Dining Cryptographers Revisited

Philippe Golle[1] and Ari Juels[2]

[1] Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
E-mail: pgolle@parc.com
[2] RSA Laboratories
Bedford, MA 01730, USA
E-mail: ajuels@rsasecurity.com

Abstract. Dining cryptographers networks (or DC-nets) are a privacy-preserving primitive devised by Chaum for anonymous message publication. A very attractive feature of the basic DC-net is its *non-interactivity*. Subsequent to key establishment, players may publish their messages in a single broadcast round, with no player-to-player communication. This feature is not possible in other privacy-preserving tools like mixnets. A drawback to DC-nets, however, is that malicious players can easily jam them, i.e., corrupt or block the transmission of messages from honest parties, and may do so without being traced.
Several researchers have proposed valuable methods of detecting cheating players in DC-nets. This is usually at the cost, however, of multiple broadcast rounds, even in the optimistic case, and often of high computational and/or communications overhead, particularly for fault recovery.
We present new DC-net constructions that simultaneously achieve non-interactivity and high-probability detection and identification of cheating players. Our proposals are quite efficient, imposing a basic cost that is linear in the number of participating players. Moreover, even in the case of cheating in our proposed system, just one additional broadcast round suffices for full fault recovery. Among other tools, our constructions employ bilinear maps, a recently popular cryptographic technique for reducing communication complexity.

Keywords: anonymity, dining cryptographers, mix network, non-interactive, privacy.

1 Introduction

Anonymous message transmission is a fundamental privacy-preserving tool, both in the literature and in practice. Toward this aim, Chaum devised two seminal techniques: mixnets [10] and "dining-cryptographers" nets [11], also known as DC-nets. Mixnets have seen broad exploration in the literature, and serve as the basis for several fielded anonymity systems, e.g., [3, 13, 17, 19]. (See [14] for a good bibliography.) DC-nets, by contrast, have remained relatively neglected, apart from a small scattering of papers, e.g., [1, 2, 11, 21, 22]. One reason for this is perhaps that DC-nets, unlike mixnets, cannot operate by proxy; in particular, the players operating a DC-net must be identical with those providing input. In many real-world cases, however, this is not necessarily a serious drawback, as in the Crowds system [19], where participants provide mutual protection of privacy. Moreover, as formulated by Chaum for the case involving honest players, DC-nets have one very compelling feature unavailable in mixnets:

> In a basic DC-net, anonymous message transmission may be accomplished by players in a *non-interactive* manner, i.e., in a single broadcast round.

Non-interactivity is of course very naturally attractive as a practical feature of system design. It also renders security definitions and proofs simpler than in the case of mixnets (for which formal definitions have been quite elusive).

There is a major drawback to DC-nets, however, and a large obstacle to their deployment: They are subject to straightforward jamming by malicious players. Such players can prevent the delivery of messages from honest participants, either by broadcasting invalid messages or even simply by dropping out of the protocol. Several valuable techniques have been proposed for addressing this problem, but to this point have had the limitation of requiring either unfeasibly intensive computation and/or multiple rounds of interaction among players.

Our first contribution in this paper is a set of techniques permitting the identification of cheating players with very high probability, *while retaining the property of non-interactivity.* The resulting DC-net constructions are computationally efficient: Assuming n players, they require each participant to perform a number of modular exponentiations that is linear in n during the broadcast phase. Any player, whether a participant or not, may perform a quadratic number of exponentiations for verification of the output. Indeed, the computational costs of our constructions are comparable to those of the most efficient mixnets (assuming n players processing n inputs). Our DC-net proposals are therefore reasonable for small sets of, say, some dozens of players.

Of equal importance, we propose techniques that permit recovery from lost or corrupted messages in a single, additional broadcast round, provided that there is a majority of honest players. Previous proposals have required multiple rounds for this purpose, or assumed a re-broadcast of messages. The computational costs for our recovery protocol are comparable to those for the basic message-transmission protocol.

Although it is possible to detect cheating by a player in a non-interactive mix network, we maintain that under any reasonable set of security assumptions, it is not possible for such a mix network to recover from failure (and thus from cheating) by even one player without an additional round of interaction. Our reasoning is as follows. Suppose that we could recover the inputs of all participating players regardless of who participated. Then if a given player P_i did participate, and furnished message m_i as input, an adversary could determine m_i by taking the difference between the set M of all messages submitted and the set M' of all messages except that of P_i (the adversary would obtain M' by simulating the absence of P_i).

We describe two different DC-net constructions, which we characterize as *short* and *long*. In a short DC-net, the basic unit of message transmission is an algebraic group element. For such DC-nets, we propose techniques that detect cheating with overwhelming probability. A long DC-net, by contrast, permits efficient transmission of messages of arbitrary length essentially by means of a form of hybrid encryption. (It may be viewed as roughly analogous to a "hybrid" mixnet.) For long DC-nets, we describe techniques to detect cheating with high, but not overwhelming probability; an adversary in this case may feasibly perform some limited jamming of messages.

In both constructions, we make use of bilinear maps, cryptographic techniques that have achieved much recently popularity as tools for reducing protocol interactivity [5]. In consequence, the security of our constructions is predicated on the Decisional Bilinear Diffie-Hellman assumption (DBDH) (see, e.g., [6]), as well as the random oracle assumption [4].

Organization In section 2, we explain the basic concepts of DC-net construction, and describe previous results on the topic. We present our formal model and other preliminary material in section 3. In section 4, we describe our short DC-net construction, followed in section 5 by presentation of our long DC-net proposal. We conclude in section 6. In the paper appendix, we offer security definitions and proofs for the protocols presented in the body of the paper.

2 Background

The intuition behind DC-nets is best introduced with a simple two-player example. Suppose that Alice and Bob possess k-bit messages m_A and m_B respectively. They wish to publish these messages anonymously, that is, in such a way that an observer cannot determine which player published which message. Suppose further that Alice and Bob share k-bit secret keys $k_{AB}(0)$ and $k_{AB}(1)$, as well as a secret, random bit b. Alice and Bob publish message pairs as follows:

if b = 0:	Alice:	$M_{A,0} = k_{AB}(0) \oplus m_A,$	$M_{A,1} = k_{AB}(1)$
	Bob:	$M_{B,0} = k_{AB}(0),$	$M_{B,1} = k_{AB}(1) \oplus m_B$
if b = 1:	Alice:	$M_{A,0} = k_{AB}(0),$	$M_{A,1} = k_{AB}(1) \oplus m_A$
	Bob:	$M_{B,0} = k_{AB}(0) \oplus m_B,$	$M_{B,1} = k_{AB}(1)$

An observer can compute $M_{A,0} \oplus M_{B,0}$ and $M_{A,1} \oplus M_{B,1}$, yielding the (unordered) message pair (m_A, m_B). The origin of these messages, however, remains unconditionally private: Without knowing the secrets shared by Alice and Bob, the observer cannot determine which player published which message. Observe that this protocol is non-interactive, in the sense that once their secrets are established, Alice and Bob need not communicate directly with one another.

This basic protocol may be extended to multiple players $P_1, P_2, \ldots, P_n$. Suppose that each pair of players (P_i, P_j) shares a set of keys $k_{i,j}(w)$ for $i, j, w \in \{1, 2, \ldots, n\}$, where $k_{i,j}(w) = k_{j,i}(w)$.

Each player P_i computes a vector of values as follows:

$$W_i = \{W_i(1) = \oplus_{j=1}^{n} k_{i,j}(1), W_i(2) = \oplus_{j=1}^{n} k_{i,j}(2), \ldots, W_i(n) = \oplus_{j=1}^{n} k_{i,j}(n)\}.$$

We refer to each message $W_i(w)$ as a *pad*, and refer to each value $k_{i,j}(w)$ as a *partial pad*. Observe that $\oplus_{i=1}^{n} W_i(w) = \mathbf{0}$, i.e., the pads in a given position w cancel when XORed together.

To broadcast messages in this scheme, each player P_i chooses a random position c_i and XORs her message m_i with the pad $W_i(c_i)$ in W_i. This yields a new vector $V_i = \{V_i(1), V_i(2) \ldots V_i(n)\}$ differing from W_i in the position c_i. Provided that all players have selected different positions c_i, the vector $V = \oplus_{i=1}^{n} V_i$ (i.e., the vector formed by XORing all messages in a given position), will consist of the set of messages posted by all players. Provided that keys and position selections $\{c_i\}$ are secret, the privacy of messages, i.e., the hiding of their originators, is unconditional.

As noted in Chaum's original paper, shared secrets may be established non-interactively via Diffie-Hellman key exchange, yielding computationally secure privacy.

A note on "collisions": Even when all players are honest, a problem arises in multi-player DC-nets in the selection of message positions $\{c_i\}$. In particular, there is no good non-interactive means of enabling all players to select distinct message positions. Hence, with some probability, two (or more) players will attempt to transmit messages in the same slot. In other words, players P_i and P_j will select $c_i = c_j$, so that the message $m_i \oplus m_j$ appears in the final vector V, rather than the individual messages. Some multi-round DC-net protocols address this problem via *reservation* procedure, whereby players request "slots" in advance. In all cases, however, DC-nets involve collisions, whether of messages themselves or reservation requests. (The problem can be avoided through techniques like secure multiparty computation of a secretly distributed permutation of slots among players, but this is impractical.)

We do not treat the issue of collisions in this paper, but simply regard a DC-net as a primitive that provides only partial throughput, i.e., drops some fraction of messages. Better throughput may be achieved by high-layer protocols, e.g., protocol repetition, either serially or in parallel.

2.1 Previous work

As already explained, a basic DC-net is subject to jamming by even a single dishonest player. Such a player P_i may simply set the vector V_i to a series of random pads. This effectively jams the DC-net: All elements in the final output V will be random and thus no messages will be successfully delivered. Worse still, the very privacy guarantees of the DC-net render it impossible to trace the source of the jamming in this case. Alternatively, an attacker may corrupt messages by tampering with bits in a valid vector V_i. It is on this security problem that the literature on DC-nets mainly focuses.

In his original paper [11], Chaum proposes the detection of dishonest players via a system of "traps" in a multi-round protocol. Prior to message transmission, a reservation protocol takes place in which players reserve future message slots. At this time, each player commits to a declaration of "trap" or "non-trap" for her reserved slot. To jam the DC-net, a dishonest player must transmit a message in a slot she has not reserved. But if she tries to transmit a message in a slot that is a "trap," then the attack may be detected during a decommitment phase.

An important follow-up result is that of Waidner and Pfitzman [21], who identify a weakness in this original protocol, and show that an attacker can feasibly strip the anonymity of honest players. (Improved reservation techniques in [2] and [22] reduce this possibility to some extent.) They propose a multi-round solution to this problem, also based on the idea of setting "traps" during a reservation phase. Like Chaum's protocol, theirs is only guaranteed to identify one dishonest player for a given "trap." No obvious method for fault recovery is available, apart from re-broadcasting. That said, it should be noted that the goal of this work is a little different than ours. While these researchers have sought to achieve unconditional untraceability assuming only honest point-to-point communication, our aim is to achieve privacy under only computational hardness assumptions.

Most recently, in [1], von Ahn, Bortz, and Hopper consider a constant-round anonymous-broadcast protocol that is essentially a DC-net variant (with an initial partitioning of players into autonomous groups). They accomplish the distribution of secrets for each protocol invocation via a secret-sharing protocol. In their scheme, the correctness of pads is proven via a cut-and-choose protocol. In the optimistic case, their protocol requires three broadcast rounds, and has $O(n^2)$ communications complexity (assuming a constant number of cut-and-choose invocations). In the presence of cheating players, the communications complexity rises to $O(n^4)$.

One problem with these previous protocols is that the computational and communications costs of catching cheating players with overwhelming probability is very high, requiring either many "traps" or many cut-and-choose invocations. This may not be problematic in cases where players may be reliably identified and where cheating carries a high penalty. For Internet systems, however, in which identities are not trustworthy, and participation in anonymous systems may be short-lived, even a small amount of cheating in the form of, e.g., tampering with messages, may be highly problematic. There is the risk that a savvy attacker may simply create false identities and then discard them when cheating is detected.

Our work is similar to the approach of von Ahn *et al.* in that we employ cryptographic proofs of correctness rather than "traps" in order to detect cheating. We employ a different strategy for pad computation, however, that has the benefit of more efficient proofs of correct pad computation. In particular, for our short DC-net proposal, in which players perform only a linear number of modular exponentiations (in n) on furnishing inputs, we show how to detect cheating with overwhelming probability. Another critical feature of our proposal is, of course, its non-interactivity in the optimistic case. Additionally, even in the presence of faults, our protocols may be completed in just two broadcast rounds, and with $O(n^2)$ communications complexity.

3 Preliminaries

For the sake of simplicity, we assume throughout this paper the presence of a reliable broadcast channel. As is well known, such a channel may be simulated via Byzantine agreement in a network with reliable point-to-point delivery. (See [7] for recent results in this area.) Another possible instantiation would be a Web server that performs the function of receiving and publishing messages in an honest and reliable manner. (Our constructions may also be employed in the presence of an unreliable broadcast channel provided that a given message is seen either by all players or by none. In this case, a dropped message may be modelled as a faulty player.) We further assume that all messages are authenticated, i.e., securely bound to the identities of their originators. In practice, this may be accomplished via digital signatures.

We define next the component functions of DC-nets. We denote the set of participants in the DC-net by $P = P_1, P_2, \ldots, P_n$. In what follows, when we specify a value as public or published, we assume it is transmitted to all players in P via an authenticated channel or entity. Setup is achieved by means of a parameter generation function $\mathsf{paramgen}$ and a key distribution function $\mathsf{keydist}$. These functions are similar to those employed in standard discrete-log-based distributed cryptographic protocols. They are called once at the beginning to set up long-lived parameters shared by all players. A difference here, however, is that we employ admissible bilinear maps as a basic tool in our constructions, and must therefore make use of elliptic-curve based algebraic groups accordingly. We assume the appropriate background on the part of the reader, and refer to [5] for further details and notation.

- **Parameter generation:** Taking security parameter l as input, the function $\mathsf{paramgen}$ outputs a quintuple $\rho = (p, G_1, G_2, \hat{e}, Q)$, where G_1 and G_2 are two groups of order p, Q is a generator of G_1 and $\hat{e} : G_1 \times G_1 \to G_2$ is an admissible bilinear map [5]. We require furthermore that the Decisional Bilinear Diffie-Hellman (DBDH) assumption holds for $\hat{e}$. Using the terminology of [5], the function $\mathsf{paramgen}$ is a parameter generator that satisfies the DBDH assumption. (For our "long" DC-net construction, we may weaken our hardness assumption to the Bilinear Diffie-Hellman problem (BDH), i.e., the computational variant, rather than the decisional one.) In practice, the map $\hat{e}$ may be instantiated with the Weil pairing over a suitable elliptic curve. The function $\mathsf{paramgen}$ may be executed by a trusted entity, which is our working assumption here. (Alternatively, it may be accompanied by a non-interactive proof of correct execution.) The quintuple ρ is published. We leave system parameters implicit in our notation where appropriate.

– **Key generation:** The function keydist takes as input the parameter specification ρ. It yields for each player P_i a private key $x_i \in_U \mathbb{Z}_p$ and a corresponding public key $y_i = x_i \cdot Q$. Each private key x_i is additionally shared among other players in a (k,n)-threshold manner. In particular, let f_i be a polynomial over $\mathbb{F}_p$ of degree $k-1$ selected uniformly at random such that $f_i(0) = x_i$. Player $P_j \in P$ receives from player P_i the private share $x_{i,j} = f_i(j)$, with a corresponding public share $y_{i,j} = x_{i,j} \cdot Q$. We assume that the function keygen is an algorithm executed by a trusted entity and a secure environment. (In practice, it may be instantiated by means of a distributed protocol; see [16] for an example of such a protocol and a discussion of the underlying security guarantees.)

We now describe the functions employed in the DC-net itself. We assume that players have access to a trustworthy global session counter s and specification $\Pi_s \subset P$ of players participating in the session s. Note that the privacy properties of our construction (defined in appendix A) do not rely upon publication of s or Π_s in a trustworthy manner, but the robustness does.

Posting: $(V_{i,s}, \sigma_{i,s}, i, s) \leftarrow \mathsf{post}(s, m_i, x_i)$; $[\Pi_s, \{y_j\}_{j \in P}]$.
The function post is invoked in session s by every player in Π_s. It returns to each player a set of outputs that hides that player's input, as well as auxiliary data that proves the correctness of the outputs. More precisely, the function post is a randomized function that takes as input the session counter s, a message m_i and the private key x_i of player P_i. Inputs to the function also include the set of players Π_s participating in the sessions and all public keys. For visual clarity, we set off the latter parameters in square brackets. We define $\pi_s = |\Pi_s|$ to be the number of participants in session s. The function post outputs:

– An output vector $V_{i,s} = \big(V_{i,s}(1), \ldots, V_{i,s}(\pi_s)\big)$. Let us denote the vector of random pads used by player P_i as $W_{i,s} = \big(W_{i,s}(1), \ldots, W_{i,s}(\pi_s)\big)$. The elements of the output vector and of the pad vector agree in all positions but one: the position c_i where message m_i is xored with the pad. In other words $V_{i,s}(w) = W_{i,s}(w)$ for all $w \neq c_i$ and $V_{i,s}(c_i) = m_i \oplus W_{i,s}(c_i)$
– Subsidiary data $\sigma_{i,s}$. The value $\sigma_{i,s}$ includes the identity of player P_i and a proof of valid formatting of the vector $V_{i,s}$.

Verification: $\{0,1\} \leftarrow \mathsf{verify}((V, \sigma), s, i, \Pi_s)$; $[\{y_j\}_{j \in P}]$.
The function verify determines the correctness of the vector V output by a given player P_i. When V is deemed correct, verify outputs '1'; otherwise it outputs '0'. This function can be called non-interactively by any player who wishes to verify the correctness of an output vector produced by another player.

Message extraction: $M \leftarrow \mathsf{extract}(\{V'_{i,s}\}_{i \in \Pi}, \Pi_s)$; $[\{y_j\}_{j \in P}]$.
Once all players in Π_s have posted their output vectors, it should be possible for any entity to extract the messages input to the mix procedure. We denote by extract the function that accomplishes this. The outputs of extract is a set M

of at most π_s distinct messages.

Pad reconstruction: $W_{i,s} \leftarrow \mathsf{reconstruct}(i, \Pi_s, \{x_{i,j}\}_{j \in \Pi_s})$.
If a player $P_i \in \Pi_s$ fails to produce a correct output vector (or any output at all), a quorum of other players in Π_s can reconstruct that missing output. We denote by reconstruct the function that accomplishes this.

We denote by $\mathsf{DC} = \{\mathsf{paramgen}, \mathsf{keydist}, \mathsf{post}, \mathsf{verify}, \mathsf{extract}, \mathsf{reconstruct}\}$ the complete set of functions constituting a DC-net.

4 Short DC-Net Protocol

4.1 Intuition and tools

In our first construction, the basic message unit is an algebraic group element. We would like to enable players to prove correct behavior in this setting with overwhelming probability. This combination of features leads to two basic problems:

Problem 1: We would like any given player P_i to be able to compute a partial pad $k_{i,j}(w)$ with any other player P_j in a non-interactive way. In fact, P_i must be able to compute many such partial pads non-interactively, namely one partial pad for every value w. Additionally, P_i must be able to prove the correctness of any partial pad $k_{i,j}(w)$ (or more precisely, of any pad, which is composed of partial pads).

The contradiction: Suppose that P_i computes partial pad $k_{i,j}(w)$ using a standard D-H protocol employing her own secret key x_i and the public key y_j of player P_j. (I.e., $k_{i,j}(w) = y_j^{x_i}$.) Since this computation is algebraic in form, P_i can efficiently prove statements in zero knowledge about $k_{i,j}(w)$. On the other hand, it is only possible to perform this D-H computation once, and P_i needs to do so for many different values of w! An alternative possibility is to hash $y_j^{x_i}$ with w to generate partial pad $k_{i,j}(w)$. In this case, though, there is no way to prove that $k_{i,j}(w)$ was correctly constructed with overwhelming probability without inefficient techniques like cut-and-choose or general secure function evaluation.

The solution: It is in resolving this problem that bilinear mapping comes into play.[3] It is possible to think of a bilinear map as a way of effecting a D-H exchange non-interactively across many different algebraic bases. In particular, P_i can compute the partial pad $k_{i,j}(w) = \hat{e}(y_j, x_i Q_w) = \hat{e}(Q, Q_w)^{x_i x_j}$, where Q_w

[3] There are other possible solutions to this problem without use of bilinear maps, e.g., changing keydist such that the sum $\sum_i x_i = 0 \bmod q$ becomes a special condition on private keys. This, however, would mean that in practice the protocol could never be efficiently realized by having players generate their own key pairs. Also, this type of solution would not work for the long DC-net construction.

is a randomly selected elliptic-curve point specific to w. We may thus think of P_i as performing a D-H exchange relative to a different algebraic base $\hat{e}(Q, Q_w)$ for every different value of w.

Problem 2: When a player P_i publishes a vector V of pads, she must prove its correctness. This means proving that every element of V is a correct pad – except the one element modified to contain the message m_i that P_i wants to publish. The problem here is that P_i of course does not wish to reveal *which* element of V contains the message m_i!

The solution: For each pad position w in her published vector, player P_i commits to a bit b_w. She lets $b_w = 0$ if the element in position w represents a correct pad, and $b_w = 1$ otherwise. P_i then proves two things:

1. For every position w, either the pad is correct OR the bit $b_w = 1$.
2. The sum $\sum_w b_w = 1$, i.e., the vector V contains at most one message.

To prove both of these facts, we use standard techniques for non-interactive proofs regarding statements involving discrete logs. We do so over the groups G_1 and G_2. As explored in many papers, these techniques permit honest-verifier zero-knowledge proof of knowledge of discrete logs [20], proof of equivalence of discrete logs [12], and first-order logical statements on such statements [9]. The proof protocols may be made non-interactive through use of the Fiat-Shamir heuristic [15]; they may be perfectly simulated with application of the random oracle model to the hash function used to generate challenges. We draw on the notation of Camenisch and Stadler [8] for a unified treatment and formal specification of these proofs in our detailed protocol. (E.g., $\mathrm{PoK}\{x : e = g^x \bigwedge f = h^x\}$ means a proof of knowledge that $\log_g e = \log_h f$, and is NIZK for our purposes.)

4.2 Protocol details

Parameter and key generation. The function $\mathsf{paramgen}$ outputs the set of parameters $\rho = (p, G_1, G_2, \hat{e}, Q)$. We also assume the existence of a hash functions $h : \{0,1\}^* \to G_1$ that is publicly known. The function $\mathsf{keydist}(\rho)$ then outputs a secret key $x_i \in \mathbb{Z}_p$ for each player P_i. Recall that shares of this secret key are distributed to other players and that all public keys are published.

Message posting. The pads $W_{i,s}(k)$ for player P_i in session s are computed as follows. We compute the point $Q_k = h(s||k)$ on G_1 and let

$$W_{i,s}(k) = \prod_{j \in \Pi_s; j \neq i} \hat{e}(Q_k, y_j)^{\delta_{i,j} x_i},$$

where $\delta_{i,j} = 1$ if $i < j$ and $\delta_{i,j} = -1$ if $j < i$. Player P_i then chooses at random a value $c_i \in \Pi_s$ and multiplies the message $m_i \in G_2$ with pad $W_{i,s}(c_i) \in G_2$ to produce the output vector $V_{i,s}$. We turn now to the computation of the auxiliary verification data $\sigma_{i,s}$:

1. Let g and h be two fixed random generators in a group G of order q for which the discrete logarithm problem is hard. Player P_i chooses independently at random n values $r_1, \ldots, r_n \in \mathbb{Z}_q$. For $1 \leq k \leq n$, where $k \neq c_i$, P_i computes $w_k = h^{r_k}$. P_i computes $w_{c_i} = gh^{r_{c_i}}$.
2. The prover proves knowledge of $\log_h(g^{-1} \prod_{i=0}^{n} w_i)$, i.e., PoK$\{r : \prod_{i=0}^{n} w_i/g = h^r\}$..
3. For $1 \leq k \leq \pi_s$, P_i proves the following statement:

$$\left(W_{i,s}(k) = \hat{e}(\prod y_j, Q_k)^{x_i} \textbf{ and } P_i \text{ knows } \log_h(w_k)\right) \quad \textbf{or} \quad \left(P_i \text{ knows} \log_h(w_k/g)\right)$$

i.e., PoK$\{x, r : (W_{i,s}(k) = \hat{e}(\prod y_j, Q_k)^x \bigwedge w_k = h^r) \bigvee (w_k/g = h^r)\}$.

The string $\sigma_{i,s}$ consists of all the values computed in steps 1 and 2 above. Finally, the function post outputs $(V_{i,s}, \sigma_{i,s}, i, s)$.

Verification. Anyone can verify non-interactively that the values computed in $\sigma_{i,s}$ are correct.

Message extraction. Given the π_s vectors $V_{1,s}, \ldots, V_{\pi_s,s}$ published by the players in Π_s, anyone can non-interactively compute $r_k = \prod_{i \in \Pi_s} V_{i,s}(k)$ for $k \in \Pi_s$. Recall that the definition of the pads is such that $\prod_{i \in \Pi_s} W_{i,s}(k) = 1$. We need now to introduce a notation for the subset of players who chose to publish their message in position k for a given k. For $k \in \Pi_s$, we denote $c^{-1}(k) = \{i \in \Pi_s \mid c_i = k\}$. Note that the subset $c^{-1}(k)$ could be empty, or contain a single or multiple players. Now it is clear that in every position k for which $c^{-1}(k)$ is a singleton $\{i\}$, we have $r_j = m_i$. All other messages m_i for which $c^{-1}(c_i)$ is not a singleton are unrecoverable in the output. The output of the function extract is the set of messages m_i which are recovered in the output.

Pad reconstruction. If a subset of players $\mathcal{P} \subseteq \Pi_s$ fail to publish their output vector, the remaining players can reconstruct the pads of missing players, and compute the output of the DC-net, as follows. Each player P_i for $i \notin \mathcal{P}$ publishes $x_{j,i} \cdot Q_k$ for all $j \in \mathcal{P}$. Anyone can verify the correctness of these values by checking that $\hat{e}(Q, x_{j,i}Q_k) = \hat{e}(y_{j,i}, Q_k)$. Furthermore, these values enable any player to recompute the pads of missing player P_j since $\hat{e}(Q_k, y_i)^{x_j}$ can be derived from the values $\hat{e}(x_{j,i}Q_k, y_i)$ by polynomial interpolation.

5 Long DC-Net Protocol

5.1 Intuition and tools

In order to obtain a "stretched" pad of the desired length in our long DC-net, it is necessary to apply a PRNG to a secret seed K, i.e., to use symmetric-key techniques. In consequence, proofs based on the algebraic structure of pads are no longer possible, and there are no efficient techniques for effecting proofs with overwhelming probability. Our use of symmetric-key techniques thus engenders two basic problems:

Problem 1: We face the same basic problem as in the short DC-net: It is necessary to prove correct construction of vectors without revealing where the messages are positioned. But the use of symmetric-key primitives means that we cannot benefit from the same NIZK proof techniques as in the short DC-net.

The solution: We resolve this problem by employing proof techniques that detect cheating players with high, but not overwhelming probability. In particular, we use a technique very similar to that of "randomized partial checking" [18] for mixnets. The idea is for a player P_i to prove correctness of her published vector V by generating a random challenge R non-interactively. This challenge R specifies a subset of half of the elements in the vector V. P_i reveals the underlying seeds for these as part of her proof. These seeds are derived essentially just like pads in the short DC-net. Thus, it is possible to provide a simple proof of correctness that may be non-interactively verified by other players.

One problem, of course, is that if P_i transmits a message m_i, then with probability $1/2$, the challenge R will lead to opening of the seed for the position containing that message. This problem may be resolved quite simply: P_i chooses challenges until she finds one that does not lead to opening of the seed for the message position. Some tens of attempts will permit this with overwhelming probability.

Since only half of the seeds are revealed, some number of invalid pads can escape detection. In particular, for a given challenge, any seed will be revealed with probability $1/2$. Hence, given u invalid pads, an adversary must perform work roughly 2^u to compute a challenge R that does not reveal cheating. In practice, therefore, we would expect an adversarial player to be unable to insert more than, say, 80 invalid pads into a vector. Thus such a player can "jam" only a limited number of slots. Assuming large enough vectors and adversarial control of a small enough set of players, the throughput for the DC-net remains fairly high.

Thus, our proof protocol is as follows. Let h be a hash function from $\{0,1\}^*$ to $\mathbb{Z}_n$ (modelled in our proof as a random oracle).

1. The player chooses a random seed r and computes $h(V||r||1), h(V||r||2), \ldots$ until all these values form a subset $S \subset \{1, \ldots, n\}$ of size $|S| = n/2$. Note that $i \neq j$ does not imply $h(V||r||i) \neq h(V||r||j)$ so that more than $n/2$ computations may be required to obtain the set S.
2. If $i_0 \in S$, the set S is discarded. The prover returns to step 1 and chooses a new random seed. Step 1 is successful on average after 2 tries.
3. Otherwise, the protocol outputs the random seed r and the set S. For all $j \in S$, the protocol also outputs the secret key k_j.
4. The verifier verifies that the set S is correctly computed from randomness r. For all $j \in S$, the verifier uses the key k_j to verify the correctness of V_j.

Problem 2: Since the seeds used to compute pads in our long DC-net assume the same form as those in the short DC-net, the reconstruction procedure is very similar. The only difference in the process is that once a seed is recovered, the

PRNG must be applied to obtain the corresponding pad. What we highlight, however, is that our use of bilinear maps is solving a fundamental problem in the long DC-net construction.

In the short DC-net, honest players could, in principle, make do without using bilinear maps. Indeed, they can reconstruct a pad in a verifiable way without revealing any long term secrets, by exploiting the algebraic structure of pads. (As explained in the footnote above, it is possible in principle to have, for example, secret keys $\{x_i\}$ that cancel, i.e., such that $\sum_i x_i = 0 \bmod q$, thereby engendering pads that "cancel." Note that this results in a very cumbersome key setup.) In the case of long DC-nets, however, there is no good way to do this. Briefly stated, the application of the PRNG eliminates algebraic structure on the pads.

The only way, therefore, to achieve "cancellation" of pads in a long DC-net, is for pairs of players to share secrets. But as already noted, in a standard setup without bilinear maps, it is possible for a pair of players (P_i, P_j) to establish a shared secret S non-interactively *only once* through application of D-H to their public keys. This secret S can be used to generate new secrets for multiple sessions through application of symmetric-key primitives, e.g., secrets may be generated as $h(S,1), h(S,2), \ldots$. But without expensive general techniques, there is no way to reconstruct a given secret $h(S,w)$ without revealing S itself and consequently compromising *all* shared secrets between P_i and P_j.

The solution: This is where bilinear maps are helpful. As explained above, the intuition is that for a single pair of public keys, a bilinear map may be thought of as permitting non-interactive D-H key establishment across many different algebraic bases. Thus, each seed may be reconstructed individually by honest players holding shares of the private keys of P_i and P_j. Under the (Bilinear) Diffie-Hellman assumption, this may be accomplished without compromising the privacy of other seeds. (In algebraic terms, one seed might assume the form $S_1 = {g_1}^{x_i x_j}$, while another assumes the form $S_2 = {g_2}^{x_i x_j}$. Provided that g_1 and g_2 are random, knowledge of S_1 does not permit computation of S_2.)

5.2 Protocol details

In this section, we define our long DC-net protocol and highlight the differences with the short DC-net. The main differences between the long and short schemes lie in the definition of the auxiliary data $\sigma_{i,s}$ and the verification algorithm.

Parameter and key generation. This step is nearly identical to the short protocol. The function $\mathsf{paramgen}$ outputs parameters $\rho = (p, G_1, G_2, \hat{e}, Q)$. As in the short protocol, we use a hash functions $h : \{0,1\}^* \to G_1$. We also assume the existence of a publicly known pseudo-random number generator $f : G_2 \to \{0,1\}^l$, where l is the length in bits of messages processed by the long DC-net. (For the purposes of our proofs, we model this as a random oracle.) The function $\mathsf{keydist}(\rho)$ distributes keys to all players.

Message posting. Recall that we define the point $Q_k = h(s||k)$ on G_1. The pads $W_{i,s}(k)$ for player P_i in session s are computed as follows:

$$W_{i,s}(k) = \oplus_{j \in \Pi_s; j \neq i} \ f\big(\hat{e}(Q_k, y_j)^{x_i}\big)$$

Recall that player P_i then chooses at random a value $c_i \in \Pi_s$ and XORs the message m_i with pad $W_{i,s}(c_i)$ to produce the output vector $V_{i,s}$. We turn now to the computation of the auxiliary verification data $\sigma_{i,s}$:

1. Recall that the number of participants in session s is denoted π_s. Let φ be a hash function from $\{0,1\}^*$ to $\mathbb{Z}_{\pi_s}$. Using φ and a random value r, the player P_i computes a subset $S \subset \{1, \ldots, \pi_s\}$ of size $\pi_s/2$ such that $c_i \notin S$.
2. For all $j \in S$, P_i proves that the value $V_{i,s}(j)$ is computed correctly by revealing $x_i Q_j$.

The string $\sigma_{i,s}$ consists of the values computed in steps 1 and 2 above. Finally, the function post outputs $(V_{i,s}, \sigma_{i,s}, i, s)$.

Verification. Anyone can verify non-interactively that the values computed in $\sigma_{i,s}$ are correct.

Message extraction. Given the π_s vectors $V_{1,s}, \ldots, V_{\pi_s,s}$ published by the players in Π_s, anyone can non-interactively compute $r_k = \oplus_{i \in \Pi_s} V_{i,s}(k)$ for $k \in \Pi_s$. Recall that the definition of the pads is such that $\oplus_{i \in \Pi_s} W_{i,s}(k) = 0$. Using the same notations as in the short protocol, it is clear that $r_k = \oplus_{i \in c^{-1}(k)} \ m_i$. In other words, in every position k for which $c^{-1}(k)$ is a singleton $\{i\}$, we have $r_j = m_i$. All other messages m_i for which $c^{-1}(c_i)$ is not a singleton are unrecoverable in the output. The output of the function extract is the set of messages m_i which are recovered in the output.

Pad reconstruction. If a subset of players $\mathcal{P} \subseteq \Pi_s$ fail to publish their output vector, the remaining players can reconstruct the pads of missing players, and compute the output of the DC-net, as follows. Each player P_i for $i \notin \mathcal{P}$ publishes $x_{j,i} \cdot Q_k$ for all $j \in \mathcal{P}$. Anyone can verify the correctness of these values by checking that $\hat{e}(Q, x_{j,i} Q_k) = \hat{e}(y_{j,i}, Q_k)$. Furthermore, these values enable any player to recompute the seeds of missing player P_j since the value $\hat{e}(Q_k, y_i)^{x_j}$ can be computed from the values $\hat{e}(x_{j,i} Q_k, y_i)$ by polynomial interpolation. The pads themselves may then be computed through application of f.

6 Conclusion

We have proposed two new DC-net constructions. Unlike previous DC-net proposals, our constructions allow for efficient detection and identification of cheating players with high probability. When cheating is detected, a single additional broadcast round enables full fault recovery. Our DC-net protocols are thus resilient to the jamming attacks that negated the simplicity and non-interactivity of earlier DC-net proposals.

In the appendix, we define a formal model in which we prove the privacy and correctness of our constructions. We observe that our comparatively simple definitions and proofs are made possible by the non-interactivity of DC-nets.

References

1. L. von Ahn, A. Bortz and N. J. Hopper. k-anonymous message transmission. In *Proc. of ACM CCS '03*, pp. 122-130. ACM Press, 2003.
2. J. Bos and B. den Boer. Detection of disrupters in the DC protocol. In *Proc. of Eurocrypt '89*, pp. 320-327. LNCS 434.
3. P. Boucher, A. Shostack, and I. Goldberg. Freedom Systems 2.0 architecture. Zero Knowledge Systems, Inc. White Paper, December 2000. Available at http://freehaven.net/anonbib/
4. M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proc. of ACM CCS '93*, pp. 62-73. ACM Press, 1993.
5. D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. In *SIAM J. of Computing*, 32(3), pp. 586-615, 2003.
6. R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In *Proc. of Eurocrypt '03*, pp. 255-271. LNCS 2656.
7. C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In *Proc. of Crypto '01*, pp. 524-541. LNCS 2139.
8. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *Proc. of Crypto '97*, pp. 410-424. LNCS 740.
9. R. Cramer, I. Damgaard, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Proc. of Crypto '94*, pp. 174-187. LNCS 839.
10. D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. In *Communications of the ACM*, 24(2), pp. 84-88, 1981.
11. D. Chaum. The dining cryptographers problem: unconditional sender and recipient untraceability. In *Journal of Cryptology*, 1(1), pp. 65-75, 1988.
12. D. Chaum and H. van Antwerpen. Undeniable signatures. In *Proc. of Crypto '89*, pp. 212-216. LNCS 435.
13. G. Danezis, R. Dingledine and N. Mathewson. Mixminion: design of a type III anonymous remailer protocol. In *IEEE Symposium on Security and Privacy 2003*, pp. 2-15.
14. R. Dingledine. Anonymity bibliography. Available on the web at http://freehaven.net/anonbib/
15. A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proc. of Crypto '86*, pp. 186-194. LNCS 263.
16. R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Proc. of Eurocrypt '99*, pp. 295-310. LNCS 1592.
17. D. Goldschlag, M. Reed and P. Syverson. Onion routing. In *Communications of the ACM*, 42(2), pp. 39-41. 1999.
18. M. Jakobsson, A. Juels and R. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *Proc of USENIX'02.*
19. M. Reiter and A. Rubin. Anonymous web transactions with Crowds. In *Communications of the ACM*, 42(2), pp. 32-38. 1999.

20. C. Schnorr. Efficient signature generation by smart cards. In *Journal of Cryptology*, 4(3), pp. 161-174. 1991.
21. M. Waidner and B. Pfitzmann. The Dining Cryptographers in the disco: unconditional sender and recipient untraceability with computationally secure serviceability. In *Proc. of Eurocrypt '89*, p. 690. LNCS 434.
22. M. Waidner. Unconditional sender and recipient untraceability in spite of active attacks. In *Proc. of Eurocrypt'89*, pp. 302-319. LNCS 434.

A Security Definitions

A.1 Privacy

We consider a static adversary $\mathcal{A}$ capable of actively corrupting a set $P_{\mathcal{A}}$ of fewer than $n/2$ players in P. We regard a mix network DC as private if $\mathcal{A}$ is unable to determine the origin of any message input by an honest player with probability significantly better than a random guess. We capture this concept by way of an experiment in which $\mathcal{A}$ selects just two honest players p_0 and p_1 as targets. The adversary may also choose a pair of plaintexts (m_0, m_1) to serve as inputs for these two target players. The plaintexts are randomly assigned to p_0 and p_1; the task of $\mathcal{A}$ is to guess this assignment.

We let $\tilde{\mathsf{post}}_i(\cdot,\cdot,\cdot)$ denote an oracle that posts a message on behalf of player P_i. The adversary may specify s, m and Π_s. The oracle is assumed to have access to the private keys of P_i. The adversary may not invoke a given oracle twice on the same session identifier s. (In a real-world protocol, this restriction is easily enforced through use of a local counter.)

The oracle $\tilde{\mathsf{post}}_i(\cdot,\cdot,\cdot)$ also produces auxiliary data $\sigma_{i,s}$. A small difficulty arises in the long protocol, where $\sigma_{i,s}$ reveals half the pads of player P_i. If the pads of all honest players are revealed in the positions where p_0 and p_1 posted m_0 and m_1, then $\mathcal{A}$ can trivially determine which player posted which message. This happens with low probability if the number of honest players is large. In our privacy experiment, we assume that the auxiliary data does not reveal the pads used by p_0 and p_1 in the positions where they posted m_0 and m_1.

The oracle $\tilde{\mathsf{post}}^*(\cdot,\cdot,\cdot,\cdot)$ is a special oracle call that causes the two targeted players to post the chosen messages (m_0, m_1). In particular, this oracle call is equivalent to two successive oracle calls: $\tilde{\mathsf{post}}_{p_0}(m_b, s, \Pi_s, \cdot)$ and $\tilde{\mathsf{post}}_{p_1}(m_{1-b}, s, \Pi_s, \cdot)$, where $p_0, p_1 \in \Pi_s$.

We let $\tilde{\mathsf{reconstruct}}_i(\cdot,\cdot)$ denote an oracle that returns the reconstructed pad of player P_i. The adversary may specify the session s and Π_s. The oracle is assumed to have access to the private key held by P_i. The oracle $\tilde{\mathsf{reconstruct}}_i$ may be called by $\mathcal{A}$ at any point during the experiment, with the following restriction: $\mathcal{A}$ may not call $\tilde{\mathsf{reconstruct}}_{p_0}$ or $\tilde{\mathsf{reconstruct}}_{p_1}$ for the session s in which $\mathcal{A}$ chose to call the special oracle $\tilde{\mathsf{post}}^*$. This restriction is natural: it simply states that $\mathcal{A}$ is not allowed to ask for the pads of players p_0 and p_1 in the session in which it must guess the assignment of messages m_0, m_1 to p_0, p_1.

We let $\in_U$ denote uniform, random selection from a set. Security parameters are left implicit.

Experiment $\mathbf{Exp}_{\mathcal{A}}^{priv}(\mathsf{DC}); [k, n, l]$
  $\mathsf{paramgen}(l); \mathsf{keydist};$
  $P_{\mathcal{A}} \leftarrow \mathcal{A}(\{PK_i\});$
  $(m_0, m_1, p_0, p_1) \leftarrow \mathcal{A}^{\{\tilde{\mathsf{post}}_i(\cdot,\cdot,\cdot,\cdot)\}_{i \in P - P_{\mathcal{A}}}, \tilde{\mathsf{reconstruct}}_i(\cdot,\cdot)};$
  $b \in_U \{0, 1\};$
  $b' \leftarrow \mathcal{A}^{\{\tilde{\mathsf{post}}_i(\cdot,\cdot,\cdot,\cdot)\}_{i \in P - P_{\mathcal{A}}}, \tilde{\mathsf{reconstruct}}_i(\cdot,\cdot), \tilde{\mathsf{post}}^*(\cdot,\cdot,\cdot,\cdot)};$
  if $b' = b$ output '1' else output '0';

We define the advantage of $\mathcal{A}$ in this experiment as

$$\mathbf{Adv}_{\mathcal{A}}^{priv}(\mathsf{DC}); [k, n, l] = \mathsf{pr}[\mathbf{Exp}_{\mathcal{A}}^{priv}(\mathsf{DC}); [k, n, l] = \text{`1'}] - 1/2 \, .$$

We say that our scheme is private if this advantage is negligible for all adversaries $\mathcal{A}$ with polynomial running time (where the quantities are defined asymptotically with respect to l in the usual manner). The following propositions show that our short and long DC-nets are private. (The proofs are in appendix B.)

Proposition 1. *The short DC-net protocol of section 4 is private if the Decisional Bilinear Diffie-Hellman (DBDH) problem is hard in the group G_1.*

Proposition 2. *The long DC-net protocol of section 5 is private if the Bilinear Diffie-Hellman (BDH) problem is hard in the group G_1.*

Remark: the non-interactivity of the mix network DC makes possible this relatively simple definition of privacy. In a mix network involving interaction among players, an adversary can change the behavior of honest players by inducing errors or failures in the outputs of corrupted players. The resulting broad scope of adversarial behavior induces considerably more complex privacy definitions.

A.2 Correctness

We define correctness in terms of the ability of a corrupted player P_i to post a vector V that has an invalid format, but is accepted by the function verify. Invalid formatting may mean that V includes incorrectly computed pads or, alternatively, that V contains an inadmissibly large number of messages. More formally, we deem a vector V as correct if it constitutes a valid output of post for the private key of the correct player. (Other definitions are possible.) We use the triangular brackets '$\langle\ \rangle$' to denote the set of possible function outputs.

Experiment $\mathbf{Exp}_{\mathcal{A}}^{corr}(\mathsf{DC}); [k, n, l]$
  $\mathsf{paramgen}(l); \mathsf{keydist};$
  $P_{\mathcal{A}} \leftarrow \mathcal{A}(\{PK_i\}); ((V, \sigma, i, s), \Pi_s) \leftarrow \mathcal{A};$
  if $(V, \sigma, i, s) \notin \langle \mathsf{post}(s, m, x_i); [\Pi_s, \{y_j\}_{j \in P}] \rangle$ for any m and
    $\mathsf{verify}((V, \sigma), s, i, \Pi_s); [\{y_j\}_{j \in P}] =$ '1' then output '1';
  else output '0';

We define the advantage of $\mathcal{A}$ in this experiment as $\mathbf{Adv}_{\mathcal{A}}^{priv}(\mathsf{DC}); [k, n, l] = \mathsf{pr}[\mathbf{Exp}_{\mathcal{A}}^{corr}(\mathsf{DC}); [k, n, l] = \text{`1'}]$. We regard our scheme as providing correctness if for all adversaries $\mathcal{A}$ with polynomial running time, this advantage is negligible.

Proposition 3. *The short DC-net protocol of section 4 is correct.*

Proposition 4. *The long DC-net protocol of section 5 satisfies a weaker property. If an adversary submits an output in which k pads out of n are incorrectly computed, the probability that* verify *accepts this output is 2^{-k}.*

B Proofs of Privacy

Proposition 1. *The short DC-net protocol of section 4 is private if the Decisional Bilinear Diffie-Hellman (DBDH) problem is hard in the group G_1.*

Proof. Let $\mathcal{A}$ be a polynomial-time adversary who wins $\mathbf{Exp}_{\mathcal{A}}^{priv}(\mathsf{DC})$ with non-negligible advantage ϵ. We use $\mathcal{A}$ to solve DBDH challenges with non-negligible advantage as follows. We first call $\mathsf{paramgen}(l)$ to get parameters $(p, G_1, G_2, \hat{e}, Q)$ where G_1 and G_2 are groups of order p, Q is a generator of G_1 and $\hat{e} : G_1 \times G_1 \rightarrow G_2$ is an admissible bilinear map. Let (aQ, bQ, cQ, dQ) be a DBDH challenge in G_1 (the challenge is to determine whether $d = abc$ or d is random).

We give $\mathcal{A}$ the output of $\mathsf{paramgen}(l)$. Next we simulate $\mathsf{keydist}$ for $\mathcal{A}$. We let the public keys of two players (say P_1 and P_2) be $y_1 = aQ$ and $y_2 = bQ$. For every other player, we choose a private key $x_i \in_U \mathbb{Z}_p$ and compute the corresponding public key $y_i = x_i \cdot Q$. Given all these public keys, $\mathcal{A}$ returns the set $P_{\mathcal{A}}$ of players it controls. If $P_1 \in P_{\mathcal{A}}$ or $P_2 \in P_{\mathcal{A}}$, we abort. Otherwise, we give $\mathcal{A}$ the private keys of all the players in $P_{\mathcal{A}}$. We also give $\mathcal{A}$ the shares of the private keys held by all the players in $P_{\mathcal{A}}$. For the private key of player P_1 and P_2, which we do not know, we generate random shares.

$\mathcal{A}$ can then call the oracle $\tilde{\mathsf{post}}_i(\cdot,\cdot,\cdot,\cdot)$ any number of times for $i \in P - P_{\mathcal{A}}$. For all but one session for which $\mathcal{A}$ calls $\tilde{\mathsf{post}}$, we let $h(s||k) = r_{s,k}Q$, where the values $r_{s,k} \in_U \mathbb{Z}_p$. For one session s_0, we define $h(s_0||k)$ differently. We choose 2 "special" positions $k_0, k_1 \in_U \{1, \ldots, n\}$ as well as $R \in_U \mathbb{Z}_p$. We define $h(s_0||k_0) = cQ$, $h(s_0||k_1) = RcQ$ and for $k \notin \{k_0, k_1\}$, we let $h(s_0||k) = r_{s_0,k}Q$ for values $r_{s_0,k}$ chosen at random in $\mathbb{Z}_p$.

To simulate $\tilde{\mathsf{post}}_i(\cdot,\cdot,\cdot,\cdot)$ for $\mathcal{A}$ in session s, we need the pads $W_{i,s}(k) = \prod_{j \in \Pi_s; j \neq i} \hat{e}(Q_k, y_j)^{\delta_{i,j} x_i}$, where $Q_k = h(s||k)$. For all session $s \neq s_0$, we have $Q_k = h(s||k) = r_{s,k}Q$ and therefore we can compute the pad $W_{i,s}(k)$ for all players P_i (even for P_1, P_2) using the equality $\hat{e}(Q_k, y_j)^{x_i} = \hat{e}(y_i, y_j)^{r_{s,k}}$. For session s_0, we can compute the pads of all players except P_1 and P_2 whose private key we do not know. If $\mathcal{A}$ calls $\tilde{\mathsf{post}}$ for P_1 or P_2 in session s_0, we abort.

Note that knowledge of the pads also enables us to simulate the auxiliary data $\sigma_{i,s}$ in both the short and the long protocol, as well as the oracle $\tilde{\mathsf{reconstruct}}_i$.

$\mathcal{A}$ then chooses two messages m_0, m_1 to be posted by two players p_0, p_1 of the adversary's choice. If $(p_0, p_1) \neq (P_1, P_2)$, we abort the simulation. $\mathcal{A}$ may again call $\tilde{\mathsf{post}}_i$ and we simulate that oracle as before.

Finally, $\mathcal{A}$ calls $\tilde{\mathsf{post}}^*$ for a particular session. If that session is not s_0, we abort. Otherwise, we simulate $\tilde{\mathsf{post}}^*$ as follows. For P_1, we define the pads:

$$W_{1,s_0}(k_0) = \hat{e}(Q, dQ)^{\delta_{1,2}} \prod_{3 \leq j \leq n} \hat{e}(cQ, y_1)^{\delta_{1,j} x_j},$$

$$W_{1,s_0}(k_1) = \hat{e}(Q, dQ)^{\delta_{1,2}} \prod_{3 \le j \le n} \hat{e}(RcQ, y_1)^{\delta_{1,j} x_j},$$

$$W_{1,s_0}(k) = \prod_{2 \le j \le n} \hat{e}(y_1, y_j)^{\delta_{1,j} r_{s_0,k}} \quad \text{for} \quad k \notin \{k_0, k_1\}$$

We define the pads for P_2 similarly. We choose a bit b at random and let P_1 post m_b in position k_1 and P_2 post m_{1-b} in position k_2. We simulate the corresponding NIZK proofs for the auxiliary data using standard techniques by allowing the simulator to set random oracle responses before making commitments.

$\mathcal{A}$ outputs a guess b'. If $b' = b$, we guess that (aQ, bQ, cQ, dQ) is a DBDH tuple, and otherwise that it is not. It remains to show that our guess is correct with non-negligible advantage:

- When $d = abc$, by definition of $\mathcal{A}$, we have $b' = b$ with advantage ϵ.
- When $d \neq abc$, our simulation of the pads $W_{1,s_0}(k_0), W_{1,s_0}(k_1), W_{2,s_0}(k_0)$ and $W_{2,s_0}(k_1)$ was incorrect. There is consequently no way for $\mathcal{A}$ to distinguish between respective partial pads for P_1 and P_2 of the form $(V_1, V_2) = (Rand \oplus m_1, Rand)$ and $(V_1, V_2) = (Rand, Rand \oplus m_2)$, because they are identically distributed (here, $Rand$ denotes random values). In other words, $\mathcal{A}$ can't possibly guess the bit b with non-negligible advantage.

This shows that when the simulation does not abort, $\mathcal{A}$ solves DBDH challenges with advantage $\epsilon/2$. The probability that the simulation does not abort is greater than a value that is polynomial in the security parameter. Overall, we have used $\mathcal{A}$ to solve DBDH challenges with non-negligible advantage. □

Proposition 2. *The long DC-net protocol of section 5 is private if the Bilinear Diffie-Hellman (BDH) problem is hard in the group G_1.*

Proof. The proof is similar to that of Proposition 1. Let $\mathcal{A}$ be a polynomial-time adversary who wins $\mathbf{Exp}^{priv}_{\mathcal{A}}(\mathsf{DC})$ with non-negligible advantage ϵ and let (aQ, bQ, cQ) be a BDH challenge (the challenge is to compute dQ, where $d = abc$). We embed the BDH challenge as before. The difference worth noting is that the output of the bilinear function, in the long protocol, is expanded with a PRNG f. We model f as a random oracle. There are two possible distributions for the simulator: distribution D, where the simulator calls $f(dQ)$ (for the correct BDH value d), and distribution $\tilde{D}$, where the simulator uses a random value. $\mathcal{A}$ cannot distinguish between D and $\tilde{D}$ unless it calls f on input dQ.

If $\mathcal{A}$ cannot distinguish D from D', it cannot distinguish a real-world protocol invocation from one in which random pads are used and therefore cannot learn anything about which player posted which message. $\mathcal{A}$ then must be able to distinguish D from D' and so must call the random oracle on input dQ occasionally. We answer the BDH challenge with one of $\mathcal{A}$'s calls to the random oracle and win with non-negligible probability since $\mathcal{A}$ is polynomially bounded. □