Selectively Traceable Anonymity

Luis von Ahn, Andrew Bortz, Nicholas J. Hopper, and Kevin O'Neill

Abstract. One serious impediment to the widespread adoption of anonymous communication protocols is the threat of abuse. Anonymous communication can, by its very nature, facilitate socially unacceptable behavior. In this paper we present one approach to dealing with abuse: selective traceability. A system for anonymous communication is said to be *selectively traceable* if it allows the tracing of a message's sender after a set of sensible conditions have been met (e.g., both the FBI and the ACLU have agreed to trace the message, or 90% of the users agree that the message should be traced, etc.). We introduce general techniques to transform a large class of anonymous communication protocols (including most of those that have been proposed so far, such as DC-Nets, Mix-Nets, and their derivatives) into selectively traceable anonymous communication protocols. We also present more efficient modifications to two existing anonymous communication protocols that do not affect the asymptotic efficiency of the underlying schemes. Our resulting protocols are provably secure against malicious adversaries.
We also study anonymity protocols where participants can *prove* that they did not send a particular message, and which therefore allow a form of ad-hoc and uncoordinated tracing. If a protocol has this undesirable property, we call it *exculpable*. Exculpability is not usually addressed in the anonymous communication literature, but because some anonymity protocols are exculpable and others are not, we believe it is important to do so. We discuss the notion of exculpability in anonymous communication, and show, in particular, how the state-of-the-art proposals in DC-Net-based protocols allow exculpation. We also show that our generic transformations do not alter the exculpability (or non-exculpability) of the underlying protocols.

1 Introduction

Protocols for anonymous communication have been studied extensively in the literature. (For recent examples, see [AKP03,DS03,ABH03,KORS04,GJ04,GRPS02,LS02,RR98,SBS02,SGR97].) The goal of an anonymous communication protocol is to allow parties to send and receive communication anonymously. Examples of such communication include email, Web browsing, and, more generally, the sending and receiving of TCP packets. Anonymous communication has several important potential applications that include 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 the potential 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 illegal pornography. In some cases, especially when the anonymity guarantees provided by a system are very strong, the negative consequences of allowing users to communicate anonymously outweigh the benefits, thus creating a major stumbling block for the wide-spread 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 not formally addressed "revoking" the anonymity of a message. In this paper we argue that it would be useful to have anonymity protocols that 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*.

Another reason for looking closely at tracing in anonymity protocols is that some existing anonymity protocols already allow a certain form of ad hoc tracing by allowing participants to *prove* that they did not send some particular message. If a protocol has this property, we call it *exculpable*, because participants can prove that they are "not to blame" for a particular message. Exculpability is related to tracing in that a protocol that is exculpable allows a certain form of gradual and uncoordinated tracing: every participant except the sender can show that they did not send the

message. We claim that tracing based on exculpability is undesirable because it can be unfair and inadvertent. While similar to the notions of *coercibility* in election protocols [A04,BT94,JJ02], *deniability* in encryption [CDNO97], and *adaptive security* in multiparty computation [CFGN96], exculpability is not usually addressed in the anonymous communication literature, but because some anonymity protocols are exculpable and others are not, we believe it is important to do so.

We present two definitions of traceable anonymity. In one, which we refer to as *weak traceable anonymity*, a message should be traced whenever the revocation policy is satisfied, and 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 exculpable: the message can be traced when the tracing policy is satisfied, but something about the identity of the sender can be revealed even if the tracing policy is not satisfied, by having one of the participants prove they did not send the message. A strong traceable protocol does not allow exculpation.

We are cognizant of the fact that tracing, like anonymity, may be abused. In particular, we want to avoid any requirements that tracing information be logged 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* $R \in \mathcal{V}$ agree to it. (We assume that $\mathcal{V}$ is *monotone*, so that if $R \in \mathcal{V}$ and $R \subseteq R'$, then $R' \in \mathcal{V}$. 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, $\mathcal{V} = \emptyset$, in which tracing is not allowed.
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 judges on the Supreme Court of the United States, and let $\mathcal{V} = \mathcal{V}(5)$. Under this tracing scheme, a majority of the judges must agree to tracing before it takes place.
4. Let V consist of the FBI and the CIA, and let $\mathcal{V} = \mathcal{V}(1) = \{\{\text{FBI}\}, \{\text{CIA}\}\}$. With this policy, the FBI or the CIA can trace whenever they please.

We present definitions and several technical results relating to selectively traceable anonymous communication. We remark that selectively traceable anonymous communication can be formulated as a multiparty computation [GMW87], and that any adaptively secure protocol for general multiparty computation [CFGN96] would satisfy our definitions. The solution thus obtained, however, would be extremely inefficient and we do not pursue it. Our technical results include:

– **A generic and efficient transformation from any anonymous communication protocol against honest-but-curious adversaries.** We show that a large class of systems for anonymous communication can be transformed into systems with selectively traceable anonymity that is secure against honest-but-curious adversaries. The transformation uses group signatures [CvH91,ACJT00], which allow a member of a group to sign a message so that it can be verified that *someone* in the group signed it, but not which particular member of the group did. Group signatures also allow a designated group manager (which can be implemented distributedly) to revoke the anonymity of any signature. The idea of our basic transformation is to append a

group signature to every message sent on an anonymous communication protocol and *requiring the receivers to drop all messages that are not signed.* To trace a message, a tracing set revokes the anonymity of its signer.

It is crucial in this construction that the receivers drop messages that are not signed. If the receivers are curious and actually do look at unsigned messages, the system can degrade into a non-traceable anonymous communication protocol: whenever a participant wants to send a message that cannot be traced, they simply send it without a signature. Since in general receivers will have an incentive (curiosity) to read unsigned messages before dropping them, we believe this to be a serious game-theoretic problem of the construction. The tracing functionality of this construction, therefore, is only secure against honest-but-curious adversaries. (The construction does not change the anonymity properties of the underlying anonymous protocol.)

- **A generic but less efficient transformation that yields security against malicious adversaries.** The above game-theoretic problem can be corrected using generic non-interactive zero-knowledge proofs, by requiring the sender of a message to prove, in zero knowledge, that the message to be sent is signed with a group signature. The resulting construction, though generic and more efficient than general adaptively secure multiparty computation, is somewhat inefficient due to the inefficiency of generic zero-knowledge proofs. More efficient protocols could be constructed in this manner by using a specific anonymous communication protocol and a specific group signature and devising efficient zero-knowledge proofs specific for the resulting language.
- **Two efficient transformations from specific DC-Net-based 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.
- **Exculpability results.** We discuss the notion of exculpability in anonymous communication, and show, in particular, how the DC-Net-based protocols in [ABH03,GJ04] allow exculpation. We also show that our generic transformations do not alter the exculpability (or non-exculpability) of the underlying protocols.

2 Threshold Cryptography and Group Signatures

We use two main building blocks for the technical results that follow: threshold El Gamal encryption 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 later be revoked by the "group manager."

Distributed El Gamal Decryption

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 revocation policy, however, we want decryption to occur only when all the voters in some revocation set $R \in \mathcal{V}$ agree to take part. In other words, we describe 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 revocation group $R \in \mathcal{V}$ take part in the decryption.

To implement such a scheme, we use threshold El Gamal encryption [P91], which we now describe for completeness.

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 [B98].

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 $y = y_1 y_2$ 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 $b/d_1 d_2 = My^k/y^k = M$. 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, $M = b/a^x$.

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`: *On input a security parameter ℓ, this probabilistic algorithm outputs the initial group public key GPK (including all system parameters) and the secret key S for the group manager.*
- `JOIN`: *A protocol between the group manager and a user that results in the user becoming a new member of the group. The user's output is a membership certificate and a membership secret.*
- `SIGN`: *A probabilistic algorithm that on input a group public key, a membership certificate, a membership secret, and a message m outputs group signature of m.*
- `VERIFY`: *An algorithm for establishing the validity of an alleged group signature of a message with respect to a group public key.*
- `OPEN`: *An algorithm that, given a message, a valid group signature on it, a group public key and a 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 Ateniese, Camenisch, Joye, and Tsudik [ACJT00].

More than one recent group signature scheme (including [ACJT00,CS03,CG04,KTY04]) implements `OPEN` as an instance of El Gamal decryption. For such schemes it is possible to distribute the functionality of the group manager so that each instance of `OPEN` operates according to a threshold scheme. (Similarly, `JOIN` can typically be implemented using a distributed threshold signature scheme [Sho00,KY02].)

3 Generic Transformations

In this section we present a way to convert a generic anonymous communication protocol to a new protocol that permits selective anonymity revocation. We assume that there is an independent set V of voters and a threshold revocation policy $\mathcal{V} \subseteq 2^V$. (We remark that any monotone revocation 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.

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 (the operations use a set of public parameters selected by an initial setup stage):

- `SEND`: a procedure executed by P_i that takes as input an anonymous message $m \in \mathcal{M}$, a recipient P_j, and a set S_i of secret parameters, and outputs a vector $\boldsymbol{c}$ (or multiple vectors) of protocol messages $c_{i,1}, \ldots, c_{i,n} \in \mathcal{PM}$ to be sent to $P_1, \ldots, P_n$.
- `PROCESS`: a procedure executed by P_j that takes as input a vector of protocol messages $c_{1,j}, \ldots, c_{n,j}$ received from $P_1, \ldots, P_n$, as well as secret parameters S_j, and outputs a new vector $\boldsymbol{c'}$ of protocol messages $c'_{j,1}, \ldots, c'_{j,n} \in \mathcal{PM}$ to be sent to $P_1, \ldots, P_n$. (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 vector $\boldsymbol{c}$ (or multiple vectors) of protocol messages $c_{1,j}, \ldots, c_{n,j}$ received from $P_1, \ldots, P_n$, as well as secret parameters S_j, and outputs a message $m \in \mathcal{M}$ to be sent to a party 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 protocol message from a single party and forwards it along to another party, possibly after performing some operation on the message such as encryption and/or decryption.

For our generic transformations we also assume that some appropriate set of parties (such as the voters, the group members, or a trusted certification authority) implement a "group manager," in a distributed fashion, executing the `JOIN` protocol of the group signature scheme to allow new members to send and receive messages anonymously in the system.

Transformation 1: The first transformation we propose affects the `SEND` and `RECOVER` steps of a given protocol. In the new protocol the sender P_i must sign the message $m \in \mathcal{M}$ using group signatures, and the resulting message m' 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 revocation, a revocation set $R \in \mathcal{V}$ may open the signature to reveal the sender of the message.

A significant problem with Transformation 1 is that nothing stops the party P_j executing `RECOVER` from reading the recovered message, or sending it on to its intended recipient, simply out of curiosity or out of a desire to subvert the revocation scheme. In some protocols, such as those based on DC-Nets, the `RECOVER` operation is done publicly, so *everyone* sees the original message, and may look at it out of curiosity. 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-revocable protocol.

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].

Detour: 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 only secure when the prover is not allowed to choose the CRS; for our transformation it 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).

Let GPK be the group verification key for a group signature scheme. To use NIZK proofs we construct the language $L_{\Pi}(GPK)$ for the underlying anonymous communication protocol Π as follows. Let

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

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

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

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

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

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 $(c_1, \dots, c_n)$ to produce a vector of original protocol messages $(c'_1, \dots, c'_n)$, 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 $(c_1, \dots, c_n)$ 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.

The protocol just described is secure against malicious adversaries and preserves the anonymity guarantees of the underlying protocol.

4 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.

4.1 *k*-AMT

The k-AMT protocol implements a shared channel as a secure multiparty sum computation, using Pedersen commitments[1] to ensure correctness. The basic protocol has four phases:

1. **Commitment Phase:**
   - P_i splits $X_i \in \mathbb{Z}_q$ into n random shares $s_{i,1}, ..., s_{i,n}$
   - P_i chooses $r_{i,j} \leftarrow \mathbb{Z}_q$
   - P_i computes commitments $C_{i,j} = C_{r_{i,j}}(s_{i,j})$
   - P_i broadcasts $\{C_{i,j} : 1 \leq j \leq n\}$
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 the values $R_i = \sum_j r_{j,i} \bmod q$ and $S_i = \sum_j s_{j,i} \bmod q$
   - P_i broadcasts (R_i, S_i)
   - All players check that $C_{R_i}(S_i) = \prod_j C_{j,i} \bmod p$
4. **Result:**
   Each player computes the result as $X = \sum_i S_i \bmod q$, computes $R = \sum_i R_i \bmod q$ and checks that $C_R(X) = \prod_{i,j} C_{i,j} \bmod p$

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 the additional procedures defined in the k-AMT protocol.

The new protocol essentially capitalizes on 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.

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 revocation policy, as in Section 2.

[1] 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 $C_r(x) = g^r h^x$.

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 non-interactive 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).
   To verify a certificate, which all parties will do for all broadcast commitments, simply verify the proof of knowledge.

Now, to trace a message: identify the slot that it was transmitted on, obtain a number of parties as required by the revocation 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 send 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.[2] Note that this modification doesn't affect the asymptotic efficiency of the underlying protocol.

4.2 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] 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].

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.
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.

More formally, we describe the new procedures below:

– **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

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

   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 $\hat{e}(Q_{i,t}, Q_s) = \hat{e}(P, Z_{i,t})$.

We note, in passing, that in the full GJ protocol [GJ04] shares of the private keys x_i are distributed amongst the players to allow them to reconstruct the pads of players who do not participate in any given session. This is done so that players can compute the output of DC-net without full participation, but it follows that the GJ protocol *already* provides a threshold tracing scheme, with the players themselves acting as voters.

5 Exculpability in Anonymous Protocols

Informally, we say that an anonymity protocol is exculpable if every player who did not send a message can produce a proof that this is the case. More formally, we say that a protocol is ρ-exculpable if there is a separate "proof protocol" $\mathbb{P}$ between a player P_i and a verifier V such that 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 ρ. (In other words, ρ measures the confidence of the proof procedure.) If a protocol is 1-exculpable, only the legitimate sender of a message cannot exculpate himself (but everybody else can); if a protocol is 0-exculpable the verifier should not believe any proofs. If ϵ is negligible, we say that a protocol that is $(1-\epsilon)$-exculpable is *strongly exculpable*, and that a protocol that is at most ϵ-exculpable is *non-exculpable*. If a protocol is ρ-exculpable for some constant ρ, we say that it is *plausibly non-exculpable*.

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 — see [HO03] for one formalization of this definition) lets us ignore acts of exculpation that can arise simply because two players belong to different anonymity sets.

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

- A *proof procedure* $\mathbb{P}$ is a pair $(\mathcal{P}, \mathcal{V})$ of programs such that $\mathcal{V}$ outputs either `accept` or `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 $view_{P_i}(X : m)$).
- Let P_i and P_j be arbitrary players in Π. We say that Π is *ρ-exculpable* if there is a proof procedure $\mathbb{P}$ and an adversary A such that

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

  regardless of P_i's actions in the second case.

Notice that this definition is in some sense "weak" because the adversary (verifier) is allowed to choose the message. This makes non-exculpability a stronger definition, however. The exculpability

of several protocols from the literature satisfies a stronger definition, in which the message is not chosen by the adversary; we will demonstrate this below.

Exculpability for group signature schemes can be defined analogously. We remark that the "full anonymity" requirement of [BMW03] implies that group signatures satisfying their definition are non-exculpable.

5.1 Exculpability Preservation

Here we show that the general transformations in Section 3 preserve (up to a negligible additive factor) the exculpability of the underlying (non-traceable) anonymous communications protocol, given that the selected group signature scheme is non-exculpable. 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-exculpable. This implies that using a non-exculpable anonymous protocol will result in a non-exculpable selectively traceable protocol.

The proof will proceed in two steps. First, we will show that simply adding group signatures to the messages (i.e., Transformation 1 of Section 3) preserves the exculpability properties of the underlying anonymous communication protocol. Afterwards, we will prove that adding the NIZK proofs also preserves the exculpability.

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-exculpable and that the group signature scheme $\mathcal{GS}$ used in the transformation is at most μ-exculpable. 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 $\sigma^* = \texttt{SIGN}_j(m^*)$. $\mathcal{V}$ 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 non-exculpability 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 `reject`. Otherwise $\mathcal{V}$ outputs the decision of $\mathcal{V}^*$. This ensures that $\mathcal{P}$ does not "cheat" by 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:

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

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-exculpable.[3] Thus we have that

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

as claimed.

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 will show that if Π^* is ρ-exculpable then Π is at least $\rho - \epsilon$ exculpable, 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^*.

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.

[3] Suppose that $|\Pr[\mathbb{P}_i^*(P_i : m) = \texttt{accept} \mid S_i] - \Pr[\mathbb{P}_i^*(P_i : m) = \texttt{accept} \mid S_j]| > \mu$. 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.

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 `reject`, 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.

5.2 Exculpability in *k*-AMT and GJ DC-Nets

In Section 4 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 exculpable. 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 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.

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

$$p_i(s) = \prod_j k_{i,j}(s)^{\delta_{i,j}} = \prod_j \hat{e}(y_j, z_i(s))^{\delta_{i,j}}.$$

Much the same as with k-AMT, $W_i(s)$ — 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.

We have argued that exculpability is an undesirable property for anonymity protocols. However, these two protocols represent the state-of-the-art for anonymity systems that are cryptographically secure in the presence of malicious adversaries as well as robust against non-participation and denial-of-service attacks. (Chaum's original DC-Net protocol and its variants [Cha88,WP89,Wai89] were not exculpable because the shared secrets used to compute pads were computed interactively and participants could plausibly deny having used any particular pad.) Whether there exists a strong traceable DC-Net-based protocol is an interesting open question.

6 Conclusion

In this paper we have discussed selective tracing and exculpability as two issues that designers of anonymity protocols should bear in mind. We have described a general 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 also proposed two generic solutions for adding selective tracing to anonymity protocols, and have shown how two recent anonymity protocols based on DC-Nets can be made weakly traceable.

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 in general) are highly desirable. There are also protocols for which our second transformation is not helpful. The Crowds system [RR98], where messages are sent as plaintext, is one example. Crowds is also an example of a system that is scalable to very large networks, and all of the implementations for tracing described here are not scalable to the same degree because they require voters to keep track of all the users of the anonymity system.

In closing, we want to stress that we are not advocating anonymity tracing as a necessary feature of anonymity protocols, and we are definitely not advocating tracing policies that make use a single trusted authority. Our goal is to point out that tracing is already an implicit feature of several existing protocols (e.g., exculpable protocols) and that tracing should therefore be dealt with more explicitly — even if the tracing policy $\mathcal{V}$ is simply $\emptyset$ so that tracing is impossible. We also claim that anonymity systems may be easier to deploy in some contexts if they include tracing functionality, and to that end we want to develop systems that provide 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.

References

[AKP03] D. Agrawal, D. Kesdogan, and S. Penz. Probabilistic Treatment of MIXes to Hamper Traffic Analysis. In *2003 IEEE Symposium on Security and Privacy 2003*, pages 16–27, 2003.

[A04] A. Acquisti. Receipt-Free Homomorphic Elections and Write-in Ballots. *Cryptology ePrint Archive Report 2004/105*, 2004.

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

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

[AM03] G. Ateniese and B. de Medeiros. Efficient Group Signatures without Trapdoors. In: *Advances in Cryptology – Asiacrypt 2003*, 2003.

[BMW03] M. Bellare, D. Micciancio and B. Warinschi. Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions. In: *Advances in Cryptology — Eurocrypt 2003, Lecture Notes in Computer Science Vol. 2656*, pages 614–629.

[BT94] J.C. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections (extended abstract). In: *Proceedigns of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, pages 544–553, 1994.

[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.

[B98] D. Boneh. The Decision Diffie-Hellman Problem. *Third Algorithmic Number Theory Symposium*, pp 48–63, 1998.

[CG04] J. Camenisch and J. Groth. Group Signatures: Better Efficiency and New Theoretical Aspects. In *Fourth Conference on Security in Communication Networks - SCN '04*, 2004.

[CS03] J. Camenisch and V. Shoup. Practical Verifiable Encryption and Decryption of Discrete Logarithms. In *Advances in Cryptology - Crypto 2003*, pages 126–144, 2003.

[CDNO97] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable Encryption. In: *Advances in Cryptology – CRYPTO 97*, pages 90–104, 1997.

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

[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. *Advances in Cryptology: CRYPTO'86*, pages 200-212, 1987.

[CvH91] D. Chaum and E. van Heyst. Group Signatures. *Advances in Cryptology, EUROCRYPT '91*, pages 257-265.

[DS03] C. Díaz, A. Serjantov. Generalising Mixes. In *Privacy Enhancing Technologies 2003*, pages 18–31, 2003.

[GJ04] P. Golle and A. Juels. Dining Cryptographers Revisited. *Advances in Cryptology – Eurocrypt '04*, 2004.

[GRPS02] S. Goel, M. Robson, M. Polte, and E. G. Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Unpublished manuscript, 2002.

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

[HO03] J. Y. Halpern and K. O'Neill. Anonymity and information hiding in multiagent systems. *Journal of Computer Security*, to appear.

[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: *Advances in Cryptology – Asiacrypt 2002*, pages 192–205, 2002.

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

[KORS04] L. Kissner, A. Oprea, M.K. Reiter, D.X. Song, K. Yang. Private Keyword-Based Push and Pull with Applications to Anonymous Communication. In: *Proceedings of ACNS 2004*, pages 16–30, 2004.

[LS02] B.N. Levine and C. Shields. Hordes: A multicast based protocol for anonymity. *Journal of Computer Security*, 10(3):213–240, 2002.

[P91] T.P. Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology — Eurocrypt '91*, pages 522–526, 1991.

[PK03] A. Pfitzmann and M. Köhntopp. Anonymity, unobservability, and pseudonymity: A proposal for terminology. Draft, version 0.14, 2003.

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

[SBS02] R. Sherwood, B. Bhattacharjee, and A. Srinivasan. P5: A protocol for scalable anonymous communication. In *IEEE Symposium on Security and Privacy*, pages 58–70, 2002.

[SGR97] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous connections and onion routing. In *IEEE Symposium on Security and Privacy*, pages 44–54, 1997.

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

[Sho00] V. Shoup. Practical Threshold Signatures. In: *Advances in Cryptology – Eurocrypt 2000*, 2000.

[Wai89] M. Waidner. Unconditional sender and recipient untraceability in spite of active attacks. In: *Advances in Cryptology – EUROCRYPT'89*, pages 302-319, 1989.

[WP89] M. Waidner and B. Pfitzmann. The Dining Cryptographers in the Disco - Unconditional Sender and Recipient Untraceability with Computationally Secure Serviceability (Abstract). In: *Advances in Cryptology – EUROCRYPT 1989*, page 690, 1989.

[Yao86] A. C. Yao. How to Generate and Exchange Secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, 1986, pages 162–167.