cascade Networks through Reputation. In *FC*.

John R. Douceur. 2002. The Sybil Attack. In *1st International Workshop on Peer-to-Peer Systems*.

Joan Feigenbaum, James A Hendler, Aaron D Jaggard, Daniel J Weitzner, and Rebecca N Wright. 2011. Accountability and deterrence in online life. In *'11 ICWS*.

Jun Furukawa and Kazue Sako. 2001. An Efficient Scheme for Proving a Shuffle. In *CRYPTO*.

Sharad Goel, Mark Robson, Milo Polte, and Emin Gun Sirer. 2003. *Herbivore: A Scalable and Efficient Protocol for Anonymous Communication*. Technical Report 2003-1890. Cornell University.

David Goldschlag, Michael Reed, and Paul Syverson. 1999. Onion Routing for Anonymous and Private Internet Connections. *Commun. ACM* (1999).

Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. 1995. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. (1995).

Philippe Golle and Ari Juels. 2004. Dining Cryptographers Revisited. In *Eurocrypt*.

Philippe Golle, Sheng Zhong, Dan Boneh, Markus Jakobsson, and Ari Juels. 2002. Optimistic Mixing for Exit-Polls. In *ASIACRYPT*.

Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. 2007. PeerReview: Practical Accountability for Distributed Systems. In *SOSP*.

Jan Iwanik, Marek Klonowski, and Miroslaw Kutylowski. 2004. DUO-Onions and Hydra-Onions – failure and adversary resistant onion protocols. In *IFIP CMS*.

Markus Jakobsson. 1998. Flash Mixing. In *EUROCRYPT*.

Markus Jakobsson and Ari Juels. 2001. An Optimally Robust Hybrid Mix Network. In *PODC*.

Shahram Khazaei, Tal Moran, and Douglas Wikström. 2012a. A mix-net from any CCA2 secure cryptosystem. In *ASIACRYPT*.

Shahram Khazaei, Björn Terelius, and Douglas Wikström. 2012b. Cryptanalysis of a universally verifiable efficient re-encryption mixnet. In *International conference on Electronic Voting Technology/Workshop on Trustworthy Elections*.

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

Catherine Meadows. 2003. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *Selected Areas in Communications, IEEE Journal on* 21, 1 (2003), 44–54.

Masashi Mitomo and Kaoru Kurosawa. 2000. Attack for Flash MIX. In *ASIACRYPT*.

C. Andrew Neff. 2001. A verifiable secret shuffle and its application to e-voting. In *CCS*.

C. Andrew Neff. 2003. Verifiable mixing (shuffling) of ElGamal pairs. *VHTi Technical Document, VoteHere, Inc* (2003).

Omkant Pandey, Rafael Pass, and Vinod Vaikuntanathan. 2008. Adaptive One-Way Functions and Applications. In *CRYPTO*.

Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. 1994. Efficient anonymous channel and all/nothing election scheme. In *EUROCRYPT*.

G. Perng, M.K. Reiter, and Chenxi Wang. 2006. M2: Multicasting Mixes for Efficient and Anonymous Communication. In *26th ICDCS*.

Birgit Pfitzmann. 1994. Breaking an Efficient Anonymous Channel. In *EUROCRYPT*.

Birgit Pfitzmann and Andreas Pfizmann. 1990. How to break the direct RSA-implementation of mixes. In *EUROCRYPT*.

Michael K. Reiter and Aviel D. Rubin. 1999. Anonymous Web transactions with Crowds. *Commun. ACM* (1999).

Phillip Rogaway and Thomas Shrimpton. 2004. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance.. In *FSE*.

Andrei Serjantov, Roger Dingledine, and Paul Syverson. 2003. From a Trickle to a Flood: Active Attacks on Several Mix Types. *Information Hiding* (2003).

Victor Shoup. 2004. Sequences of games: a tool for taming complexity in security proofs. In *IACR Cryptology ePrint Archive*.

Emin Gün Sirer, Sharad Goel, Mark Robson, and Dŏgan Engin. 2004. Eluding Carnivores: File Sharing with Strong Anonymity. In *11th ACM SIGOPS European Workshop*.

Douglas R. Stinson. 2005. *Cryptography: Theory and Practice, Third Edition*. Chapman & Hall/CRC.

Brad Stone and Matt Richtel. 2007. The Hand That Controls the Sock Puppet Could Get Slapped. *New York Times* (2007).

Ewa Syta, Aaron Johnson, Henry Corrigan-Gibbs, Shu-Chun Weng, David Wolinsky, and Bryan Ford. 2013. *Security Analysis of Accountable Anonymous Group Communication in Dissent*. Technical Report TR1472. Yale University.

Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. 2000. Towards an Analysis of Onion Routing Security. In *Design Issues in Anonymity and Unobservability*.

Luis von Ahn, Andrew Bortz, and Nicholas J. Hopper. 2003. k-anonymous message transmission. In *CCS*.

Luis von Ahn, Andrew Bortz, Nicholas J Hopper, and Kevin ONeill. 2006. Selectively traceable anonymity. In *PETS*.

Michael Waidner and Birgit Pfitzmann. 1989. The Dining Cryptographers in the Disco: Unconditional Sender and Recipient Untraceability with Computationally Secure Serviceability. In *Eurocrypt*.

Douglas Wikström. 2003. Five Practical Attacks for "Optimistic Mixing for Exit-Polls". In *Selected Areas in Cryptography*.

Douglas Wikström. 2004. A Universally Composable Mix-Net. In *TCC*.

David Isaac Wolinsky, Henry Corrigan-Gibbs, Aaron Johnson, and Bryan Ford. 2012. Dissent in Numbers: Making Strong Anonymity Scale. In *OSDI*.

Yale Law Journal. 1961. The Constitutional Right to Anonymity: Free Speech, Disclosure and the Devil. *Yale Law Journal* (1961).

Received January 2013; revised January 2013; accepted January 2013