Selectively Traceable Anonymity

Luis von Ahn[1], Andrew Bortz[2], Nicholas J. Hopper[3], and Kevin O'Neill[4]

[1] Computer Science Department, Carnegie Mellon University, Pittsburgh, PA USA
[2] Computer Science Department, Stanford University, Palo Alto, CA USA
[3] Digital Technology Center & CSE Department, University of Minnesota, Minneapolis, MN USA
[4] Computer Science Department, Cornell University, Ithaca, NY USA

Abstract. Anonymous communication can, by its very nature, facilitate socially unacceptable behavior; such abuse of anonymity is a serious impediment to its widespread deployment. This paper studies two notions related to the prevention of abuse. The first is *selective traceability*, the property that a message's sender can be traced with the help of an explicitly stated set of parties. The second is *noncoercibility*, the property that no party can convince an adversary (using technical means) that he was not the sender of a message. We show that, in principal, almost any anonymity scheme can be made selectively traceable, and that a particular anonymity scheme can be modified to be noncoercible.

1 Introduction

Anonymous communication has several important potential applications, including anonymous email for "whistle-blowing," anonymous web browsing to access useful but possibly embarrassing or incriminating information (e.g., "how to deal with a drug addiction"), and mechanisms to ensure individual privacy in electronic transactions. At the same time, there are obvious ways in which anonymity protocols could be used for antisocial or criminal purposes such as slander, threats, and transfer of illegal content. In some cases, especially when the anonymity guarantees are strong, the negative consequences of allowing users to communicate anonymously can outweigh the benefits. This is a major stumbling block for the widespread adoption of anonymizing systems.

Systems for anonymous communication have generally tried to provide the strongest possible guarantees while providing some reasonable level of efficiency and ease-of-use, but, surprisingly, have usually not addressed "revoking" the anonymity of a message in a formal manner.[5] In this paper we argue that it would be useful to have anonymity protocols that *explicitly* allow the tracing of a message's sender whenever a set of fair and sensible conditions is met. To this effect, we define *selectively traceable anonymous communication*, which allows tracing a message when a *tracing policy* is satisfied, such as a fixed fraction of the participants voting to trace the message.

Another reason for examining tracing in anonymity protocols is that some existing anonymity protocols already allow a form of tracing by allowing participants to *prove* that they did not send some particular message. If a protocol has this property, we call it *coercible*, because participants can be coerced into proving that they did not send a particular message. Coercibility is related to tracing in that a coercible protocol allows gradual and uncoordinated tracing: every participant except the sender can show that they did not send the message. If the anonymity set of a message is small, this can be easier than tracing through other means. The notion of uncoercible anonymity is similar to the notions of coercibility in election protocols [JJ02], *deniability* in encryption [CDNO97], and *adaptive security* in multiparty computation [CFGN96].

We present two definitions of traceable anonymity. In one, which we refer to as *weak traceable anonymity*, a message should be traced whenever the tracing policy is satisfied; in the other, *strong traceable anonymity*, nothing about the sender of a message should be learned unless the tracing policy is satisfied. To clarify the distinction between these definitions, we mention that a weak traceable protocol can be coercible: the message can be traced when the tracing policy is satisfied, but something about the identity of the sender

[5] One exception is the mechanisms in various anonymous cash and election protocols that allow revoking the anonymity of a user who double-spends or double-votes.

can be revealed even if the tracing policy is not satisfied if any participants prove that they did not send the message. A strong traceable protocol does not allow such coercion.

In this paper, we present definitions and several technical results relating to selectively traceable anonymous communication. Our technical results include:

A generic transformation that adds selective traceability. We show that a large class of systems for anonymous communication can, in principle, be transformed into systems with selectively traceable anonymity, using a construction that first appears in [KTY04]: append an anonymous "group signature" to every message sent on an anonymous channel and *require the receivers to drop all messages that are not signed.* We note that this transformation suffers from an incentive problem: receivers have no incentive to drop unsigned messages, and thus senders have no incentive to sign messsages. We show that, in principle, almost any anonymity scheme can be transformed to avoid this problem without sacrificing anonymity.

Two efficient transformations from specific DC-Net-like protocols. We show efficient transformations from two specific DC-Net-based protocols: [ABH03,GJ04]. The transformations do not affect the efficiency of the underlying non-traceable protocols and yield security against malicious adversaries.

Coercibility results. We discuss the notion of coercibility in anonymous communication, and show how the DC-Net-based protocols in [ABH03,GJ04] allow coercion. We show a simple modification to the [ABH03] protocol that gives noncoercibility. We also show that our generic transformations do not alter the coercibility (or noncoercibility) of the underlying protocols. These results show that, in principle, strong traceable anonymity can be acheived.

2 Threshold Cryptography and Group Signatures

We use two main building blocks for the technical results that follow: threshold El Gamal decryption and group signatures. The first technique generalizes El Gamal encryption so that private keys are distributed among a number of principals; the second provides a way for a principal to sign a message anonymously in such a way that the signer's anonymity can be revoked by the group manager.

Distributed El Gamal Decryption [Ped91a]. We will use a public-key encryption system to encrypt information that identifies the sender of a message. To do so in a way that respects a particular tracing policy, however, we want decryption to occur only when all the voters in some tracing set T agree to take part. In other words, we require a cryptosystem with the following features:

1. There is an "aggregate" public key y that can be used to encrypt messages, as with regular public-key cryptosystems.
2. Each voter v_i has a secret private key x_i that can be used to "partially" decrypt a ciphertext C, and decryption is computationally hard unless all the voters in some tracing set T take part in the decryption.

Group Signatures. Group signature schemes [CvH91] provide a way for members of a group to sign messages anonymously. That is, they allow a member of a group to digitally sign a document in such a way that it may be verified that the document was signed by a group member, but not which particular group member signed it unless a designated group manager "opens" the signature.

Definition 1. *(From [ACJT00]): A group signature scheme is a digital signature scheme comprised of the following five procedures:*

- `SETUP` *outputs the initial group public key GPK (including all system parameters) and the secret key for the group manager.*
- `JOIN` *allows a new user to join the group. The user's output is a membership certificate and a membership secret.*

– $\texttt{SIGN}(m)$, *given* GPK, *a membership certificate and secret, and a message* m, *outputs a group signature on* m.
– `VERIFY` *establishes the validity of an alleged group signature* σ *on message* m *with respect to* GPK.
– `OPEN` *given a message* m *with valid group signature* σ, *the key* GPK *and the group manager's secret key, determines the identity of the signer.*

Group signature schemes must satisfy a variety of properties. Signatures produced using `SIGN` must be accepted using `VERIFY`, for example, and the actual signer of a message should remain anonymous until the signature is opened by the group manager. For more details, see [ACJT00].

Many group signature schemes (e.g., [ACJT00,KTY04]) implement `OPEN` as an instance of El Gamal decryption. In these schemes the group manager can be distributed so that each instance of `OPEN` operates according to a threshold scheme.

3 Selective Traceability

Tracing, like anonymity, may be abused. Accordingly, we want to avoid any requirements that tracing information be logged or enforaced by any single, central authority, since in many cases the primary reason for having an anonymity protocol is to provide protection against central authorities. To describe a general framework for traceable schemes, it will therefore be important to specify *who* is able to trace. The setting we consider is as follows: there is a finite set G of *users* who may be able to send or receive messages anonymously, and there is a finite set V of *voters* who are authorized to trace a message. There is also a set $\mathcal{V} \subseteq 2^V$, the *tracing policy*, such that an act of tracing only occurs when all the members of a *tracing set* $T \in \mathcal{V}$ agree to it. (We assume that $\mathcal{V}$ is *monotone*, so that if $T \in \mathcal{V}$ and $T \subseteq T'$, then $T' \in \mathcal{V}$. It therefore suffices to consider only the minimal sets in $\mathcal{V}$.) We call $(G, V, \mathcal{V})$ a *tracing scheme*. Some examples of tracing policies include:

1. The trivial tracing policy, in which explicit tracing by voters is not allowed, can be represented with $\mathcal{V} = \emptyset$. (For many protocols, a sufficiently large subset of the users of a system can cooperate to trace messages; but this is an implicit process, rather than one enforced by the protocol.)
2. Given V and an integer $1 \leq t \leq |V|$, let $\mathcal{V}(t) = \{R \subseteq V \mid |R| = t\}$. $\mathcal{V}(t)$ is a *threshold tracing policy*, with parameter t. Tracing occurs only when at least t members of V agree that tracing should occur.
3. Let V be the set of n members of a legislative body (e.g., the US Senate's 100 members or the UK House of Commons' 646 members); then $\mathcal{V}(\lfloor n/2 \rfloor + 1)$ is the policy that says a legislative act is required to trace a message.

We note that there is a close relationship between the tracing scheme $\mathcal{V}$ of a selectively traceable anonymity protocol and the "trust model" of any anonymity protocol. In particular, when a static set of nodes must be trusted not to reveal the sender of all messages, it is clear that the tracing policy must include this subset as an element. On the other hand, a tracing policy explicitly specifies sets of voters (not necessarily participants) who may trace a message regardless of its origin or destination; a participant must therefore trust these sets of voters. In the case of a tracing policy, however, these sets are always static, and always have the power to trace a message; in many existing anonymity protocols, the set of nodes that can trace any particular message varies by message. Thus "trust models" are mostly a side-effect of the protocols employed by some anonymous communication schemes, whereas tracing policies are conscious decisions to allow tracing the anonymity of a message.

3.1 Generic transformations

In this section we present a method to convert a generic anonymous communication protocol to a new protocol that permits selective tracing. We assume that there is an independent set V of voters and a threshold tracing policy $\mathcal{V} \subseteq 2^V$. (We remark that any monotone tracing policy may be implemented using our method, though in the worst case the length of the shares may be exponential in the size of the voting set. Here we focus only on the threshold case.) We do not assume anything about the voters except that they can be trusted with a secret share of the El Gamal private key that will be used for decryption. The voters may be principals in the original anonymous communication scheme, but this isn't a necessary requirement. For this work, we make the simplifying assumption that a group manager enforces some binding between a user's identity in the JOIN protocol and that user's physical identity.

Let $\mathcal{M}$ be the set of possible *anonymous messages*, which are generated by one party to be processed for anonymous delivery to another party, and let $\mathcal{PM}$ be the set of *protocol messages*, which are exchanged by parties during the execution of the protocol. Our generic transformation applies to protocols that include a finite number of parties $\{P_1, \ldots, P_n\}$ and include the primitive operations SEND, PROCESS, and RECOVER, which we now describe. (These operations use a set of public parameters selected by an initial setup stage, and each player P_i may use his secret parameters S_i in any stage):

- SEND: a procedure executed by P_i that takes as input an anonymous message $m \in \mathcal{M}$ and a recipient P_j, and outputs a list $\boldsymbol{c}$ of pairs $(c_{i,j}, P_j)$ where $c_{i,j}$ is a protocol message to be sent to P_j.
- PROCESS: a procedure executed by P_j that takes as input a list of pairs $(c_{i,j}, P_i)$, where the $c_{i,j}$ are protocol messages received from P_i, and outputs a new list $\boldsymbol{c}'$ of pairs $(c'_{j,k}, P'_k)$ where $c'_{j,k}$ is a protocol message to be sent to P'_k. (We remark that there may be several rounds of PROCESS operations during a single execution of the protocol.)
- RECOVER: a procedure executed by P_j that takes as input a list $\boldsymbol{c}$ (or multiple vectors) of pairs $(c_{i,j}, P_i)$, where the $c_{i,j}$ are protocol messages received from P_i, and outputs a list of pairs (m_k, P'_k) where each m_k is an anonymous message to send to P'_k.

All well-known anonymity protocols in the security literature implement variants of these protocols. With mixes and onion-routing protocols, for example, a PROCESS step takes a batch of protocol messages and shuffles and forwards them along to other parties, possibly after performing some operation on the messages such as encryption and/or decryption.

Transformation 1: The first transformation we consider (already mentioned in [KTY04]) affects the SEND and RECOVER steps of a given protocol. In the new protocol the sender P_i must sign the message $m \in \mathcal{M}$ to get a group signature σ, and the resulting message $m' = (m, \sigma)$ is the one that must be processed by the SEND operation. For any party P_j executing a RECOVER operation to recover a message m, P_j must ensure that m has been signed using a group signature and must discard the message if it has not been signed.

If a receiver P_k presents an anonymous message to the voting group V for tracing, a tracing set $T \in \mathcal{V}$ may open the signature to reveal the sender.

A significant problem with Transformation 1 is that nothing stops the party P_j executing RECOVER from reading a recovered unsigned message, or sending it on to its intended recipient — regardless of whether P_j is simply curious or is attempting to subvert the tracing scheme. As soon as unsigned messages are read instead of dropped, senders have no incentive to sign messages that they may later be blamed for, and the system degrades into a non-traceable protocol. Of course one could appoint a trusted "auditor" to check that all messages are signed before delivery but this would both have the effect of severely degrading the anonymity of the system (the auditor sees ALL messages delivered!) and would create a single point of failure for the anonymity protocol; we seek a solution that violates anonymity for traceability *only* to the extent that it enforces the tracing policy.

Transformation 2: In most anonymity protocols, the PROCESS step involves protocol messages from which the original anonymous message m cannot be efficiently recovered by the party executing the step. The message may be encrypted, for example, or split into shares using some secret-sharing scheme. (One exception to this is the Crowds framework [RR98], where messages may be sent in plaintext. Protocol participants essentially flip a coin to decide whether to execute a PROCESS or a RECOVER operation, and they can see the anonymous messages at every step.) The transformation we outline below may be applied whenever it is impossible or computationally infeasible to recover m from the PROCESS step.

Our solution to the game-theoretic problem of Transformation 1 is to require that an agent P_j executing a PROCESS step must check that the protocol messages $c_{1,j}, \ldots, c_{n,j}$ are all generated from underlying anonymous messages that have been signed using the group signature scheme. To do this without revealing anything about the underlying message, we use noninteractive zero-knowledge (NIZK) proofs [BDMP91]; briefly, these are objects that prove the truth of a statement without revealing anything about the proof. Essentially, we define *valid* protocol messages to be those that are the output of `SEND` on a signed-message, or `PROCESS` on a set of valid messages; then modify the `SEND` procedure to output of NIZK of validity, and modify the `PROCESS` procedure to verify the validity of all inputs and output a NIZK of the validity of its outputs. Full details appear in Appendix B.

Efficiency. We stress that the point of this general scheme is not to suggest a protocol that should be used in practice, but to show that *in principle*, any anonymity scheme can provide selective traceability. Indeed, the most efficient general constructions of NIZKs [GOS05] have length roughly $6000T$ bits, where T is the number of bit operations required to verify that $x \in L$ given witness w. Since in the previous transformation, this involves (at minimum) verifying a group signature or checking a NIZK, and the most efficient such signatures require roughly $T = 10^6$ bit operations per verification, the generic transformation cannot be considered practical.

3.2 More Efficient Transformations

In this section, we will demonstrate simple modifications to allow selective tracing of two DC-Net-based protocols: k-AMT [ABH03] and a protocol due to Golle and Juels [GJ04] which we refer to as GJ. Both protocols make slight alterations to the basic DC-Net protocol [Cha88] to implement a shared channel; these modified protocols are then run in several parallel copies, and cryptographic mechanisms are employed to prove that each participant broadcasts on at most one channel, ensuring fair access to the medium. Our approach considers the messages sent on each channel orthogonally and allows determining who has broadcast on a single channel, so for ease of exposition we will describe the protocols here only in terms of a single shared channel.

k-AMT. The k-AMT protocol implements a shared channel as a secure multiparty sum computation, using Pedersen commitments[6] to ensure correctness. Here we assume that player P_i wants to send message X_i. The basic protocol has four phases:

1. **Commitment Phase:**
   – P_i splits $X_i \in \mathbb{Z}_q$ into n random shares $s_{i,1}, ..., s_{i,n}$, and chooses $r_{i,j} \leftarrow \mathbb{Z}_q$
   – P_i computes and broadcasts commitments $\{C_{i,j} = C_{r_{i,j}}(s_{i,j}) : 1 \leq j \leq n\}$.
2. **Sharing Phase:**
   – For each $j \neq i$, $P_i \longrightarrow P_j \ : \ (r_{i,j}, s_{i,j})$.
   – P_j checks that $C_{r_{i,j}}(s_{i,j}) = C_{i,j}$
3. **Broadcast Phase:**
   – P_i computes and broadcasts $R_i = \sum_j r_{j,i} \bmod q$ and $S_i = \sum_j s_{j,i} \bmod q$.
   – All players check that $C_{R_i}(S_i) = \prod_j C_{j,i} \bmod p$
4. **Result:** Each player computes $X = \sum_i S_i \bmod q$ and $R = \sum_i R_i \bmod q$; if $C_R(X) = \prod_{i,j} C_{i,j} \bmod p$, the player outputs the anonymous message X.

As was previously mentioned, k-AMT actually runs several parallel copies of this protocol and includes procedures for proving that a party has transmitted on at most one parallel channel or "slot." Here we will describe how to augment the basic protocol so that it is selectively traceable. It should be clear that these modifications are orthogonal to those additional procedures.

The new protocol exploits the relationship between El Gamal encryption and Pedersen commitments to allow the voters to "decrypt" the commitments generated in Phase 1 (when the tracing policy is satisfied). Here we describe the necessary modifications.

[6] If p, q are primes such that $p = 2q + 1$, and $g, h \in \mathbb{Z}_p^*$ both have order q, a Pedersen commitment to the value $x \in \mathbb{Z}_q$ is generated by randomly choosing $r \in \mathbb{Z}_q$ and computing $C_r(x) = g^x h^r$.

1. **Initialization:** As a group, choose securely an El Gamal key pair (G, x, y) where $y = G^x$, such that the private key x is shared by threshold secret sharing according to the desired tracing policy, as in Section 2.
2. **Commitment Phase:** In addition to the $\{C_{i,j} : j \in [M]\}$ commitments broadcast by party P_i, we will have P_i broadcast a certificate that can be proven correct for a given set of commitments, but can only be opened by the owner of the private key of the El Gamal encryption scheme above.
   Assuming that a round of k-AMT is correctly computed, we are guaranteed that $S_i = \prod_j C_{i,j} = g^{X_i} h^{R_i}$, where $R_i = \sum_j r_{i,j}$. Let $a_i = G^{R_i}$ and $b_i = g^{-X_i} y^{R_i}$. Together, a_i and b_i form an El Gamal encryption of g^{-X_i} with the public key y.
   Finally, we compute σ_i to be an efficient noninteractive proof of knowledge that the discrete log of a_i with respect to base G is the same as the discrete log of $S_i b_i$ with respect to base hy. The certificate broadcast in addition to the commitments is just (a_i, b_i, σ_i).

Now, to trace a message: identify the slot that it was transmitted on, obtain a number of parties as required by the tracing policy, and securely compute the decryption M of each party's certificate for that slot. For all participants who sent nothing on the channel we have $X_i = 0$, and thus $M = g^{-X_i} = 1$. All other participants transmitted something on the channel, and in particular if only one participant i sent a message we have $X = X_i$, and thus $M \cdot g^X = 1$.

To compute σ_i, we want to show that $\log_G a_i = \log_{hy} S_i b_i$. In general, to prove that $\log_g y = \log_h z$ when $\log_g h$ is unknown and hard to compute, it suffices to prove knowledge of $\log_{g/h}(y/z)$. (If there exists a such that $y = g^a$ and $z = h^a$, then because $g^a z = h^a y$ we have $\log_{g/h}(y/z) = a$. If $y = g^a$ and $z = h^b$, with $a \neq b$, then knowledge of $\log_{g/h}(y/z)$ can easily be used to compute $\log_g h$.) Therefore, σ_i is a noninteractive proof of knowledge of $\log_{G/hy}(a_i/S_i b_i)$, and can be computed efficiently using standard techniques.[7] Note that this modification doesn't affect the asymptotic efficiency of the underlying protocol.

We prove in Appendix C that under the Decisional Diffie-Hellman assumption, the protocol remains secure against computationally bounded adversaries that have not corrupted a tracing set.

The GJ DC-Net Protocol. The GJ DC-Net protocol takes advantage of bilinear maps to perform many Diffie-Hellman key exchanges noninteractively, thus achieving a single-round (noninteractive) DC-net protocol. The protocol works over groups $\mathbb{G}_1, \mathbb{G}_2$ of prime order q, and with an admissible bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. (A map is bilinear if $\hat{e}(aP, bP) = \hat{e}(P, P)^{ab}$.) We denote the group operation in $\mathbb{G}_1$ using additive notation, and the group operation in $\mathbb{G}_2$ using multiplicative notation, as is common when dealing with admissible bilinear maps. ($\mathbb{G}_1$ is typically an elliptic curve group.) We let $P \in \mathbb{G}_1$ be a public parameter and assume all parties know a map $H : \{0,1\}^* \to \mathbb{G}_1$, which we will model as a random oracle. As previously mentioned, the GJ protocol is actually comprised of several parallel executions of a simple shared channel along with some auxiliary information that proves a player has transmitted on at most one channel; for simplicity, and because our modifications are orthogonal, we describe only the single channel and omit the auxiliary information. For a description of the full protocol, see [GJ04].

1. **Setup Phase** Every player P_i picks private key $x_i \in \mathbb{Z}_q$ and publishes $y_i = x_i P$ as his public key.
2. **Pad Construction** Let s be some unique identifier of a particular execution of the shared channel. (For example, a running count appended to the list of users). All players compute the element $Q_s \in \mathbb{G}_1$ as $Q_s = H(s)$. Then each pair of players (noninteractively) computes a shared Diffie-Hellman key
   $$k_{i,j}(s) = \hat{e}(y_j, x_i Q_s) = \hat{e}(P, Q_s)^{x_i x_j} = \hat{e}(y_i, x_j Q_s) = k_{j,i}(s) \ .$$
   Each player i computes his "pad" $p_i(s) = \prod_j k_{i,j}(s)^{\delta_{i,j}}$, where $\delta_{i,j} = -1$ if $i < j$ and 1 otherwise.

[7] In the random oracle model, a proof of knowledge of $\alpha = \log_\gamma \beta$ has the form $(\zeta = \gamma^\rho, \lambda = \alpha H(\zeta) + \rho)$, where $\rho \in_R \mathbb{Z}_q$ and $H : \mathbb{Z}_p^* \to \mathbb{Z}_q$ is a random oracle; the proof is accepted if $\gamma^\lambda = \beta^{H(\zeta)} \zeta$; interactive versions of this protocol first appear in [CEGP86].

3. **Transmission** In session s, we let the intended message of P_i be the element $m_i(s) \in \mathbb{G}_2$, where $m_i(s)$ is the identity element $1 \in \mathbb{G}_2$ if P_i has no message to send. To transmit, each player P_i publishes value $W_i(s) = m_i(s)p_i(s)$.
4. **Message Extraction** The final message is extracted by computing

$$m(s) = \prod_i W_i(s) = \prod_i m_i(s) \prod_j k_{i,j}(s)^{\delta_{i,j}} = \prod_i m_i(s) \ ,$$

since $k_{i,j}(s)^{\delta_{i,j}} = k_{j,i}(s)^{-\delta_{j,i}}$. Thus if exactly one $m_i(s) \neq 1$, then we have $m(s) = m_i(s)$.

To support selective tracing, the only modification to the previous procedures is in the setup phase: after generating key pair (x_i, y_i) and publishing y_i, player P_i will share his private key x_i among the voters in a similar fashion to that described in section 2. Then to trace the message $m(s)$, the voters will compute the pads $p_i(s)$ for each i using their shares. If the published value $W_i(s) = m(s)p_i(s)$, then P_i is the sender. We formally describe the new procedures in Appendix D.
We note that in the full GJ protocol [GJ04] shares of the private keys x_i are distributed amongst the *players* to allow any two-thirds of them to reconstruct the pads of players who do not participate in any given session. So, even though this is done for different reasons, the GJ protocol silently implements a threshold tracing scheme, with $V = \{P_1, \ldots, P_n\}$ and $\mathcal{V} = \mathcal{V}(\frac{2n}{3})$.

4 Coercibility in Anonymous Protocols

Informally, we say that an anonymity protocol is *coercible* if every player who did not send a message can produce a proof that this is the case. More formally, consider a "proof protocol" $\mathbb{P}$ between a player P_i and a verifier V, where the difference in the probabilities that V "accepts the proof" when P_i sent the message and P_i did not send the message is at least some value ρ_P. We call a protocol ρ-coercible if, over all $\mathbb{P}$, $\rho = \max(\rho_P)$. In other words, ρ measures the confidence of the best proof procedure. If a protocol is 1-coercible, only the legitimate sender of a message cannot exculpate himself (but everybody else can); if a protocol is 0-coercible the verifier should not believe any proofs. If ϵ is negligible, we say that a protocol that is $(1-\epsilon)$-coercible is *strongly coercible*, and that a protocol that is at most ϵ-coercible is *noncoercible*. If a protocol is ρ-coercible for some constant ρ, we say that it is *plausibly noncoercible*.

In this section we assume that all the players in the protocol Π are plausible senders of any message m. Assuming that all the players belong to the same "anonymity set" (i.e., the set of players who could have sent a particular message) lets us ignore "proofs of innocence" that can arise simply because two players belong to different anonymity sets.

Formally, for an anonymous communications protocol Π we define coercibility as follows:

- A *proof procedure* $\mathbb{P}$ is a pair $(\mathcal{P}, \mathcal{V})$ of programs such that $\mathcal{V}$ outputs either `acc` (for *accept*) or `rej` (for *reject*). (Intuitively, $\mathcal{P}$ can be thought of as a program that is run by some player P_i.)
- After the public parameters of Π are chosen, $\mathcal{V}$ is allowed to choose a message m as a function of the parameters. This is the message that, if sent during an execution of the protocol, $\mathcal{V}$ will ask players in Π to prove they have not sent.
- Let $view_X(P_j : m)$ denote the *view* of party X in the anonymity protocol Π when P_j sends message m and m is delivered. The view includes X's inputs (including random tape) and any protocol messages sent and received during the execution of Π.
- Let A denote an (arbitrary) adversary who cannot compromise the anonymity guarantee of Π. For any player X, denote by $view_A(X : m)$ the views of all parties corrupted by A as well as all protocol messages from Π that A observes. Essentially, A will serve as $\mathcal{V}$'s agent in Π: we allow the verifier access to A's view of Π to help in deciding whether to accept $\mathcal{P}$'s proof that P_i didn't send m. Denote by $\mathbb{P}_i(X : m)$ the output of $\mathcal{V}$ (on input m and $view_A(X : m)$) when interacting with $\mathcal{P}$ (on input m and $view_{P_i}(X : m)$).

– We say that Π is *ρ-coercible* if there is a proof procedure $\mathbb{P}$, an adversary A, and players P_i and P_j such that
$$|\Pr[\mathbb{P}_i(P_j : m) = \texttt{acc}\ ] - \Pr[\mathbb{P}_i(P_i : m) = \texttt{acc}\ ]| \geq \rho\ ,$$
regardless of P_i's actions in the second case.

Notice that this definition is weak in the sense that the verifier is allowed to choose the message. In other words, the protocol is coercible if there *exists* a message and adversary such that some player can prove that she did not send the message. (This makes noncoercibility a stronger definition, because it rules out any convincing proofs of innocence.) As we will demonstrate, the coercibility of several protocols from the literature is much stronger — and therefore more problematic — because it allows any player to prove she is not the sender of any message she did not send.

Coercibility for group signature schemes can be defined analogously. We remark that noncoercibility of group signatures satisfying the security definitions of [BMW03] is implied by the "full anonymity" condition.

Recently, Danezis and Clulow [DC05] have introduced the notion of *compulsion-resistant* anonymity protocols. In their setting, an adversary may *compel* certain noncooperative nodes to reveal their secrets (via, for example decrypting ciphertexts or revealing logs or secret keys) in an attempt to trace a message back to its sender. Noncoercibility and compulsion-resistance are related in that both concern the ability of an adversary to trace a message after it has been sent. Our notion is different from compulsion-resistance in several ways. First, a coercive adversary is given a complete transcript of a protocol execution, whereas the perhaps more realistic (but weaker) "compulsive" adversary has only an anonymous reply block. Second, our constructions consider mainly DC-Net based protocols whereas [DC05] is concerned mainly with mix-based protocols. Finally, the goals of noncoercibility and compulsion-resistance differ somewhat: a noncoercible protocol aims to make compulsory revelation of secrets useless because no such revelation will convincingly exonerate a nonsender, whereas a compulsion-resistant protocol aims to make such compulsory tracing prohibitively expensive.

4.1 Coercibility in various anonymity protocols

In the simplest formulation of Chaum's mix-net protocol [Cha81], each party sends a message to the mix, who decrypts and shuffles the messages before forwarding them to the recipients. This protocol is clearly coercible against a global passive adversary: if P_i sent ciphertext c_i to the mix, and c_i does not decrypt to m, he can open c_i to plaintext $p_i \neq m$ to the verifier. The true sender, on the other hand, cannot. It is similarly clear that, in the worst case, any forwarding-based scheme which relies on static public or shared keys allows similar acts of exculpation to a global passive adversary: by decrypting all received ciphertexts and opening all sent ciphertexts, P_i can prove that he was not the originator of any message he did not send. Clearly some players will be reluctant to sacrifice their anonymity entirely in order to give such proofs. It is conceivable, however, that the consequences of non-exculpation could be serious enough that such a privacy loss would be acceptable to P_i. In this work we leave open the interesting question whether such forwarding-based protocols remain coercible in settings that employ forward-security or against different adversarial models.

In Section 3.2 we focused on selective tracing in protocols based on DC-Nets, in part because of the reliance of those protocols on cryptographic techniques that are amenable to tracing. For similar reasons, both of those protocols are coercible. Here we show how participants in those protocols are able to prove easily that they did not send particular messages that were sent by other participants during an execution of the protocol.

In a GJ DC-Net, player P_i can prove that he didn't send a message during session s by publishing the quantity $z_i(s) = x_i Q_s$. (Note that $z_i(s)$ doesn't reveal anything about P_i's private key x_i.) From $z_i(s)$, P_i's pad $p_i(s)$ can be publicly computed as $p_i(s) = \prod_j k_{i,j}(s)^{\delta_{i,j}} = \prod_j \hat{e}(y_j, z_i(s))^{\delta_{i,j}}$. $W_i(s)$ — the value publically declared by P_i — will be the same as $p_i(s)$ if and only if P_i did not send the message.

In k-AMT, player P_i broadcasts commitments $C_{i,j} = C_{r_{i,j}}(s_{i,j})$ of the random shares $s_{i,1}, ..., s_{i,n}$ broadcast to the other players when P_i sends message X_i. If P_i wants to prove that she did not send a message, i.e., that $X_i = 0$, she needs only to open the commitments $C_{i,j}$ by announcing the shares $s_{i,j}$ and the random values $r_{i,j}$. (Opening a commitment $C_{i,j}$ to some value $s'_{i,j} \neq s_{i,j}$ is as computationally hard as computing $\log_g(h)$, where g and h are the generators used in the commitment scheme.) Other users can easily check that $\sum_j s_{i,j} = 0$, thus proving that P_i did not send the message in question.

We note, however, that k-AMT can be modified to be noncoercible. The key idea is that when $\log_g h$ *is* known, a player can open a commitment to any value (Pedersen commitments are thus *equivocable*), and in particular can show that his commitments sum to zero, even if they do not. We can thus modify the k-AMT protocol to start each round by choosing a new h so that $\log_g h$ is uniformly chosen and can be recovered exactly when $2n/3$ players reveal their secret information; each round continues as before, and at the end of each round $\log_g h$ is revealed. We note that Pedersen [Ped91b] gives an appropriate protocol for choosing h with these properties. We also note that this modification to k-AMT is incompatible with the tracing modification of Section 3.2. Thus, while applying the generic transformation to this modification of k-AMT can result in a strong selectively traceable protocol, no *efficient* construction is known.

4.2 Coercibility Preservation

Here we show that the general transformations in Section 3 preserve (up to a negligible additive factor) the coercibility of the underlying (non-traceable) anonymous communications protocol, given that the selected group signature scheme is noncoercible. That is, we will show that any proof system that has an acceptance gap of ρ in the transformed protocol can be converted into a proof system with acceptance gap at least $\rho - \mu$ for the underlying anonymous protocol if the group signature scheme is at most μ-coercible. This implies that using a noncoercible anonymous protocol will result in a noncoercible selectively traceable protocol.

Group Signature transformation. Let Π denote an anonymous communication protocol and let Π^* denote the protocol that results from applying Transformation 1 to Π. Suppose that Π^* is ρ-coercible and that the group signature scheme $\mathcal{GS}$ used in the transformation is at most μ-coercible. Then there must be a proof procedure $\mathbb{P}^* = (\mathcal{P}^*, \mathcal{V}^*)$ for Π^* with acceptance gap ρ, for some adversary A^* and a pair of players P_i and P_j. We construct a proof procedure $\mathbb{P}$ for Π, which "simulates" the group signature part of Π^* so that it can run $\mathbb{P}^*$:

- On input the public parameters from Π, $\mathcal{V}$ plays the role of the group manager in $\mathcal{GS}$ to pick a group public key GPK. $\mathcal{V}$ appends GPK to the parameters (producing a set of public parameters consistent with Π^*) and runs $\mathcal{V}^*$ to choose a message m^*. $\mathcal{V}$ computes a signing key for P_j and computes $\sigma^* = \mathtt{SIGN}_j(m^*)$. $\mathcal{V}$ also chooses the message $m = (m^*, \sigma^*)$.
- $\mathcal{V}$ and $\mathcal{P}$ jointly execute the `JOIN` protocol from $\mathcal{GS}$ to produce P_i's signing key. This is so that when $\mathcal{P}$ runs $\mathcal{P}^*$ he can supply a transcript of the `JOIN` protocol. (Note however, that if P_i sends m in Π, this view will be slightly different than if P_i sent m^* in Π^*, because m is signed by P_j. We prove, essentially, that the noncoercibility of $\mathcal{GS}$ means that this doesn't matter for the acceptance probabilities.)
- $\mathcal{V}$ appends GPK and σ^* to his input $view_A$ to form a view $view^*_A$ consistent with Π^*. Similarly, $\mathcal{P}$ appends GPK and his signing key and σ^* to $view_i$ to form a view $view^*_i$ consistent with Π^*.
- $\mathcal{V}$ executes $\mathcal{V}^*(m^*, view^*_A)$, and $\mathcal{P}$ executes $\mathcal{P}^*(m^*, view^*_i)$.
- $\mathcal{P}$ proves in zero-knowledge that his actions are consistent with the extra inputs computed with $\mathcal{V}$. If this proof fails, or $\mathcal{P}$ aborts the protocol, $\mathcal{V}$ outputs `rej` . Otherwise $\mathcal{V}$ outputs the decision of $\mathcal{V}^*$. This prevents $\mathcal{P}$ from cheating (using different inputs) to increase the acceptance probability.

Let us compute the acceptance gap of $\mathbb{P}$. To do so, we will imagine an experiment in which Π^* delivers m^* with a group signature from either P_i or P_j. Denote the event that P_i's signing key is used by S_i, and the event that P_j's key is used by S_j. Then we have that:

$$\begin{aligned}
\rho &\leq |\Pr[\mathbb{P}_i^*(P_i : m) = \texttt{acc} \ \mid S_i] - \Pr[\mathbb{P}_i^*(P_j : m) = \texttt{acc} \ \mid S_j]| \\
&\leq |\Pr[\mathbb{P}_i^*(P_i : m) = \texttt{acc} \ \mid S_i] - \Pr[\mathbb{P}_i^*(P_i : m) = \texttt{acc} \ \mid S_j]| \\
&\qquad\qquad\qquad + |\Pr[\mathbb{P}_i^*(P_i : m) = \texttt{acc} \ \mid S_j] - \Pr[\mathbb{P}_i^*(P_j : m) = \texttt{acc} \ \mid S_j]| \\
&= |\Pr[\mathbb{P}_i^*(P_i : m) = \texttt{acc} \ \mid S_i] - \Pr[\mathbb{P}_i^*(P_i : m) = \texttt{acc} \ \mid S_j]| \\
&\qquad\qquad\qquad + |\Pr[\mathbb{P}_i(P_i : m) = \texttt{acc}\ ] - \Pr[\mathbb{P}_i(P_j : m) = \texttt{acc}\ ]| \\
&\leq \mu + |\Pr[\mathbb{P}_i(P_i : m) = \texttt{acc}\ ] - \Pr[\mathbb{P}_i(P_j : m) = \texttt{acc}\ ]|
\end{aligned}$$

where the second line follows by the triangle inequality, the third follows from the definition of the proof procedure $\mathbb{P}$ — it is running $\mathbb{P}^*$ exactly in the (imaginary) case that S_j happens — and the last follows because $\mathcal{GS}$ is at most μ-coercible.[8] Thus we have that

$$|\Pr[\mathbb{P}_i(P_i : m) = \texttt{acc}\ ] - \Pr[\mathbb{P}_i(P_j : m) = \texttt{acc}\ ]| \geq \rho - \mu \ .$$

NIZK transformation. Let Π denote an anonymous communication protocol that results from applying Transformation 1, and let Π^* denote the result of applying Transformation 2 to Π, that is, adding the NIZK proofs to the protocol. We also show that if Π^* is ρ-coercible then Π is at least $\rho - \epsilon$ coercible, for a negligible function ϵ. Informally, this is because NIZK proofs are *simulatable*: a party who can choose the common reference string used for the proof can, without a witness, produce simulated proofs that are indistinguishable from accepting proofs. Because both $\mathcal{P}$ and $\mathcal{V}$ may need to generate proofs on strings that the other has not seen, they will use a *secure two-party computation protocol* [Yao86] to generate the CRS and any simulated proofs so that neither learns anything about the CRS except the proofs they need to emulate Π^*. The formal proof appears in Appendix E.

5 Conclusion

In this paper we have discussed selective tracing and coercibility as two issues that designers of anonymity protocols should bear in mind. We have described a framework for describing tracing policies that we believe to be general enough to capture most situations where fair and sensible tracing policies are desired. We have shown that, in principle, strong selectively traceable anonymity schemes for any tracing policy can be implemented by modifying a recent protocol of [ABH03].

Extending this work to protocols based on mixes is one possible direction for future work. Our proposed "Transformation 2" (in Section 3) is extremely inefficient in both space and time — more efficient transformations that apply to specific protocols (or at least to mix-style protocols) are highly desirable.

We are not advocating anonymity tracing as a necessary feature of anonymity protocols, but rather suggesting that any tracing — whether implicit (e.g., coercible protocols) or explicit — should be examined carefully so that system designers can make more specific anonymity guarantees. While it is rarely a good idea to have tracing possible by the action of a single trusted authority, it may be easier to deploy an anonymity protocol in some contexts if it includes some tracing functionality. To that end, we want to develop systems that provide flexible tracing policies that are less likely to be abused. Finally, the issue of traceable anonymity presents interesting technical problems that may help to further the goals of anonymity research. We hope that this will be the case.

Acknowledgments. The authors thank Joe Halpern, Yongdae Kim, David Molnar, Hovav Shacham, Gun Sïrer, the attendees of the Stanford Security Lunch and the U. Minnesota Security and Cryptography Seminar, and several anonymous referees for helpful discussions and comments. This work was supported by the US National Science Foundation under grants CCR-0122581, CCR-0058982, and CNS-0546162, and the US Army Research Office and the CyLab Center at Carnegie Mellon University.

[8] Suppose that $|\Pr[\mathbb{P}_i^*(P_i : m) = \texttt{acc} \ \mid S_i] - \Pr[\mathbb{P}_i^*(P_i : m) = \texttt{acc} \ \mid S_j]| > \mu$. Then $\mathbb{P}$ gives a way for P_i to prove that he did not generate the group signature σ^* with acceptance gap greater than μ: $\mathcal{V}$ and $\mathcal{P}$ run Π^* together, with $\mathcal{V}$ playing the roles of other parties, and $\mathcal{P}$ sends m^* using the group signature σ^*. Then they run $\mathbb{P}$ on their views of this execution; the acceptance gap will be preserved.

References

[ABH03] L. von Ahn, A. Bortz, and N. J. Hopper. k-anonymous message transmission. In *10th Conference on Computer and Communications Security*, pp. 122–130, 2003.

[ACJT00] G. Ateniese, J. Camenisch, M. Joye and G. Tsudik. A Practical and Provably Secure Coalition-Resistant Group Signature Scheme. *CRYPTO 2000*, pp. 255–270.

[BMW03] M. Bellare, D. Micciancio and B. Warinschi. Foundations of Group Signatures. In *Eurocrypt 2003*, (LNCS 2656), pp. 614–629.

[BGW88] M. Ben Or, S. Goldwasser and A. Wigderson. Completeness theorems for noncryptographic fault-tolerant distributed computation. In *Proc. 20th ACM STOC*, 1988.

[BDMP91] M. Blum, A. De Santis, S. Micali, and G. Persiano. Noninteractive Zero-Knowledge Proof Systems. *SIAM Journal on Computation*, 20(6): 1084–1118, 1991.

[Bon98] D. Boneh. The Decision Diffie-Hellman Problem. *Proc. 3rd ANTS*, pp 48–63, 1998.

[CDNO97] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable Encryption. In *CRYPTO 97*, pp. 90–104.

[CFGN96] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively Secure Multiparty Computation. *MIT LCS Technical Reports* TR96-682, 1996.

[Cha81] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. In *Communications of the ACM* 4(2), February 1981.

[Cha88] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[CEGP86] D. Chaum, J. Evertse, J. van de Graaf and R. Peralta. Demonstrating Possession of a Discrete Logarithm Without Revealing It. In *CRYPTO'86*, pp. 200–212.

[CvH91] D. Chaum and E. van Heyst. Group Signatures. In *EUROCRYPT '91*, pp. 257–265.

[DC05] G. Danezis and J. Clulow. Compulsion Resistant Anonymous Communications. In *7th Information Hiding Workshop*, June 2005.

[DKNS04] Y. Dodis, A. Kiayias, A. Nicolosi and V. Shoup. Anonymous Identification in Ad-Hoc Groups. In *EUROCRYPT '04*.

[GJ04] P. Golle and A. Juels. Dining Cryptographers Revisited. In *EUROCRYPT '04*.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proc. 19th ACM Symp. on Theory of Computing*, pp 218–229, 1987.

[Gre03] T.C. Greene. Net anonymity service back-doored. *The Register*, 21 August, 2003.

[GOS05] J. Groth, R. Ostrovsky and A. Sahai. Perfect Non-Interactive Zero Knowledge for NP. *Electronic Colloquium on Computational Complexity* report TR05-097, 2005.

[JJ02] A. Juels and. M. Jakobsson. Coercion-Resistant Electronic Elections. *Cryptology ePrint Archive Report 2002/165*, 2002.

[KY02] J. Katz and M. Yung. Threshold Cryptosystems Based on Factoring. In *Asiacrypt 2002*, pp. 192–205.

[KTY04] A. Kiayias, Y. Tsiounis and M. Yung. Traceable Signatures. In: *Advances in Cryptology – Eurocrypt 2004*, 2004.

[Ped91a] T.P. Pedersen. A threshold cryptosystem without a trusted party. In *Eurocrypt '91*, pp.522–526.

[Ped91b] T.P. Pedersen. Efficient and information theoretic secure verifiable secret sharing. In *CRYPTO '91*.

[RR98] M. Reiter and A. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[Sho00] V. Shoup. Practical Threshold Signatures. In *Eurocrypt 2000*.

[Yao86] A. C. Yao. How to Generate and Exchange Secrets. In *Proc. 27th IEEE FOCS*, pp. 162–167, 1986.

A Threshold El Gamal decryption

Let $\mathbb{G}_q$ be a group of prime order q, generated by g (e.g., let p, q be primes such that $q|(p-1)$, and let $\mathbb{G}_q$ be the unique order q subgroup of $\mathbb{Z}_p^*$). Let x be an integer between 0 and $q-1$. An El Gamal private key is the pair (g, x) and its corresponding public key is the tuple $(\mathbb{G}_q, q, g, y)$ where $y = g^x$. (When q, $\mathbb{G}_q$, and g are clear we refer to x and y as the public and private keys.) To encrypt a plaintext message $M \in \mathbb{G}_q$, a random integer $k \in \mathbb{Z}_q$ is selected, and the ciphertext is the pair $C = (a, b)$, where $a = g^k$ and $b = My^k$. To decrypt a ciphertext $C = (a, b)$, just take $M = b/a^x$. The encryption process is semantically secure under the Decisional Diffie-Hellman assumption [Bon98].

El Gamal encryption can be generalized so that the private key is distributed among n different principals or voters, and all n must agree to participate for each act of decryption. We describe the case for $n = 2$; the generalization to n is straightforward. Let voters v_1 and v_2 generate private keys x_1 and x_2, and publish the public keys $y_1 = g^{x_1}$ and $y_2 = g^{x_2}$. Let $y = y_1y_2$ be the aggregate public key. To decrypt the El Gamal ciphertext $C = (a, b) = (g^k, My^k)$, v_i publishes $d_i = y_i^k = (a^k)^{x_i}$ for $i = 1, 2$, and the ciphertext C can be decrypted as $b/d_1d_2 = My^k/y^k = M$. The "aggregate private key" is $x = x_1 + x_2$, but it is never computed explicitly.

Now suppose that we have n voters $v_1, \ldots, v_n$ and a threshold t, and we want decryption to occur if and only if there are t voters who take part in the decryption process. As before let v_i choose a private key x_i and publish $y_i = g^{x_i}$ so that an aggregate public key $y = \prod_i y_i$ can be computed by anyone. (The private key $x = \sum_i x_i$ cannot be computed explicitly.) Now let v_i generate shares $s_i[1], \ldots s_i[n]$ of x_i according to a linear threshold secret-sharing scheme such as Shamir's [Sha79], send $s_i[j]$ securely to v_j, and publish $g^{s_i[1]}, \ldots, g^{s_i[n]}$. After this procedure each voter can compute $X_i = \sum_j s_j[i]$, and the quantity $Y_i = \prod_j g^{s_j[i]}$ is publicly computable.

If we want to decrypt a ciphertext $C = (a, b)$, suppose that V is a set of voters with $|V| \geq t$. Because the secret-sharing scheme is linear there exists a publicly computable vector w_V such that $w_V[i] \neq 0$ only if $v_i \in V$, and $\sum_i w_V[i] \cdot Xi = x$. (For example, in Shamir's scheme, $w_V[i]$ is computed by Lagrangian interpolation: $w_V[i] = \prod_{v_j \in V, j \neq i} \frac{v_j}{v_j - v_i}$) Voters in V vote to decrypt C by publishing $d_i = a^{X_i}$. To decrypt they then calculate $\prod_i d_i^{w_V[i]} = a^x$. As before, $M = b/a^x$.

B A NIZK Primer

NIZK proofs allow a party to demonstrate noninteractively that some value y is in a language L for any $L \in NP$. An NIZK proof holds in relation to a randomly chosen *common reference string* (CRS), Σ, which can be obtained prior to the proof by a distributed computation. Σ essentially serves as a series of random challenges that a prover can only answer if he has a witness for $y \in L$. NIZK proofs also have the property that they can be *simulated*: if the prover is allowed to choose the CRS by himself, he can choose it along with some trapdoor information t such that having t will later allow him to prove, relative to Σ, arbitrary statements about L. (That is, he can show that $y' \in L$ for any y', without a witness.) These "simulated" proofs are then indistinguishable from actual proofs to any other party. It follows that an NIZK proof is secure only when the prover is not allowed to choose the CRS. For our transformation, the CRS should be chosen as a distributed computation when the other public parameters are produced. For a more thorough introduction to NIZK proofs, see [BDMP91].

As an example, an NIZK proof can be used to prove that y is the result of a polynomial-time computable function F on some input x known by the party. In this case, an NIZK proof for some value y serves as proof that the party knows x such that $F(x) = y$, and furthermore, it doesn't reveal anything more about x than is already revealed by y. If F is efficiently computable (say, polynomial time), then generic constructions are available that allow us to construct NIZK proofs for pairs $(x, y) \in F$. Because NP is closed under disjunction, NIZK proofs can be given for disjunctive statements, i.e., that a value $y \in L_1 \cup L_2$, for NP languages L_1 and L_2. NIZK proofs are polynomial in length, but for general languages of the type above, are long and not entirely practical (because the polynomial functions characterizing the length have high degree and involve large constants).

To apply NIZKs to the task of constructing a traceable anonymity scheme, let GPK be the group verification key for a group signature scheme. We construct the language $L_\Pi(GPK)$ for the underlying anonymous communication protocol Π as follows. Let

$$L_S(GPK) = \{x : x \in \texttt{SEND}((m, \sigma), \ldots) \wedge \texttt{VERIFY}(GPK, m, \sigma) = TRUE)\} ,$$

that is, $L_S(GPK)$ is the set of legitimate protocol messages generated by SEND on a (group)-signed anonymous message. $L_S(GPK) \in NP$, since (m, σ) serves as a polynomial-length witness for $x \in L_S(GPK)$ when σ is a group signature on m. Let

$$L_P(GPK) = \{x : x \in \texttt{PROCESS}(c_1, \ldots, c_n, \ldots),\ \bigwedge_{i=1}^{n}(c_i \in L_S(GPK) \vee c_i \in L_P(GPK))\}\ ,$$

so that $L_P(GPK)$ is the set of legitimate protocol messages generated by PROCESS with inputs that were "legitimate" protocol messages (i.e., generated by some sequence of SEND executions on signed anonymous messages and subsequent PROCESS executions). For any polynomial-time protocol there is a polynomial-length witness for the statement $x \in L_P(GPK)$ as well: the original signed messages and the random bits used by all parties in the protocol so far.

Similarly, we define the language

$$L_R(GPK) = \{x : x \in \texttt{RECOVER}(c_1, \ldots, c_n, \ldots),\ \bigwedge_{i=1}^{n}(c_i \in L_P(GPK))\}\ ,$$

which is the set of output messages originating from signed input messages and is similarly in NP. Finally, we define the language

$$L_\Pi(GPK) = L_S(GPK) \cup L_P(GPK) \cup L_R(GPK)\ .$$

The transformed protocol works as follows: the new SEND procedure executes the original SEND procedure on a pair $(m, \texttt{SIGN}(m))$ to produce a vector of protocol messages c_i, and then produces an NIZK proof π_i that each protocol message c_i is in $L_\Pi(GPK)$. The transformed protocol messages are then pairs (c_i, π_i). The new PROCESS procedure on a tuple of protocol messages each of the form (c_j, π_j) first checks that all proofs are correct; if for some j the proof is incorrect then (c_j, π_j) is replaced by $(\varepsilon, \varepsilon)$ (ε here just means "the empty message"). Then the original PROCESS procedure is invoked on $(c_1, \ldots, c_n)$ to produce a vector of original protocol messages $(c'_1, \ldots, c'_n)$, and the proofs π_i are used as witnesses to produce NIZK proofs π'_i that $c'_i \in L_\Pi(GPK)$. The transformed protocol messages are again the pairs (c'_i, π'_i). Finally, the new RECOVER procedure is similarly transformed, as follows: the protocol messages are checked for correctness, and if any proof fails the corresponding original protocol message is left empty; the original RECOVER procedure is executed on the vector $(c_1, \ldots, c_n)$ to produce output message (m, σ) and a proof is produced that $m \in L_\Pi(GPK)$; finally, (m, σ) is output along with the proof.

We note that in some protocols with information-theoretic anonymity (e.g., DC-Nets [Cha88]), the naive application of the above transformation may fail, since each single protocol message is uniformly distributed and independent of the message being transmitted. (In this case, a proof that the protocol message P_i sends to P_j is consistent with some input is meaningless, because *any* such message would be consistent.) We note, however, that a well-known approach to the problem, described in the multiparty computation literature [BGW88,GMW87], can be applied here: first, modify the setup procedure to output a vector of commitments, one to each player's secret input; then modify $\texttt{PROCESS}(c_1, \ldots, c_n)$ to output commitments to $c_1, \ldots, c_n$ as well commitments to $c'_1, \ldots, c'_n$; finally, apply the above transformation. If these commitments are unconditionally hiding, they do not alter the anonymity of the underlying protocol.

C Security proof sketch for traceable *k*-AMT

Theorem 1. *Let A be a polynomial time adversary, and let B be the set of players corrupted by A when attacking the selectively traceable modification of k-AMT. Then, if $B \notin \mathcal{V}$, A gains no advantage over an attack on unmodified k-AMT.*

Proof. (Sketch) The theorem follows from the security of threshold El-Gamal encryption. Suppose A can distinguish between two possible senders in the modified k-AMT protocol, without corrupting a tracing set. Then we can design an adversary A' against the original protocol who can distinguish between the senders with essentially the same advantage. A' works as follows: after seeing every message sent in the unmodified k-AMT protocol, A' will submit a *simulated* message from the modified protocol to A; whatever messages A

decides to send in the modified protocol can be sent in the original k-AMT protocol with tracing information stripped out. As long as the *simulated* messages generated by A' are consistent with (speaking technically, *indistinguishable from*) the messages that would be generated in the modified protocol, A will have the same advantage in determining the actual sender of a message in the run of the original k-AMT protocol. (Since all parties' private inputs and outputs would be the same.) Thus A' can output the decision of A and correctly determine the true sender with the same advantage.

Thus it remains to show that A' can simulate messages from the modified protocol given access to messages from the original protocol. The only step in the original protocol that is modified is the commitment phase, in which each party sends, in addition to his commitments from the original k-AMT, an ElGamal encryption of the message sent on each slot, and a zero-knowledge proof that his commitments and encryption are consistent. Using the fact that threshold ElGamal is secure in the *indistinguishability* sense, we know that A' can simulate this portion of the modified commitment scheme by producing an encryption of 1. Using the fact that zero-knowledge proofs are simulatable, A' can also make a simulated (but possibly false) proof that the commitments and the encrypted values in a given party's broadcast message are consistent. Because true and false statements are indistinguishable in this case, it will hold that the simulated encryption and proof are indistinguishable from a consistent encryption and correct proof. All other messages in the modified protocol can be passed directly to A by A'.

D Formal description of modified GJ DC-Net

- **Modified Setup Phase**. As before P_i picks private key x_i and publishes public key $y_i = x_i P$. P_i also generates shares $z_{i,1}, \ldots, z_{i,m}$ to be sent to voters $v_1, \ldots, v_m$ using a linear secret sharing scheme consistent with the tracing policy $\mathcal{V}$. P_i publishes the values $Q_{i,t} = z_{i,t}P$ and sends v_t the share $z_{i,t}$. All players verify that the published values are consistent (for example, that there are at least m sufficient sets $V \in \mathcal{V}$ such that $\sum_{v_t \in V} w_V[t]Q_{i,t} = y_i$) and each voter v_t verifies that $Q_{i,t} = z_{i,t}P$ for all i.
- **Tracing Procedure**. To "vote" to trace message $m(s)$, voter v_t publishes the values $Z_{i,t} = z_{i,t}Q_s$ for $i \in [n]$. Once a sufficient set $V \in \mathcal{V}$ have voted to trace, the values $k_{i,j}(s)$ can be reconstructed by computing

$$k_{i,j}(s) = \hat{e}(y_i, \sum_{v_t \in V} w_V[t]Z_{j,t}) = \hat{e}(y_i, \sum_{v_t \in V} w_V[t]z_{j,t}Q_s) = \hat{e}(y_i, x_j Q_s)$$

  Once the values $k_{i,j}(s)$ are reconstructed, tracing proceeds as described above: $p_i(s)$ is computed as in the Pad Construction phase, and $W_i(s)$ is compared to $p_i(s)m(s)$; if they are equal, P_i is responsible for the message $m(s)$. Notice that fraudulent voting can be detected in this protocol: it is easy to verify that the value $Z_{i,t}$ published by v_t is consistent by checking that $\hat{e}(Q_{i,t}, Q_s) = \hat{e}(P, Z_{i,t})$.

E Proof that Transformation 2 preserves coercibility

Formally, let $\mathbb{P}^* = (\mathcal{P}^*, \mathcal{V}^*)$ have acceptance gap ρ for Π^*. Then we construct the system $\mathbb{P}$ for Π as follows:

- $\mathcal{P}$ and $\mathcal{V}$ jointly execute a secure two-party computation to choose a simulated CRS Σ^* and random shares (s_P, s_V) of the trapdoor for Σ^*.
- $\mathcal{V}$ appends Σ^* to the public parameters for Π to produce parameters consistent with Π^*. V runs V^* on these parameters and outputs the message m^* chosen by V^*.
- On inputs $view_i, m^*$ and $view_A, m^*$ to $\mathcal{P}, \mathcal{V}$ respectively, the parties simulate NIZK proofs for all messages in each of their views:
  - For each protocol message $m \in view_i$, $\mathcal{P}$ and $\mathcal{V}$ run a secure two-party computation in which $\mathcal{P}$'s input is (m, s_P), and $\mathcal{V}$'s input is s_V; $\mathcal{P}$'s output is a simulated proof that $m \in L_\Pi(GPK)$, and $\mathcal{V}$'s output is ε.

- For each protocol message $m \in view_A$, $\mathcal{P}$ and $\mathcal{V}$ run a secure two-party computation in which $\mathcal{V}$'s input is (m, s_V), and $\mathcal{P}$'s input is s_P; $\mathcal{V}$'s output is a simulated proof that $m \in L_\Pi(GPK)$, and $\mathcal{P}$'s output is ε.

  At the end of this process, $\mathcal{P}$ knows a list $\pi_\mathcal{P}$ of proofs and $\mathcal{V}$ knows a list $\pi_\mathcal{V}$ of proofs. Each party incorporates these proofs into his view appropriately, producing views $view_i^*, view_A^*$ consistent with Π^*.
- $\mathcal{P}$ and $\mathcal{V}$ run $\mathcal{P}^*(m^*, view_i^*), \mathcal{V}^*(m^*, view_A^*)$.
- $\mathcal{P}$ proves in (interactive) zero knowledge that his actions in $\mathbb{P}^*$ are consistent with the additional information computed previously. If this proof fails, or if at any point $\mathcal{P}$ aborts the protocol, $\mathcal{V}$ outputs `rej` , otherwise $\mathcal{V}$ outputs the decision of $\mathcal{V}^*$.

Since the simulated proofs $\pi_\mathcal{P}, \pi_\mathcal{V}$ are indistinguishable from proofs produced in Π^*, it should be clear that the acceptance probabilities of $\mathbb{P}$ in either case are the same as those of $\mathbb{P}^*$, up to a negligible additive factor ϵ, which is the negligible probability that the simulation procedure for (Σ^*, t) fails. Thus the acceptance gap for $\mathbb{P}$ is at least $\rho - \epsilon$, as claimed.