Efficient Anonymity-Preserving Data Collection

Justin Brickell and Vitaly Shmatikov
Department of Computer Sciences
The University of Texas at Austin
Austin, TX, USA
jlbrick@cs.utexas.edu, shmat@cs.utexas.edu

ABSTRACT

The output of a data mining algorithm is only as good as its inputs, and individuals are often unwilling to provide accurate data about sensitive topics such as medical history and personal finance. Individuals may be willing to share their data, but only if they are assured that it will be used in an aggregate study and that it cannot be linked back to them. Protocols for anonymity-preserving data collection provide this assurance, in the absence of trusted parties, by allowing a set of mutually distrustful respondents to anonymously contribute data to an untrusted data miner.

To effectively provide anonymity, a data collection protocol must be *collusion resistant*, which means that even if all dishonest respondents collude with a dishonest data miner in an attempt to learn the associations between honest respondents and their responses, they will be unable to do so. To achieve collusion resistance, previously proposed protocols for anonymity-preserving data collection have quadratically many communication rounds in the number of respondents, and employ (sometimes incorrectly) complicated cryptographic techniques such as zero-knowledge proofs.

We describe a new protocol for anonymity-preserving, collusion resistant data collection. Our protocol has linearly many communication rounds, and achieves collusion resistance without relying on zero-knowledge proofs. This makes it especially suitable for data mining scenarios with a large number of respondents.

Categories and Subject Descriptors

E.3 [**Data**]: Data Encryption; H.2.8 [**Information Systems**]: Database Applications—*Data Mining*

General Terms

Algorithms, Security

Keywords

Anonymity, Privacy, Data Mining

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
KDD'06, August 20–23, 2006, Philadelphia, Pennsylvania, USA.
Copyright 2006 ACM 1-59593-339-5/06/0008 ...$5.00.

1. INTRODUCTION

Consider a scenario in which a data miner wishes to collect data from a large set of respondents for use in a data mining experiment. The miner queries each respondent, who in turn transmits her response back to the miner. If the respondents are willing to truthfully answer the miner's query, then the miner is able to proceed with his experiment. In some cases, however, a respondent's willingness to answer truthfully is dependent on a guarantee that her answer will be used only in the aggregate and cannot be linked back to her. For example, the miner may be conducting a survey on illegal activities, or on sensitive medical conditions. If the miner can convince the respondent that her response will be anonymous among her peer respondents then she will participate truthfully in the survey; otherwise, she will not.

Data collected with the simple query/response protocol described above will not be anonymous, because the data miner can easily observe which respondent transmits which response. One way to achieve anonymity would be to *shuffle* the responses, so that the miner receives the responses in a random order. This is easy to do with the assistance of a trusted third-party shuffler, as the respondents can submit their responses to the shuffler, who collects all responses and then forwards them to the miner in a random order.

The goal of this research is to achieve the same security guarantees without the need for trusted parties. We give a protocol that allows mutually distrustful respondents to submit their responses to an untrusted data miner in a manner which guarantees that their responses will be received by the miner, but that the probability that the miner can link a response to a respondent is essentially no better than random guessing. Furthermore, our protocol is *collusion resistant*. Even if all malicious respondents freely share information with the malicious miner, they will be unable to learn the associations between honest respondents and their responses. This strong form of collusion resistance is important in online data collection scenarios where respondents cannot communicate directly with one another, and therefore are unable to determine whether other respondents are genuine participants, or shills set up by a malicious miner.

Our protocol consists of two parts. In the first part, respondents encrypt their responses and shuffle them under encryption, so that it will be impossible to determine which encrypted response belongs to which respondent. In the second part, the integrity of the shuffle is verified, and the respondents provide information to the data miner so that he can decrypt the responses.

In order to be practical for use in data mining applications with large numbers of respondents, protocols for anonymous data collection need to be efficient as well as secure. In the online data collection scenario, it is especially important to limit the number of messages that must be transmitted between the data miner and the respondents. Our protocol requires that the data miner send and receive only $O(N)$ messages when there are N respondents. Furthermore, our protocol does not rely on zero-knowledge proofs. The most efficient currently known zero-knowledge proofs for verifiable shuffles [13, 15] require 7 rounds of communication for each proof. Moreover, it is unclear whether these proofs preserve their properties when composed in parallel. Therefore, if multiple proofs are needed at any step of the protocol, they have to be carried sequentially, rendering the communication complexity of the protocol impractical.

Related Work

The problem of collecting data so that the miner is unable to link any honest respondent to her response was investigated by Yang *et al.* [21]. Their solution is to choose t of N respondents as "leaders," and have respondents encrypt their responses with the leaders' public keys. Each of the leaders shuffles and rerandomizes the ciphertexts, proving to every respondent in zero-knowledge that the shuffle has been carried out correctly. This protocol requires $O(t^2)$ zero-knowledge proofs, each of which involves several rounds of communication. (To achieve the same anonymity guarantees as our protocol, t should be equal to N, in which case communication complexity is quadratic in the number of respondents.) The protocol of Yang *et al.* does not achieve collusion resistance because if the last leader is corrupt and colludes with the data miner, they can break the anonymity of all honest respondents. This attack is due to an incorrect use of zero-knowledge proofs, and is explained in detail in Appendix A.

A related problem is secure multiparty computation, in which several parties wish to compute a function on their joint input, but without revealing their input to one another. There are generic protocols [22, 11] that allow any polynomial functionality to be computed securely in the semi-honest [9] model, but they are too inefficient for practical data mining purposes. Several papers [14, 5, 20] have given efficient special-purpose protocols for the computation of particular data mining functionalities. The scenario in this paper is different from a secure multiparty computation scenario, because the respondents *do* want to reveal their inputs to the data miner. Our security property is *unlinkability* rather than secrecy of inputs, that is, we want to prevent the data miner from learning which input came from which respondent.

Instead of submitting their responses anonymously within a peer group, respondents submitting sensitive data might randomly perturb their responses. In this case, privacy would be based on the inexactness of the responses, rather than on the lack of association between respondent and response. Several methods for random perturbation have been proposed and their applicability to data mining problems investigated [2, 1]. In all perturbation-based privacy methods, there is an inherent tradeoff. The more perturbed the data are, the more privacy is guaranteed, but the less useful the collected data are for data mining applications. By contrast, our techniques provide the data miner with exact, unperturbed responses.

Respondents submitting data using *randomized response* lie with a certain probability, so that the data miner is never certain of the truthfulness of a sensitive response. Data mining algorithms must be modified to accept randomized response data. For instance, Du and Zhan [8] modify the popular ID3 decision tree classification algorithm. By contrast, after a successful execution of our algorithm the data miner knows the exact responses and can use any unmodified data mining algorithm.

Research on k-anonymization [17, 16] observes that in some cases the information contained in the anonymous response may be specific enough to identify the respondent if the malicious data miner has access to auxiliary information such as a voter or census database. This problem is orthogonal to the problem investigated in this paper. If responses and respondents are linkable by content, then no amount of shuffling in an anonymity-preserving data collection protocol will make them unlinkable. Likewise, if responses and respondents are linkable by the collection procedure, then no amount of k-anonymization will make them unlinkable. In this paper we assume that responses and respondents are *not* linkable by content.

Finally, so-called *mix networks* such as onion routing [18, 7] have been proposed to enable anonymous communications on public networks. Mix networks aim to hide the identities of message senders, and thus seem to be a poor match for data mining scenarios, where the data miner may need to know exactly who the respondents are. Using a mix network to collect responses while ensuring that respondents are members of a well-defined set and that each respondent contributes no more than one response requires complicated cryptographic techniques such as group signatures, and is unlikely to be efficient for practical use. Also, because of the possibility that one or more network nodes may be compromised, mix networks provide only probabilistic anonymity guarantees, whereas our goal in this paper is cryptographically strong anonymity.

2. PRELIMINARIES

In this section we define cryptographic concepts and notation that are used throughout the remainder of the paper.

2.1 Public-Key Encryption

We take the standard definition of a public-key cryptosystem from [3]. A public-key (or "asymmetric") encryption scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms, as follows:

- A randomized *key generation* algorithm $\mathcal{K}$ which returns a pair (x, y) of keys. These are the private and public keys, respectively.
- A (possibly randomized) *encryption* algorithm $\mathcal{E}$, which takes a public key y and a plaintext m, and returns a ciphertext C. To denote encryption we will write
$$C = \{m\}_y.$$
- A deterministic *decryption* algorithm $\mathcal{D}$, which takes a private key x and a ciphertext C to return a message m. We will denote this as
$$m = \mathrm{dec}_x(C).$$

We require that $\mathrm{dec}_x(\{m\}_y) = m$. This means that the owner of the private key x can recover a message encrypted with the public key y.

Many times in this paper we will need to serially encrypt a text under multiple public keys $y_N, ..., y_i$, and we will use the following notation to make this less cumbersome:

$$\{m\}_{y_N:y_i} \overset{\text{def}}{=} \{...\{\{m\}_{y_N}\}_{y_{N-1}}...\}_{y_i}.$$

At times throughout this paper we will refer to a "security parameter" ρ. The security parameter has a specific cryptographic meaning, but intuitively can be thought of as the length of the key in bits [10].

2.2 Indistinguishability Under Adaptive Chosen Ciphertext Attack

We require that the cryptosystem used in our protocol have the property of *indistinguishability under adaptive chosen ciphertext attack (IND-CCA2)* [3]. Intuitively, this means that it is impossible to learn any information a plaintext m from an encryption $\{m\}_y$, even when given access to a decryption oracle for all ciphertexts other than $\{m\}_y$. Although IND-CCA2 is a very strong property, there are cryptosystems that are known to satisfy it [6, 4].

In our definition of an IND-CCA2 encryption scheme, we will make use of the *distinguishing game.*

2.2.1 The Distinguishing Game

The distinguishing game is played between a challenger and an oracle. This game is slightly different from, but equivalent to, the standard *adaptive chosen ciphertext game* [3].

1. The oracle chooses a keypair (x, y), and gives y to the challengers.

2. The challenger may encrypt polynomially many messages m using y. The challenger may also choose polynomially many ciphertexts C and send them to the oracle, who sends back $\mathrm{dec}_x(C)$.

3. The challenger chooses two plaintexts m_0 and m_1.

4. The oracle chooses a bit $b \in \{0, 1\}$ uniformly at random, and returns the ordered pair $(\{m_b\}_y, \{m_{\bar{b}}\}_y)$.

5. The challenger may encrypt polynomially many messages m using y. The challenger may also choose polynomially many ciphertexts $C \neq \{m_b\}_y, \{m_{\bar{b}}\}_y$ and send them to the oracle, who sends back $\mathrm{dec}_x(C)$.

6. The challenger attempts to guess whether $b = 0$ or $b = 1$.

Let A be a polynomial-time challenger. Then

$$\Pr[A(m_0, m_1, 0) = 1]$$

is the probability that A outputs 1 when the bit $b = 0$, and

$$\Pr[A(m_0, m_1, 1) = 1]$$

is the probability that A outputs 1 when the bit $b = 1$. In both cases, the probability is taken over the randomness of the key generation in step 1. The challenger's *advantage* is equal to

$$\Pr[A(m_0, m_1, 1) = 1] - \Pr[A(m_0, m_1, 0) = 1].$$

Now we can define indistinguishability under adaptive chosen ciphertext attack as follows:

Definition 1. *A cryptosystem is IND-CCA2 if, for all probabilistic polynomial-time challengers, the advantage in the distinguishing game is negligible (dominated by* $\frac{1}{f(\rho)}$*, where* f *is any polynomial and* ρ *is a security parameter).*

2.3 Digital Signatures

We use the standard definition of digital signature schemes from [3]. A *digital signature scheme* $\mathcal{DS} = (\mathcal{K}, \textsc{sig}, \mathrm{VF})$ consists of three algorithms:

- The randomized *key generation* algorithm $\mathcal{K}$, which returns a pair (u, v) of keys. These are the private and public keys, respectively.

- The (possibly randomized) *signing algorithm* $\textsc{sig}$, which takes a private key u and a message m to produce a signature $\sigma = \textsc{sig}_u\{m\}$.

- The deterministic *verification* algorithm VF, which takes a public key v, a message m, and a candidate signature σ. VF returns 1 if σ is a valid signature of m with key u, and 0 otherwise. That is, $\mathrm{VF}(v, m, \textsc{sig}_u\{m\})$ returns 1.

The desired security property for a digital signature scheme is *unforgeability*, which means that without the private key u, it is computationally infeasible to produce a signature $\textsc{sig}_u(m)$ for a message m that one has not previously seen signed with u. For a more formal treatment, see [3].

3. PROBLEM SPECIFICATION

The anonymity-preserving data collection protocol takes place between a large set of mutually distrustful parties. One of these parties has a special role and is denoted the "data miner," while the other N parties have interchangeable roles and are denoted "respondents." Each respondent i, $1 \leq i \leq N$ has a response d_i. All responses are assumed to be of identical length. The goal of the protocol is for the miner to learn the responses from each respondent, but without being able to determine which response came from which respondent. In other words, the miner should learn a random permutation of the set $\{d_1, ..., d_N\}$, but should not learn anything about that permutation.

We assume that during the protocol, all participants remain online. Each respondent has a secure communication channel with the data miner. These are reasonable assumptions in a scenario where the respondents are using a Web interface to communicate with a server operated by the data miner. We assume that prior to the protocol execution each respondent i has obtained a public encryption key pair (x_i, y_i) for an IND-CCA2 encryption scheme, and a signature key pair (u_i, v_i) for a secure (unforgeable) signature scheme. Each respondent and the data miner knows the public keys y_i and v_i for all respondents. Likewise, the data miner has a public key pair (x_{DM}, y_{DM}), and the public key y_{DM} is known to all respondents.

In a practical implementation, the distribution of these public keys is delegated to a trusted *certification authority*, whose job is to associate individuals with their public keys. Note that this is the only assumption of trust required by the protocol. There are several businesses providing trusted certification authority functionality, so this is a reasonable (and standard) assumption.

4. PROTOCOL CORRECTNESS

We prove the correctness of our protocol in the *malicious model* [9], where protocol participants may deviate arbitrarily from the protocol specification. In this model any participant can prevent the protocol from completing by refusing to participate, so we are unable to prove that the protocol always terminates, much less that it always terminates with correct results. Instead, we prove that if the protocol terminates, certain properties are maintained. In this section, we give three properties that together define a correct protocol for anonymity-preserving data collection in the malicious model.

4.1 Collusion Resistant Anonymity

In an online data collection scenario, a respondent knows nothing about her peer-respondents except their public keys. Some (or all) of the other participants may be shills controlled by the data miner who exist only to lure honest participants into a false sense of anonymity. Because dishonest respondents colluding with the data miner is a legitimate threat, we require that our protocol be *collusion resistant.* This means that even if k of the N respondents are corrupt and in collusion with the data miner, the data miner will be unable to determine which of the $N - k$ honest participant responses belongs to which honest participant. Of course the data miner will be able to determine whether a response comes from an honest respondent or a colluding respondent, because a colluding respondent can tell the data miner which response is hers. Note that if there is only a single honest respondent, the data miner will be able to collude with all other respondents and learn her response.

We formalize our notion of anonymity for a data collection protocol when k out of the N respondents are dishonest by defining an "anonymization game" which is similar to the distinguishing game given in section 2.2.1. The anonymization game is played between a challenger and an oracle, who participate in the data collection protocol together. The challenger plays the roles of the data miner and the k dishonest colluding respondents, while the oracle plays the role of the honest respondents. The challenger is assumed to not know the private keys of any honest respondent. The protocol is anonymous if the challenger can win the game only with negligible probability. Prior to playing the game, the challenger may choose plaintext responses for all honest respondents and give them to the oracle, who will then participate in the anonymity-preserving data collection protocol using those responses for the honest respondents. The challenger may repeat this process polynomially many times. Then the actual game begins, and the following happens:

1. The challenger chooses two honest participants h_α and h_β, and two plaintext responses m_0 and m_1. He also chooses a plaintext response d_{h_i} for each other honest participant h_i.

2. The oracle chooses a bit $b \in \{0, 1\}$ uniformly at random. Then the oracle sets $d_{h_\alpha} = m_b$ and $d_{h_\beta} = m_{\bar{b}}$.

3. The oracle participates in the anonymity protocol with the response of honest respondent h_i as d_{h_i}. The oracle plays the role of all honest respondents. The challenger plays the role of the data miner and all dishonest respondents, and he may deviate arbitrarily from the protocol specification.

4. After observing the protocol run, the challenger guesses whether $b = 0$ or $b = 1$.

Let D be a probabilistic polynomial-time challenger. Then

$$\Pr[D(m_0, m_1, h_\alpha, h_\beta, 0) = 1]$$

is the probability that D outputs 1 when the bit $b = 0$, and

$$\Pr[D(m_0, m_1, h_\alpha, h_\beta, 1) = 1]$$

is the probability that D outputs 1 when the bit $b = 1$. In both cases, the probability is taken over the randomness of the key generation and encryption algorithms used by the oracle. The challenger's *advantage* is equal to

$$\Pr[D(m_0, m_1, h_\alpha, h_\beta, 1) = 1] - \Pr[D(m_0, m_1, h_\alpha, h_\beta, 0) = 1].$$

Definition 2. *A data collection protocol is* anonymous *if, for all probabilistic polynomial-time challengers, the advantage in the anonymity game is negligible.*

Note that this definition is valid only when there are at least two honest respondents, which corresponds to our notion that it is impossible for any anonymity to exist when there is only a single honest respondent.

4.2 Integrity

Ideally, we would like to ensure that an honest data miner always receives an unaltered response from each respondent. However, this is difficult in a protocol where responses pass through every respondent during the anonymization process, as any of those respondents could be malicious (but not in collusion with the data miner) and choose to substitute some subset of the responses with other data while they are under her control. Since it seems difficult to provide authenticity guarantees on the responses while maintaining anonymity, we are satisfied to detect the occurrence of substitutions. We will say that our protocol maintains *integrity* if, at the end of protocol execution with an honest data miner, one of the following two statements is true:

1. The data miner has the correct plaintext responses for all honest respondents, or

2. The data miner is informed that some honest respondent's response has been substituted.

We are unable to make integrity claims when the data miner is dishonest, as a dishonest data miner has the power to corrupt whichever responses he wishes. However, the assumption is that the data miner is genuinely interested in learning the responses, and therefore has no such incentive.

4.3 Confidentiality

In our scenario, the respondents are taking part in a confidential survey with the data miner. The data miner should learn all plaintext responses at the end of the protocol, but a respondent should not learn any response other than her own. If the data miner is dishonest, then he can reveal the set of responses $\{d_1, ..., d_N\}$ after they have been decrypted, even though he will not know which response belongs to which respondent. We want to insure that dishonest behavior on the part of the data miner is the *only* way that the set of plaintext responses can be revealed. In other words, no coalition of dishonest respondents should be able to learn any response belonging to an honest respondent if the miner is honest.

5. EFFICIENT ANONYMOUS DATA COLLECTION

In this section we present our protocol for Anonymous Data Collection. We then prove that it satisfies all of the properties stated in section 4, and is therefore secure in the malicious model.

5.1 Protocol Setup

Prior to the protocol execution, the participants must learn one another's public encryption keys and public signature keys. As is standard in cryptographic protocols, the associations between identities and keys are handled by a *certification authority*. The certification authority is a trusted party with whom every participant is assumed to have a secure communication channel. To learn the public encryption key y_i and verification key v_i for respondent i, participants query the certification authority who responds with (y_i, v_i). Likewise, participants learn the public encryption key y_{DM} for the data miner.

The participants must also all agree upon a canonical ordering of the respondents. This can be done, for instance, by having each respondent sign an ordering sent by the data miner, and then verifying the signatures of all other respondents.

Note that this setup needs only to be done once, and afterwards the protocol participants can perform multiple protocol executions with no need to make additional contact with the certification authority.

5.2 The Protocol

Our protocol for anonymous data collection is shown in Algorithm 1. It consists of 5 phases: keypair generation, data submission, anonymization, verification, and decryption. These phases are described in more detail below.

Generation of Temporary Keypairs

In this phase, every respondent i chooses a fresh *secondary* key pair (w_i, z_i) which is distinct from the *primary* key pair (x_i, y_i) that is registered with the certification authority. Each respondent i then self-certifies her secondary public key z_i by sending the message

$$z_i, \text{TIMESTAMP}, \text{SIG}_{u_i}\{z_i, \text{TIMESTAMP}\}$$

to the data miner, who forwards these messages to the other respondents. In this way every respondent learns the second public key z_i for each other respondent i, which is guaranteed to be freshly chosen by i because signatures are not forgeable.

Data Submission

In the data submission phase, respondents encrypt their responses first with the data miner's public key, then with all respondents' secondary public keys, and finally with all respondents' primary public keys. The encryptions are applied in the canonical ordering determined during the protocol setup. The primary key encryptions are stripped off during the anonymization phase. Because the cooperation of all respondents is necessary to remove the secondary key encryptions, every respondent will have a chance to abort the protocol before anonymity is compromised if the anonymization phase did not go according to protocol specification.

Anonymization

In the anonymization phase, the respondents take turns shuffling the encrypted responses and removing a level of encryption. Since every level of encryption must be removed in order for the verification to pass, every respondent is ensured an opportunity to shuffle the encrypted responses. If two ciphertexts are identical, this means a dishonest participant has attempted to duplicate a response, and the protocol is aborted.

Verification

In this phase the respondents verify that the shuffles have been done correctly. By taking advantage of the fact that each respondent knows one of the responses, this is done without the use of zero-knowledge proofs. Each respondent i signs a message only if he sees his own ciphertext, $C'_i = \{C''_i\}_{z_N:z_1}$ among the set of permuted ciphertexts. These signatures are then verified by all respondents. If all of the signatures verify, an honest respondent can reason as follows:

- Every other honest respondent saw her ciphertext in the set of permuted ciphertexts.
- My own shuffle step must have included ciphertexts from all honest respondents.
- Since I performed a random shuffle and did not reveal the permutation, a dishonest data miner cannot know which honest ciphertext belongs to me.

Since the respondent knows her ciphertext is anonymous among the honest respondents' ciphertexts, she gives the data miner her secondary private key.

Decryption

In this phase the data miner uses the respondents' secondary private keys and his own private key to decrypt the responses. He learns the plaintext responses, but not the associations between responses and respondents.

5.3 Security Arguments

In this section we argue that the security properties introduced in section 4 are satisfied by the protocol given in algorithm 1.

5.3.1 Anonymity

An intuitive argument for the anonymity of the protocol is that the data miner and colluding respondents have two choices for their behavior. On the one hand, they can behave honestly, in which case they will learn the final decrypted plaintexts, but they will not learn the associations between C' and C ciphertexts. On the other hand, they can behave dishonestly and learn some associations between C' and C ciphertexts, but then the verification phase will fail and they will not learn the decryptions of the C' ciphertexts.

Our proof is in two parts. First, we show that when honest respondents receive ciphertexts for decryption in phase 2, then either there is exactly one copy of the correct ciphertext for each honest participant, or the deviation from the protocol is detected and the protocol is aborted before the challenger is able to win the verification game. Second, we show that a challenger who can win the anonymity game while maintaining the above property can also win

- Phase 0: Secondary Keypair Generation. For $i = 1, ..., N$
  - Respondent i chooses a public key pair (w_i, z_i).
  - Respondent i sends to the data miner:

    $$z_i, \text{TIMESTAMP}, \text{SIG}_{u_i}\{z_i, \text{TIMESTAMP}\}$$

  - The data miner forwards to all other respondents:

    $$z_i, \text{TIMESTAMP}, \text{SIG}_{u_i}\{z_i, \text{TIMESTAMP}\}$$

  - If any signature fails to verify, the protocol is aborted.
- Phase 1: Data submission. For $i = 1, ..., N$:
  - Respondent i encrypts his data d_i first with the miner's public key, and then with all respondent's secondary public keys:

    $$\begin{aligned} C_i'' &= \{d_i\}_{y_{DM}} \\ C_i' &= \{C_i''\}_{z_N:z_1} \end{aligned}$$

  - Respondent i stores C_i' for later use, and encrypts again with all respondent's primary public keys:

    $$C_i = \{C_i'\}_{y_N:y_1}$$

  - Respondent i sends the ciphertext C_i to the miner

  The miner sets the initial values of $(D_1, ..., D_N)$ as $(C_1, ..., C_N)$
- Phase 2: Anonymization. For $i = 1, ..., N$:
  - The miner sends $(D_1, ..., D_N)$ which are encrypted under the keys $y_i, ..., y_N$ to respondent i.
  - If any ciphertext is included more than once in $(D_1, ..., D_N)$, then respondent i aborts the protocol.
  - Respondent i uses her private key x_i to strip off the ith level of encryption, and permutes the pieces. Her output is $(R_1, ..., R_N)$ where

    $$R_j = \text{dec}_{w_i}[D_{\pi(j)}]$$

    where π is a random permutation on $\{1, ..., N\}$.
  - Respondent i sends $(R_1, ..., R_N)$ to the data miner.
  - The data miner sets $(D_1, ..., D_N) = (R_1, ..., R_N)$.
- Phase 3: Verification. At the beginning of this phase, the data miner holds $(D_1, ..., D_N)$. If the participants have behaved honestly, this should be a permutation of $(C_1', ..., C_N')$.
  - The data miner sends $(D_1, ..., D_N)$ to all participants.
  - Each participant i verifies that C_i' is included among $(D_1, ..., D_N)$. If it is, she sends $\text{SIG}_{u_i}\{(D_1, ..., D_N)\}$ to the data miner.
  - The data miner forwards the signatures $\text{SIG}_{u_i}\{(D_1, ..., D_N)\}$, $i = 1, ..., N$ to all respondents.
  - Each respondent i verifies the signatures. If they *all* verify, respondent i sends his secondary private key w_i to the data miner.
- Phase 4: Decryption. Using the keys $w_1, ..., w_N$ and x_{DM}, the data miner removes the remaining levels of encryption from $D_1, ..., D_N$, resulting in a permutation of the original responses $d_1, ..., d_N$.

Algorithm 1: Protocol for Anonymous Data Collection

the distinguishing game, which is a contradiction because the underlying encryption scheme is IND-CCA2.

Part 1: Suppose that during step h_i of phase 2, honest respondent h_i receives ciphertexts $(D_1, ..., D_N)$, but that there is some honest respondent h_p for which the ciphertext $\{C_{h_p}'\}_{y_N:y_{h_i}}$ appears either more than once or not at all. If any ciphertext appears more than once, this is detected by honest respondent h_i and the protocol is aborted before the challenger learns the secondary private keys.

Now we wish to show that if an honest ciphertext is dishonestly replaced so that it does not appear in step h_i of phase 2, then the verification in phase 3 will fail. Suppose that honest participant h_i does not receive $\{C_{h_p}'\}_{y_N:y_{h_i}}$ as part of the set $(D_1, ..., D_N)$.

In this case it is infeasible for the challenger to learn $\{C_{h_p}'\}_{y_N:y_{h_i-1}}$ because they did not give $\{C_{h_p}'\}_{y_N:y_{h_i}}$ to participant h_i. However, they must learn C_{h_p}' to satisfy honest participant h_p in the verification step. This is infeasible without the private key x_{h_i}. Therefore verification fails and the protocol is aborted before the challenger learns the secondary private keys.

Now we must show that the challenger cannot win the anonymization game when he does not learn the secondary private keys. It is possible that by duplicating or substituting ciphertexts of honest respondents, the challenger can learn the partially-decrypted ciphertexts C_{h_α}' and C_{h_β}' for the honest respondents h_α and h_β he has chosen in the game. For example, the challenger could substitute known information for all encrypted responses except C_{h_α}. Then after the decryption phase, he would know C_{h_α}'. However, even if the challenger learns C_{h_α}' and C_{h_β}', each of these ciphertexts is encrypted with the key z_{h_α} (as well as the keys of all other honest participants) which is unknown to the challenger. Therefore determining which decrypts to m_0 and which decrypts to m_1 is *exactly equivalent* to the distinguishing game, and cannot be done due to the assumption that the encryption scheme is IND-CCA2.

Part 2: Now suppose that the challenger honestly handles all ciphertexts belonging to honest participants. Suppose also that there is a probabilistic polynomial time algorithm D that allows this challenger to win the anonymization game with non-negligible probability. We will show how to use D as a subroutine to probabilistic polynomial-time algorithm A that wins the distinguishing game with non-negligible probability. Because of the assumption that the underlying encryption scheme is IND-CCA2, this is a contradiction, and we will conclude that no such D exists. Let the set of k honest respondents in the anonymization protocol be $H = \{h_1, ..., h_k\}$. Let D be an algorithm that allows the challenger to win the anonymization game with non-negligible probability. Then there exist honest participants h_α and h_β such that for some polynomial f, and for sufficiently large values of ρ,

$$\begin{aligned} & \Pr[D(m_0, m_1, h_\alpha, h_\beta, 1) = 1] \\ - \; & \Pr[D(m_0, m_1, h_\alpha, h_\beta, 0) = 1] \\ > \; & \frac{1}{f(\rho)}. \end{aligned}$$

To apply D, A must simulate the oracle in the anonymization game to reproduce the view of the challenger. We show how A is able to do this.

Algorithm A begins by asking the distinguishing game oracle to generate a keypair (x_o, y_o). A will use y_o for y_{h_k}, the public key belonging to the last honest respondent h_k. A will be able to simulate all messages from honest respondents despite not knowing x_o or y_o.

A applies D to learn its choices in step 1 of the anonymization game. A therefore learns the following:

- Two honest participants h_α and h_β, and two plaintext responses m_0 and m_1
- A plaintext response d_{h_i} for each other honest respondent h_i

Then, for each honest respondent h_i, A chooses

- (x_{h_i}, y_{h_i}), a primary keypair (not chosen for respondent h_k),
- (w_{h_i}, z_{h_i}), a secondary keypair, and
- (u_{h_i}, v_{h_i}), a signature keypair

A selects plaintexts m_0 and m_1 to be the plaintext responses for h_α and h_β.

A is now ready to play the role of the oracle in the anonymization game by simulating the messages of honest participants in the protocol execution. For each phase of the protocol, we will explain how A is able to reproduce the messages sent in that phase.

Phase 0: A has all the necessary keys to reproduce this phase exactly.

Phase 1: For all honest respondents h_i other than h_α and h_β, A encrypts response d_{h_i} using the appropriate public keys (and the encryption oracle to encrypt with y_{h_k}) which results in the ciphertext C_i. A then encrypts m_0 and m_1 in a special way. First, A encrypts using the secondary public keys:

$$\begin{aligned} m_0' &= \{m_0\}_{z_N:z_1}, \text{ and} \\ m_1' &= \{m_1\}_{z_N:z_1}. \end{aligned}$$

Next, A encrypts with all keys that come after h_k in the sequence:

$$\begin{aligned} M_0 &= \{m_0'\}_{y_N:y_{h_k-1}}, \text{ and} \\ M_1 &= \{m_1'\}_{y_N:y_{h_k-1}}. \end{aligned}$$

Next, A gives M_0 and M_1 to the distinguishing game oracle, getting back:

$$\begin{aligned} M_b &= \{m_b'\}_{y_N:y_{h_k}}, \text{ and} \\ M_{\bar{b}} &= \{m_{\bar{b}}'\}_{y_N:y_{h_k}}. \end{aligned}$$

Finally, A encrypts M_b and $M_{\bar{b}}$ with the remaining public keys to get the final encryptions:

$$\begin{aligned} C_{h_\alpha} &= \{m_b'\}_{y_N:y_1}, \text{ and} \\ C_{h_\beta} &= \{m_{\bar{b}}'\}_{y_N:y_1}. \end{aligned}$$

Phase 2: For all honest participant rounds h_i of phase 2 other than round h_k, A has the key x_{h_i} necessary to decrypt the ciphertexts $(D_1, ..., D_N)$ provided by the challenger (who is playing the role of the miner). A permutes the resulting decryptions and sends them to the challenger.

Round h_k is more difficult, because A does not know the key x_{h_k}. Here we must use our assumption that every honest participant's ciphertext appears exactly once in $(D_1, ..., D_N)$. In particular, the ciphertexts

$$\begin{aligned} M_b &= \{m_b'\}_{y_N:y_{h_k}}, \text{ and} \\ M_{\bar{b}} &= \{m_{\bar{b}}'\}_{y_N:y_{h_k}}. \end{aligned}$$

appear exactly once. For all ciphertexts D_j in $(D_1, ..., D_N)$ other than M_b and $M_{\bar{b}}$, A may use the decryption oracle provided by the distinguishing game to obtain $\mathrm{dec}_{x_{h_k}}(D_j)$.

However, A is not allowed to use the decryption oracle on M_b or $M_{\bar{b}}$. Instead, when A sees the ciphertexts $\{m_b'\}_{y_N:y_{h_k}}$ and $\{m_{\bar{b}}'\}_{y_N:y_{h_k}}$, he can either simulate decryption as

$$\begin{aligned} &\{m_0'\}_{y_N:y_{h_k-1}} \quad \text{and} \quad \{m_1'\}_{y_N:y_{h_k-1}} \text{ or} \\ &\{m_1'\}_{y_N:y_{h_k-1}} \quad \text{and} \quad \{m_0'\}_{y_N:y_{h_k-1}}, \end{aligned}$$

which are the values M_0 and M_1 known to A. Because A sends the challenger a random permutation of the decryptions, the challenger cannot distinguish between A making the correct choice and permuting, or A making the incorrect choice and permuting. Thus either choice will suffice, and A may randomly choose one or the other.

Phases 3 and 4: A has all the necessary keys to reproduce these phases exactly.

Now A simulates the view of the challenger and applies D to the view. If D outputs 1, then A outputs 1, and if D outputs 0, A outputs 0.

We will now analyze the probability of A outputting 1 if the distinguishing oracle chose $b = 0$ and if the distinguishing oracle chose $b = 1$. If $b = 0$, then the view of the challenger is $(m_0, m_1, h_\alpha, h_\beta, 0)$. If $b = 1$, then the view of the challenger is $(m_0, m_1, h_\alpha, h_\beta, 1)$. Let

$$\begin{aligned} p_0 &= \Pr[D(m_0, m_1, h_\alpha, h_\beta, 0) = 1], \\ p_1 &= \Pr[D(m_0, m_1, h_\alpha, h_\beta, 1) = 1]. \end{aligned}$$

Based on our assumption that D wins the anonymity game, we have that $p_1 - p_0 > \frac{1}{f(\rho)}$. Now we make a simple substitution,

$$\begin{aligned} & \Pr[A(m_0, m_1, 1) = 1] - \Pr[A(m_0, m_1, 0) = 1] \\ = \;& p_1 - p_0 \\ > \;& \frac{1}{f(\rho)}. \end{aligned}$$

We conclude that A can win the distinguishing game with non-negligible probability, which contradicts the IND-CCA2 property of the underlying encryption scheme.

5.3.2 Integrity

The honest respondents verify the presence of their C_i' ciphertext in phase 3 of the protocol; all ciphertexts must be present for any decryption to take place. When the secondary private keys are handed over to the data miner for decryption, he can easily confirm whether the private key w_i actually corresponds to the public key z_i by encrypting data with z_i and determining whether it decrypts with w_i. Then the data miner uses verified keys to decrypt verified ciphertexts, and as a result learns the correct plaintext responses.

If the verification in phase 3 fails, then the data miner has not received $\mathrm{SIG}_{u_i}(D_1, ..., D_N)$ from some honest participant h_i. In this case the data miner learns that participant h_i's response has been substituted. Thus our two-part definition of integrity given in section 4.2 is satisfied.

5.3.3 Confidentiality

The inner-most level of encryption on each response is with y_{DM}. Therefore only the data miner can perform the final decryption and learn the responses.

6. EFFICIENCY

Our protocol is efficient in terms of communication and computational complexity. In this section we will quantify the complexity of the protocol.

Computation

Operations requiring computation include key pair generation, encryption, decryption, signing, and signature verification. In phase 0, each respondent generates 1 key pair, signs 1 message, and performs N signature verifications. In phase 1, each respondent performs $2N+1$ encryptions. In phase 2, each respondent performs N decryptions. In phase 3, each respondent performs 1 signature and N signature verifications. In phase 4, the data miner performs $N^2 + N$ decryptions. We conclude that the total computational complexity is $O(N^2)$. Note, however, that the computational complexity for each individual respondent is only $O(N)$. This is advantageous because the respondents are more likely to be computationally bounded than the data miner.

Number of Communication Rounds

Phase 0 is parallelizable and requires 2 rounds. Phase 1 is parallelizable and requires only 1 round. Phase 2 cannot be parallelized and requires $2N$ rounds. Phase 3 can be parallelized and requires 4 rounds. Phase 4 does not involve communication. We conclude that the protocol requires $O(N)$ communication rounds.

Total Communication

Let us assume that the size of a response is S bits, the size of a key is T bits, and that the size of a signature is Q bits. Phase 0 requires the transmission of N^2 signatures and keys, for a total of $(Q + T)N^2$ bits. In phase 1, N S-bit ciphertexts are transmitted. In each iteration of phase 2 , $2SN$ bits are transmitted, for a total of $2SN^2$ bits. Phase 3 transmits SN^2 bits for the broadcast of $(D_1, ..., D_N)$, $(Q + T)N$ bits to transmit signatures and keys to the miner, and QN^2 bits to broadcast the signatures from the miner back to the respondents. We conclude that the protocol transmits $O((Q+S+T)N^2)$ bits. Since Q and T are constant parameters of the cryptosystem, we can simplify this to $O(SN^2)$ bits.

7. CONCLUSIONS

We have presented an efficient protocol for anonymity-preserving data collection that does not rely on zero-knowledge proofs to be secure in the malicious model. We have provided anonymity by having the respondents function as mix-servers which shuffle the set of responses. The critical insight of our research is that by taking advantage of the fact that each respondent/mix-server knows her own response, we can confirm the validity of the shuffles without using zero-knowledge proofs. In a traditional mix-net scenario, the mix-servers and the data providers are distinct entities, so validity confirmation of this type is not possible. It is our hope that the data-mining community will find our protocol useful when collecting sensitive data from respondents.

8. REFERENCES

[1] D. Agrawal and C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *PODS '01: Proc. of the 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 247–255, 2001.

[2] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proc. of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 439–450, 2000.

[3] M. Bellare and P. Rogaway. Introduction to modern cryptography, 2005. URL: http://www-cse.ucsd.edu/users/mihir/cse207/classnotes.html.

[4] X. Boyen, Q. Mei, and B. Waters. Direct chosen ciphertext security from identity-based techniques. In *CCS '05: Proc. of the 12th ACM Conference on Computer and Communications Security*, pages 320–329, 2005.

[5] C. Clifton, M. Kantarcioglou, J. Vaidya, X. Lin, and M. Zhu. Tools for privacy preserving distributed data mining. *ACM SIGKDD Explorations*, 4(2):28–34, 2002.

[6] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Proc. of Advances in Cryptology - CRYPTO 1998*, pages 13–25, 1998.

[7] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *Proc. 13th USENIX Security Symposium*, pages 303–320, 2004.

[8] W. Du and Z. Zhan. Using randomized response techniques for privacy-preserving data mining. In *KDD '03: Proc. of the 9th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 505–516, 2003.

[9] O. Goldreich. *Foundations of Cryptography: Basic Tools.* Cambridge University Press, 2001.

[10] O. Goldreich. *Foundations of Cryptography: Volume II Basic Applications.* Cambridge University Press, 2001.

[11] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC '87: Proc. of the 19th Annual ACM Conference on Theory of Computing*, pages 218–229, 1987.

[12] P. Golle, S. Zhong, D. Boneh, M. Jakobsson, and A. Juels. Optimistic mixing for exit-polls. In *Proc. of Advances in Cryptology - ASIACRYPT 2002*, pages 451–465, 2002.

[13] J. Groth. A verifiable secret shuffle of homomorphic encryptions. In *PKC '03: Proc. of the 6th International Workshop on Theory and Practice in Public Key Cryptography*, pages 145–160, 2003.

[14] Y. Lindell and B. Pinkas. Privacy preserving data mining. *J. Cryptology*, 15(3):177–206, 2002.

[15] A. Neff. Verifiable mixing (shuffling) of ElGamal pairs. http://www.votehere.net/vhti/documentation/egshuf.pdf.

[16] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):571–588, 2002.

[17] L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

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

[19] D. Wikström. Four practical attacks for "optimistic mixing for exit-polls". Technical Report T2003-04, Swedish Institute of Computer Sciences (SICS), 2003.

[20] R. Wright and Z. Yang. Privacy-preserving Bayesian network structure computation on distributed heterogeneous data. In *KDD '04: Proc. of the 10th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 713–718, 2004.

[21] Z. Yang, S. Zhong, and R. Wright. Anonymity-preserving data collection. In *KDD '05: Proc. of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 334–343, 2005.

[22] A. Yao. How to generate and exchange secrets. In *FOCS '86: Proc. of the 27th IEEE Annual Symposium on Foundations of Computer Science*, pages 162–167, 1986.

APPENDIX

A. AN ATTACK ON THE YZW PROTOCOL

In this appendix, we show an attack against the data collection protocol of Yang *et al.* [21, section 5.2]. We will refer to this protocol as the YZW protocol. This protocol is intended to be collusion resistant so that $t-1$ dishonest leaders (out of t total leaders), in collusion with a dishonest data miner, cannot learn any associations between honest respondents and their responses. Due to a subtle flaw in the use of zero-knowledge proofs, it is actually possible for a single dishonest leader to collude with the data miner and learn *all* associations.

This appendix is organized as follows. In section A.1 we provide a complete description of the malicious-model version of the protocol from [21]. Then, in section A.2, we explain how the protocol has misused zero-knowledge proofs, and how this can be exploited by a malicious data miner and a malicious respondent to break anonymity of all honest respondents. Finally, in section A.3, we argue that even with correct zero-knowledge proofs, the YZW protocol still has impractical communication complexity.

A.1 Description of the YZW protocol

Unlike our protocol, which is compatible with any IND-CCA2 cryptosystem, the YZW protocol relies on the ElGamal encryption scheme. In ElGamal, the public key is (p, g, g^x) and the private key is x, where p is a large random prime, g is the generator of the multiplicative group of integers modulo p, and x is a random integer such that $1 \leq x \leq p-2$. A ciphertext of message m is a pair $(g^k \mod p, m \cdot (g^x)^k \mod p)$, where k is a random integer such that $1 \leq k \leq p-2$.

An important property of ElGamal ciphertexts is that they can be easily *rerandomized* without access to the private key, *i.e.*, given a ciphertext $(C^{(1)}, C^{(2)})$, it is easy to produce, without decrypting, another ciphertext $(C^{(1)} \cdot g^{k'} \mod p, C^{(2)} \cdot (g^x)^{k'} \mod p)$ which encrypts the same plaintext. Moreover, given two ciphertexts, it is not feasible to determine whether they encrypt the same plaintext.

The protocol also makes use of the following three zero-knowledge proofs, which are intended to prevent any of the protocol participants from deviating from the protocol:

- $\mathsf{PoK}(C)$, where C is an ElGamal ciphertext. This is a proof of knowledge of the plaintext of C.
- $\mathsf{PoR}((C_1, ..., C_N), (C'_1, ..., C'_N))$, where all C_i and C'_i are ElGamal ciphertexts. This is a proof that $(C'_1, ..., C'_N)$ is a permuted rerandomization of $(C_1, ..., C_N)$.
- $\mathsf{PoD}(q, C^{(2)}, y)$, where $C^{(2)}$ is the second component of an ElGamal ciphertext and $y = g^x \mod p$ is a public key. This is a proof that $q = (C^{(2)})^x$, where x is the private key corresponding to y.

We restate the protocol below. There are N respondents, of whom t are designated as "leaders." Each leader i has an ElGamal key pair, in which x_i is the private key, and the public key includes $y_i = g^{x_i}$. The public key is known to all respondents, while the private key x_i is known only to leader i. Let

$$y = \prod_{y=1}^{t} y_i$$

be the product of all public y_i values. The protocol consists of three phases:

- Phase 1: N-round data submission.
  For $i = 1, ..., N$
  - Respondent i encrypts his data d_i using the public key y to produce the ciphertext C_i:
    $$C_i \stackrel{\text{def}}{=} (C_i^{(1)}, C_i^{(2)}) = (y^{r_i} d_i, g^{r_i}),$$
    where r_i is picked uniformly at random.
  - Respondent i produces a proof $z_i = \mathsf{PoK}(C_i)$, proving that he knows the plaintext of C_i.
  - Respondent i sends C_i and z_i to the miner, who forwards (C_i, z_i) to all other respondents.
  - Each respondent verifies the proof sent by respondent i and if it is missing or invalid, the protocol is aborted.
  - The data miner sets the initial values of $(D_1, ..., D_N)$ to be $(C_1, ..., C_N)$.
- Phase 2: t-round anonymization.
  For $i = 1, ..., t$
  - The miner sends $(D_1, ..., D_N)$ to leader i.
  - Leader i rerandomizes and permutes the data, so that $(R_1, ..., R_N)$ is a permuted rerandomization of $(D_1, ..., D_N)$.
  - Leader i generates a proof
    $$w_i = \mathsf{PoR}((D_1, ..., D_N), (R_1, ..., R_N)).$$
  - Leader i sends $(R_1, ..., R_N)$ and w_i to the miner, who forwards them to all other respondents.
  - Each respondent verifies the proof sent by leader i and if it is missing or invalid, the protocol is aborted.
  - The miner sets the new values of $(D_1, ..., D_N) = (R_1, ..., R_N)$.
- Phase 3: Decryption
  - The miner sends $(D_1, ..., D_N)$ to all leaders.
  - Each leader i computes partial decryptions: for $j = 1, ..., N$,
    $$p_{j,i} = (D_j^{(2)})^{x_i}.$$
  - Each leader i computes a proof
    $$v_{j,i} = \mathsf{PoD}(p_{j,i}, D_j^{(2)}, y_i).$$
  - Each leader i sends $p_{j,i}$ and $v_{j,i}$ to the miner, for $j = 1, ..., N$. The miner forwards them to the other participants.
  - Each participant verifies the proof sent by leader i, and if it is missing or invalid the protocol is aborted.
  - The miner computes the final decryptions: for $j = 1, ..., N$
    $$d'_j = D_j^{(1)} / \prod_{i=1}^{t} p_{j,i}.$$

A.2 Attacking the YZW protocol

The flaw in the YZW protocol is that the zero-knowledge proof of permuted rerandomization PoR is *not* a proof of knowledge. It guarantees that the plaintexts of two ciphertext sets are the same, but this is not enough for anonymity. As long as $(C'_1, ..., C'_N)$ is *some* permuted rerandomization of $(C_1, ..., C_N)$, then an attacker can provide the required proof even if he does *not* know the actual permutation.

In this section, we show how this can be exploited by a malicious data miner who colludes with the last leader and substitutes original ciphertexts (for which associations with respondents are known) for the honestly permuted ciphertexts. This enables them to pass all proofs required by the protocol, and then learn all associations between respondents and responses. Collusion resistance thus fails completely: it is sufficient for the data miner to corrupt the last leader in order to completely break security of the protocol.

1. All participants behave honestly until the beginning of the tth round of phase 2, when it is leader t's turn to rerandomize and permute the data.
2. At the beginning of the tth round, the data miner sends leader t both the current values of $(D_1, ..., D_N)$ and the *original* values of $(C_1, ..., C_N)$ exactly as they were collected from the respondents during phase 1. Leader t produces $(R_1, ..., R_N)$ by rerandomizing and applying a permutation π_t to $(C_1, ..., C_N)$, *not* $(D_1, ..., D_N)$ as the protocol specifies. Because $(R_1, ..., R_N)$ is also a permuted rerandomization of $(D_1, ..., D_N)$, leader t is able to produce the proof
$$\mathsf{PoR}((D_1, ..., D_N), (R_1, ..., R_N)),$$
even though he does not know the permutation to produce $(R_1, ..., R_N)$ from $(D_1, ..., D_N)$.
3. The proof $\mathsf{PoR}((D_1, ..., D_N), (R_1, ..., R_N))$ from leader t is verified by all other leaders, who then decrypt in phase 3. The data miner learns all plaintext responses permuted only by π_t
4. Leader t tells π_t to the data miner, who is then able to associate responses to respondents.

It may appear that this attack is caused simply by an imprecise description of PoR given in [21], and that any actual implementation of the PoR proof would not allow a party to pass the proof $\mathsf{PoR}((D_1, ..., D_N), (R_1, ..., R_N))$ without knowing the permutation. Unfortunately, the implementation suggested in [21] (and originally proposed in [12]) allows precisely this attack.

The proof in [12, 21] relies on the multiplicative homomorphism property of ElGamal. If $(C_1^{(1)}, C_1^{(2)})$ and $(C_2^{(1)}, C_2^{(2)})$ are ElGamal encryptions of plaintexts P_1 and P_2, then
$$(C_1^{(1)} C_2^{(1)}, C_1^{(2)} C_2^{(2)})$$
is an ElGamal encryption of $P_1 P_2$. In order to prove that two sets of ciphertexts $(C_1, ..., C_N)$ and $(R_1, ..., R_N)$ decrypt to the same set of plaintexts, participants are asked to prove that the products of the ciphertexts
$$(\prod_{i=1}^{N} C_i^{(1)}, \prod_{i=1}^{N} C_i^{(2)}) \quad \text{and}$$
$$(\prod_{i=1}^{N} R_i^{(1)}, \prod_{i=1}^{N} R_i^{(2)}),$$
decrypt to the same values (which are products of the original plaintexts). If plaintexts are chosen in such a way that it is difficult to find a set of plaintexts with the same product as the original, then equality of the products of plaintexts implies equality of the plaintexts themselves.

The reason the protocol of [21] fails is that the above statement about equality of products is true, and a malicious leader can prove it, even if he does not know the permutation. In [19], Wikström presents a clever attack that effectively allows the kth leader to present a convincing zero-knowledge proof that his output $(R_1, ..., R_N)$ is a permuted rerandomization of the output from the $k-1$st leader, when in fact $(R_1, ..., R_N)$ is a permuted rerandomization of the original ciphertexts in their original order. (In [19], the attack is described in the context of mix networks, but it also works in the anonymous data collection setting with very slight modifications.)

A.3 Fixing the YZW Protocol

The YZW protocol can potentially be fixed by substituting zero-knowledge proofs of knowledge for the incorrect proofs of [12]. Research on so called verifiable shuffles [13, 15] has led to zero-knowledge proofs in which a participant in an ElGamal rerandomization protocol proves not only that the output ciphertexts decrypt to the same set of plaintexts as the input ciphertexts, but also that the prover knows the the permutation from input ciphertexts to output ciphertexts.

Requiring this additional proof of knowledge makes the attack described above impossible, since a malicious last leader will no longer be able to rerandomize original input ciphertexts instead of the ciphertexts provided by the previous leader and pass the proof of knowledge. Fixing the YZW protocol in this way, however, still requires $O(N^2)$ communication rounds and the use of expensive zero-knowledge proofs, where N is the number of participants. It is not clear whether the proofs of [13, 15] can be carried out in parallel (in general, zero-knowledge proofs do not preserve their properties under concurrent composition), and executing them sequentially results in a protocol with impractical communication complexity. By contrast, our protocol achieves the same security guarantees with $O(N)$ communication rounds and without any zero-knowledge proofs.