Masquerade: Verifiable Multi-Party Aggregation with Secure Multiplicative Commitments

Dimitris Mouris and Nektarios Georgios Tsoutsos

University of Delaware
{jimouris, tsoutsos}@udel.edu

Abstract. In crowd-sourced data aggregation over the Internet, participants share their data points with curators. However, a lack of strong privacy guarantees may discourage participation, which motivates the need for privacy-preserving aggregation protocols. Moreover, existing solutions remain limited with respect to public auditing without revealing the participants' data. In realistic applications, however, there is an increasing need for public verifiability (i.e., verifying the protocol correctness) while preserving the privacy of the participants' inputs, since the participants do not always trust the data curators. At the same time, while publicly distributed ledgers may provide public auditing, these schemes are not designed to protect sensitive information.
In this work, we introduce two protocols, dubbed Masquerade and zk-Masquerade, for computing private statistics, such as sum, average, and histograms, without revealing anything about participants' data. We propose a tailored multiplicative commitment scheme to ensure the integrity of data aggregations and publish all the participants' commitments on a ledger to provide public verifiability. zk-Masquerade detects malicious participants who attempt to poison the aggregation results by adopting two zero-knowledge proof protocols that ensure the validity of shared data points before being aggregated and enable a broad range of numerical and categorical studies. In our experiments, we use homomorphic ciphertexts and commitments for a variable number of participants and evaluate the runtime and the communication cost of our protocols.

Keywords: Homomorphic Commitment, Homomorphic encryption, Public verifiability, Private aggregation

1 Introduction

A variety of Internet applications require gathering and aggregating data from different organizations or individuals to perform studies, collect statistics, and mine interesting patterns about the participating population. Common applications of data aggregation can be found in finance, where financial institutions need to gain insights on customer clusters, as well as in healthcare, where aggregated data are used to discover effective treatments and track the spread of highly infectious diseases. In certain use cases, all participants trust the entity conducting the study (called *data curator*) with their personal data, which results in a straightforward solution. Nevertheless, in the case of a *curious* curator that has incentives to peak into sensitive user data (e.g., for targeted advertisements or even affect election results [29]), the problem becomes significantly more challenging to both protect the privacy of each participant and compute the desired statistics.

Common approaches to support privacy-preserving crowdsourcing rely on anonymizing or adding noise to the collected data so that individual inputs cannot be deduced from the output. Unfortunately, these techniques have limitations since data anonymization can be bypassed [16,41], while local differential privacy (DP) may generate excessive accumulated noise. A practical alternative entails adding noise to the output of the queries using DP techniques so that the presence or absence of a single user cannot be detected from the query answers [20]. Unfortunately, such approaches assume a trusted curator and mostly focus on output privacy, i.e., they do not protect the data privacy of each individual user from the curator. The works in [51,21,8] provide strong privacy guarantees and high utility, however, they still add a non-negligible amount of noise to the results. We provide comparisons with related works in section 9.

The work of [51] introduced *private data aggregation (PDA)*, a special case of secure multiparty computation (MPC) in which multiple participants jointly compute a function over their private data while keeping them private [53]. MPC assumes that client data are distributed amongst two or more computing parties and that the data are secure as long as these parties do not collude. While it is possible to use general-purpose MPC to collect statistics from multiple individuals, however, this comes with significant overheads due to support for arbitrary computation. Additionally, many MPC frameworks [36,31,2] allow only the computing parties to provide *private inputs*; at the same time, since the input parties are allowed to send *any input* to the aggregation, it is possible for a malicious participant to poison the result by uploading invalid data. To defend against such attacks, specialized checks have to be implemented within MPC that require additional communication and computation. Finally, generic solutions do not provide public verifiability since data confidentiality relies on the computing parties not publishing their shares.

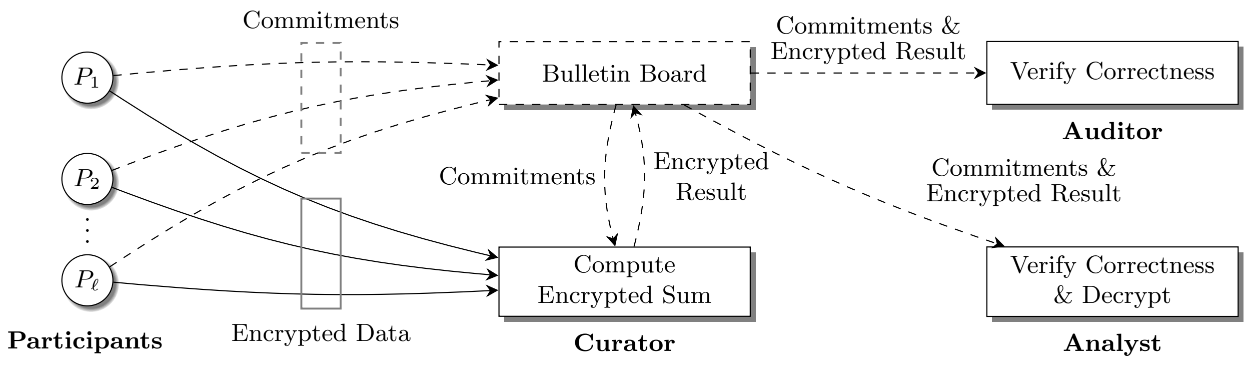
Homomorphic encryption (HE) allows performing meaningful calculations directly on encrypted data without decrypting it, which enables outsourcing a computation to semi-honest servers and returning the encrypted result to the party that initiated the computation [37,25,52]. In particular, HE offers an elegant solution for PDA: each participant encrypts and uploads their private data to a computing party (curator) that performs the aggregation, which in turn returns the encrypted sum to the requesting party (analyst), who decrypts the final sum. Partially HE (PHE) schemes support only one mathematical operation (either addition or multiplication) to be performed on ciphertexts, whereas fully HE (FHE) enables both, but incurs significant overheads, especially for non-linear operations and comparisons. Nevertheless, HE alone does not offer any integrity guarantees to the public on the correctness of the computation. Therefore, the public has to rely on the assumption that all computations are error-free and no party has introduced an invalid ciphertext to the homomorphic sum. Without special integrity checks to enable public audits, relying solely on HE allows computation errors to remain undetected.

Distributed ledgers (e.g., blockchains) can be used as public bulletin boards to enhance transparency and enable public verifiability. However, most of them are either public and reveal data patterns (e.g., Bitcoin), or private without support for public verifiability.

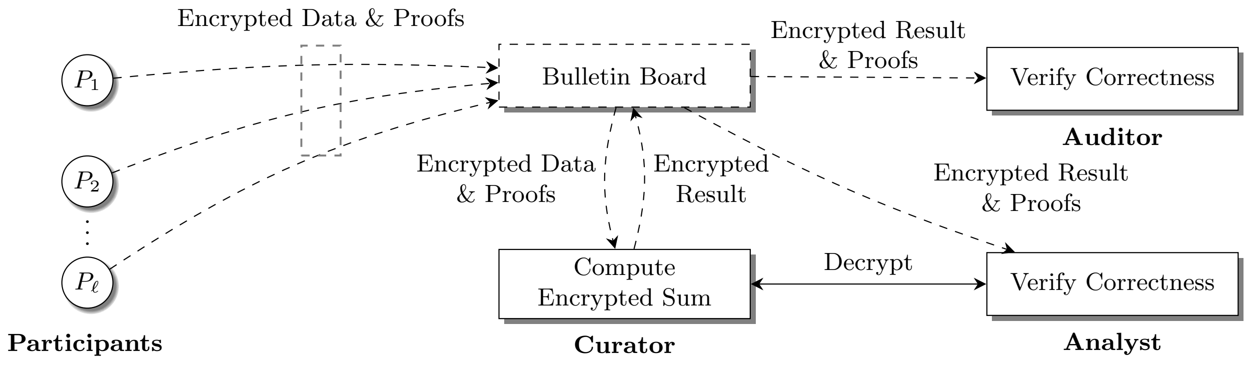
Our contribution. We present two novel constructions for *publicly verifiable PDA (PVPDA)* that allow multiple Internet users to share their data points with an *oblivious curator*, who is unable to deduce any information from the data (other than what it is learned from the published result). Our two protocols, called Masquerade and zk-Masquerade, allow any auditor to check after the protocol execution is over that the protocol results are correct. Our protocols focus on different threat models and introduce a trade-off with performance. In particular, both protocols involve one *analyst* (i.e., a party that originates the computation and will eventually reveal the statistics), *ℓ participants* that share their private data, and the curator who gathers the encrypted data and performs the secure data aggregation. In Masquerade the participants also publish commitments to their data to a public bulletin board, which can be later used to verify the correctness of the result by any auditor and prevent the curator from acting maliciously. Masquerade introduces a new multiplicative-homomorphic commitment scheme that does not rely on any trusted party. On the other hand, zk-Masquerade assumes that either the curator or the analyst as well as multiple participants are malicious and uses zero-knowledge proofs (ZKPs) to enable verifiability.

In both protocols, the analyst outsources the data aggregation to the curator and verifies the correctness of the results (we remark that our protocols enable public verifiability). High-level overviews of our protocols are illustrated in Fig. 1:

– An overview of the Masquerade protocol is depicted in Fig. 1 (a). Masquerade can be viewed as a *single-server PVPDA protocol* as the curator performs most of the computation, save for the final decryption. More specifically, the curator receives homomorphic ciphertexts directly from the participants, which do not reveal anything about their sensitive data as only the analyst has the decryption key. Next, the curator aggregates the ciphertexts and shares with the analyst only the encrypted result of the computation, rendering it impossible for the analyst to learn anything except what can be inferred from the final output. Additionally, each participant publishes commitments to their encrypted messages to a public ledger, and the analyst combines them homomorphically to attest to the correctness of the encrypted result (e.g., detect double-voting).
– An overview of our zk-Masquerade protocol is shown in Fig. 1 (b). Contrary to the previous protocol, in zk-Masquerade both the analyst and the curator share the decryption key and both parties perform



(a) Masquerade uses homomorphic commitments.



(b) zk-Masquerade uses zero-knowledge proofs.

Fig. 1. Overview of Masquerade and zk-Masquerade. (a) Each participant sends their encrypted data to the curator and publishes a commitment to it on the bulletin board. The curator, in turn, performs the homomorphic aggregation and publishes the encrypted result. Finally, the analyst verifies the correctness of the encrypted sum using the commitments and decrypts. **(b)** Each participant publishes their encrypted data along with a zero-knowledge proof that their ciphertext is well-formed to the bulletin board. Next, the curator performs homomorphic aggregation and publishes the encrypted result. Finally, both the curator and the analyst jointly decrypt and acquire the result.

similar computations. Notably, zk-Masquerade addresses the problem of *malicious participants attempting to tamper with the aggregation* by leveraging *non-interactive* zero-knowledge proof (ZKP) protocols and by having each participant publish their encrypted data and proofs to a bulletin board. Based on the type of study, these ZKP protocols ensure that the participants follow the protocol faithfully by proving that their encrypted data points lie either in a range or within a set of valid messages, without disclosing the actual plaintexts. Therefore, malicious participants cannot corrupt the homomorphic sum by sending encryptions of invalid messages. Both the curator and the analyst can check the proofs and compute the encrypted result individually, and they jointly decrypt the final result only if their encrypted aggregations are the same.

Notably, both Masquerade and zk-Masquerade enable public verifiability from any external auditor either through the commitments or through the published ciphertexts and the ZKPs.

Both our protocols enable two classes of studies: *Quantitative* and *Categorical*. An example of the former class includes privacy-preserving smart metering, where users share their electricity consumption readings with a service provider over consecutive time periods to calculate their bill. Smart metering imposes a potential risk to user privacy since fine-grained readings may disclose sensitive information about the clients' habits. Examples in the latter class include surveys where individuals select one category among multiple options. For instance, a study that investigates if the juries in federal courts come from a diverse group of socioeconomic status includes sensitive personal information. This application can be instantiated natively using our protocols, where each jury provides a private categorical input representing their ethnicity. Then, our protocols privately generate a histogram that represents how diverse a group of juries is. Our scheme establishes a novel way for quantitative and categorical studies by protecting each participant's privacy. Overall, our contributions can be summarized as follows:

- Design of a novel commitment scheme that enables *homomorphic multiplication* between commitments. We leverage this scheme to generate a global commitment over the encryption of the aggregation result using the individual commitments of the participants and provide *verifiable guarantees* to the public that there was no error in the homomorphic aggregation of the ciphertexts.
- Construct two privacy-preserving data aggregation protocols with public verifiability that offer different threat models and performance trade-offs.
- A data-encoding methodology that allows homomorphic aggregations for both quantitative variables (i.e., data that take numerical values) and categorical data (i.e., data grouped into discrete classes), which enables a broad range of privacy-preserving studies.

Roadmap: The rest of the paper is organized as follows: Following a preliminary discussion on the Paillier cryptosystem, its threshold variant, and commitment schemes (section 2), in section 3 we present an overview of our protocols, our threat models, and two applications. In Section 4, we present a new commitment scheme with multiplicative homomorphism, while in section 5 we elaborate on our two protocols called Masquerade and zk-Masquerade. Next, in section 6, we extend our protocols to support categorical studies and in section 7 we discuss the security of our approach. Sections 8 and 9 evaluate the performance of our schemes and compare them with related works. Finally, our concluding remarks appear in section 10.

2 Cryptography Background

2.1 Paillier Cryptosystem

Homomorphic encryption (HE) allows performing certain mathematical operations on encrypted data that correspond to applying related operations on plaintexts, *without decrypting*. In this paper, we employ Paillier's partially HE (PHE) scheme that supports the addition of encrypted messages [44]. Paillier uses a public/private key pair and supports probabilistic encryption (paired with deterministic decryption); its additive homomorphic property can be summarized as follows: Given two ciphertexts $c_1 = \mathsf{Enc}_{pk}(m_1)$ and $c_2 = \mathsf{Enc}_{pk}(m_2)$, one can compute $c = \mathsf{Enc}_{pk}(m_1 + m_2)$; we omit the randomness for clarity. Below we outline its basic operations.

Key Generation: Let N be the product of two large primes p and q, such that $\mathsf{GCD}(pq, (p-1)(q-1)) = 1$ and $\lambda = \mathsf{LCM}(p-1, q-1)$. Select a random generator $g \overset{\$}{\leftarrow} \mathbb{Z}_{N^2}^{\times}$ and ensure that N divides the order of g by checking if $\mu = (L(g^{\lambda} \bmod N^2))^{-1} \bmod N$ exists, where $L(u) = {}^{(u-1)}/_{N}$. The public key is $pk = (N, g)$ and the secret key is $sk = (\lambda, \mu)$.

Encryption: Select randomness $\rho \overset{\$}{\leftarrow} \mathbb{Z}_N^{\times}$ such that $\mathsf{GCD}(\rho, N) = 1$. A ciphertext $c \in \mathbb{Z}_{N^2}^{\times}$ is defined as $c = \mathsf{Enc}_{pk}(m, \rho) = g^m \rho^N \bmod N^2$ for plaintext $m \in \mathbb{Z}_N$.

Decryption: An inverse mapping from $\mathbb{Z}_{N^2}^{\times}$ to $\mathbb{Z}_N$ is defined as $m = \mathsf{Dec}_{sk}(c) = L(c^{\lambda} \bmod N^2) \cdot \mu \bmod N$.

Homomorphic Operations: A notable homomorphic property of the Pailier cryptosystem enables the product of two ciphertexts to decrypt to the sum of their corresponding plaintexts:
$\mathsf{Enc}_{pk}(m_1, \rho_1) \cdot \mathsf{Enc}_{pk}(m_2, \rho_2) = (g^{m_1}{\rho_1}^N) \cdot (g^{m_2}{\rho_2}^N) \bmod N^2 = g^{m_1+m_2}(\rho_1\rho_2)^N \bmod N^2$, which equals $\mathsf{Enc}_{pk}(m_1 + m_2, \rho_1\rho_2)$. In general, the product of ℓ ciphertexts is equivalent to the encryption of the summation of the ℓ corresponding plaintexts (we omit the randomness ρ):

$$\prod_{i=1}^{\ell} \mathsf{Enc}_{pk}(m_i) = \mathsf{Enc}_{pk}\left(\sum_{i=1}^{\ell} m_i\right) \bmod N^2. \tag{1}$$

Finally, the Paillier cryptosystem allows the multiplication of a ciphertext with a constant, which results in the encryption of the product of the plaintext with the constant, as follows:

$$\mathsf{Enc}_{pk}(m_1, \rho)^{m_2} = \left(g^{m_1}\rho^N\right)^{m_2} \bmod N^2 = g^{m_1 \cdot m_2}\rho^{N \cdot m_2} \bmod N^2 = \mathsf{Enc}_{pk}(m_1 m_2, \rho^{m_2}). \tag{2}$$

2.2 Threshold Paillier

The authors of [23] and [17] proposed threshold versions of Paillier's cryptosystem, in which, a pre-determined number of parties need to collaborate to decrypt a ciphertext jointly. For simplicity, we focus on the case where the number of parties is two, and they both need to agree to perform the decryption. More specifically, we focus on the malicious two-party setting with a dishonest majority as described in [28]. Below, we offer a brief description of the protocol; the full protocols appear in [28].

Distributed Key Generation: The goal of the two parties is to jointly generate a shared Paillier decryption key. They first generate an RSA composite N without disclosing any information about its factorization to each other; to do that, the two parties generate ElGamal and Paillier keys, prove the correctness of their public keys, and exchange them. The ElGamal keys are used for encryption and the (non-distributed) Paillier keys are used for commitments. Next, the parties generate prime candidates for the potential composite, which they run through a distributed biprimality test (i.e., both numbers are primes) for checking whether the generated composite is a product of exactly two primes of an appropriate length. During this step, each party generates an ElGamal encryption of a random prime and proves knowledge of plaintext for the corresponding ciphertexts; then, they both check in zero-knowledge if the selected primes suffice, or they repeat the candidate generation step. Next, they generate secret key shares of $\phi(N)$ as $[\![\phi(N)]\!]_0 = N - p_0 - q_0 + 1$ and $[\![\phi(N)]\!]_1 = -p_1 - q_1$ and finally they generate additive decryption keys $[\![sk]\!]_0$ and $[\![sk]\!]_1$ such that $sk = [\![sk]\!]_0 + [\![sk]\!]_1$, $sk \equiv 0 \mod \phi(N)$ and also $sk \equiv 1 \mod N$.

Encryption & Homomorphic Operations: Same as in the non-threshold variant.
Distributed Decryption: Given a ciphertext c and a public Paillier key $N = pq$ with unknown factorization, secret key shares $[\![sk]\!]_0$ and $[\![sk]\!]_1$, and ElGamal commitments to the key shares, each party raises c to its share of the secret key $[\![sk]\!]_0$ or $[\![sk]\!]_1$. Then, they prove that the decryption has been computed correctly using the commitments of the shares and multiply them to receive the final plaintext. This essentially provides a proof of correct decryption. By default, one party receives the decryption but the protocol can be executed twice so both receive the decryption and proofs [28].

2.3 Commitment Schemes

A secure commitment scheme uses an algorithm Com to enable a sender $\mathcal{S}$ to commit to a value x while keeping it *hidden*. $\mathcal{S}$ publishes a value $c = \mathsf{Com}(x, r)$ that was generated using randomness r; later, $\mathcal{S}$ can open the commitment by disclosing x, r. Com *binds* $\mathcal{S}$ to the hidden value x and offers provable guarantees to a receiver $\mathcal{R}$ that the value revealed later was the same to the one $\mathcal{S}$ originally committed to when c was published. Early commitment schemes proved that it is possible for two parties to play a card game via the telephone without having to trust each other [50].

For any commitment scheme to be secure, it must satisfy two basic properties: *binding* and *hiding*. The first guarantees that upon committing to a secret value x, $\mathcal{S}$ cannot change x later. Formally, for all non-uniform probabilistic polynomial-time (PPT) algorithms that output (x_1, r_1) and (x_2, r_2), the probability that $\mathsf{Com}(x_1, r_1) = \mathsf{Com}(x_2, r_2)$ with $x_1 \neq x_2$, is negligible. The second property requires that c should not reveal any information about x before opening, that is, for all non-uniform PPT algorithms, the probability of extracting any information about x from c should be negligible. The Pedersen commitment scheme features an additive homomorphic property so that for messages x_1 and x_2 with blinding factors r_1 and r_2 we have: $\mathsf{Com}(x_1, r_1) \cdot \mathsf{Com}(x_2, r_2) = \mathsf{Com}(x_1 + x_2, r_1 + r_2)$ [45]. In this work, we define a new multiplicative commitment scheme (section 4) that enables computational integrity for Masquerade.

2.4 Zero-knowledge Proofs (ZKPs)

A ZKP allows a *prover* $\mathcal{P}$ to assert the truth of a statement to a *verifier* $\mathcal{V}$ without revealing any further information [27]. Succinct Non-interactive/Transparent Arguments of Knowledge (SNARKs/STARKs) [24,7,40] focus on proving general-purpose statements and may incur high overheads compared to bespoke solutions such as [15,39,38]. At a high-level, ZKPs must satisfy the following properties:

- **Correctness:** If $\mathcal{P}$ is honest, $\mathcal{V}$ will accept the proof.

- **Soundness:** If $\mathcal{V}$ is honest, no cheating $\mathcal{P}$ can convince $\mathcal{V}$ to accept a false proof, except with negligible probability.
- **Zero-knowledge:** If both $\mathcal{P}$ and $\mathcal{V}$ are honest, then the proof does not reveal anything about $\mathcal{P}$'s private statement, except the fact that it is true.

2.5 Fiat-Shamir Heuristic

The *Fiat-Shamir heuristic* eliminates the interaction in public coin proof-of-knowledge protocols and makes them non-interactive [22] by relying on the random oracle (RO) model [6] to substitute the random challenge generated by $\mathcal{V}$ with a challenge that is the output of a hash function over the previous protocol messages. The sequence of messages (and their commitments) in the protocol are referred to as *transcript*. Modeling the hash function as a RO ensures the unpredictability of $\mathcal{V}$, and public coins are obtained from the hash digest bits using the transcript as input. This construction is secure as long as the hash function is a RO that prevents $\mathcal{P}$ from guessing its output before the commitments are generated.

3 Our Problem Statement

3.1 Overview

Gathering and aggregating data from multiple parties over the Internet enables powerful analytics. Companies can collect meaningful information about their clients' behavior and demographics, while healthcare researchers can discover more effective treatment methods to accelerate the overall healthcare facility. Such studies correspond to one of two categories based on the type of data being used: The first class includes variables that represent quantities (i.e., *quantitative data*) that take numerical values (such as age and profit). The variables in the second class can take one value from a fixed set of options and are called *categorical data* (such as age group, nationality, and employment status). In a quantitative study, the curator applies a function f directly to the participants' data (e.g., compute the average age of consumers or a weighted arithmetic mean based on electricity cost), while in a categorical study, the curator applies f in each different category to generate a histogram (e.g., find the total number of consumers by nationality). The latter can be depicted as a bar graph that shows the frequency distribution of the participants' responses.

Private Data Aggregation (PDA) computes similar aggregations and frequencies while tolerating a curious curator that has incentives to peek at sensitive data points (e.g., to enable targeted advertisements) [51]. PDA enables many useful applications by assuring the participants that their data will only be used towards the agreed analysis and private data cannot be tracked back to each individual; notably, the curator will never access the sensitive data. However, public verifiability is crucial to enable any external auditor to verify the correctness of the result. In this work, we are interested in defining secure publicly verifiable PDA protocols for *both quantitative and categorical studies*. To motivate our case study, we summarize an example from each of the two scenarios.

Categorical analysis. This class enables individuals to take part in surveys where they choose their replies from fixed sets of answers. For example, course evaluations include questions with categorical data, where each response is a one-hot encoding from a set of possible answers (e.g., *"The instructor used the time effectively."*, with four possible responses *a) Strongly Disagree, b) Disagree, c) Agree, d) Strongly Agree* on a *Likert scale* [34]). These evaluations comprise an essential indicator for faculty and departments to get feedback and improve their classes. In course evaluations, it is crucial that there is no link between the participant and their responses so faculty are not biased towards individuals, and most importantly, individuals are more likely to provide more accurate comments. In this case, public verifiability guarantees to the students that their evaluations were counted and also the students can prove that they only submitted one response.

Quantitative analysis. In this category, individuals can privately report numerical values that will be used to compute a total sum, an average, or a weighted average. A notable example is smart metering, which can be applied to multiple participating households. Smart meters communicate the electrical usage almost in real-time to provide better system monitoring and customer billing than traditional meters. However, there are inherent privacy concerns since fine-grained measurements (e.g., one every 15 minutes) may reveal personal information about the number of people in a household and their activities [13,18]. Our protocols solve this problem by aggregating data over multiple participating households for a given period that would still be sufficient for smart grid operators to perform enhanced monitoring and optimize prices but at the same time will conceal private information. Public verifiability is essential to convince all the households that the result was computed correctly. Our protocols naturally extend this application by computing the weighted arithmetic mean of the participating households, as electricity rates can vary based on the area. This allows participants and communities to compare their electricity consumption based on their (public) area price and gain useful insights about their electricity rates.

3.2 Publicly Verifiable Private Data Aggregation (PVPDA)

Inspired by [5], we extend the notion of PDA to Publicly Verifiable PDA (PVPDA) which allows any auditor that has access to the public messages that were exchanged during the protocol to check the correctness of the output. The goal of a PVPDA functionality is to enable the curator to aggregate the participants' private data and at the same time allow any auditor to verify the protocol's correctness (e.g., by inspecting a bulletin board). Next, the analyst (who initiated the study and is interested in learning the final aggregation) receives the final output along with guarantees that it was computed correctly. Below we discuss two different threat models for PVPDA.

3.3 Threat Model

The different applications supported by our protocols require protection against a variety of threats. We instantiate two protocols for PVPDA by utilizing a helper party which we call a curator, and an analyst that initiates the study and learns the final result. Each protocol introduces a different threat model but both are aligned with the models of secure two-party computation and they assume that the curator and the analyst do not collude. Instantiating our protocol assuming two parties with mutual distrust enables computing the aggregation without exposing sensitive information about the participants' data [53]. We also assume the existence of digital certificates within a public key infrastructure, as well as secure communication channels (e.g., TLS). Below, we delve into the different threat models of Masquerade and zk-Masquerade.
Auditors. Both our protocols allow any auditor to inspect the correctness of the PVPDA protocol. The auditors do not have any input during the protocol execution, and can thus behave arbitrarily. Any party can be an auditor and inspect the public messages on the bulletin board and verify the correctness of the protocol themselves.

3.3.1 Masquerade (homomorphic-commitment-based) Threat Model

Participants. In this protocol, we assume that the participants are verified and are interested in learning the correct final result. Thus, they follow the protocol specification and do not behave maliciously, i.e., they submit valid inputs (i.e., within a pre-agreed range or within a set of known messages). However, we assume that participants may try to learn other participants' private data.

Masquerade is also robust against *participant dropouts* who submit their inputs and stop participating. Additionally, to defend against Sybil attacks that create a large number of illicit clients to influence the result [19], both the curator and the analyst agree on a list of identities. Each participant signs their message and the other participants (including the curator and the analyst) can verify the message signatures to prevent forgeries [1]. Finally, to prevent clients from biasing the result, we allow one private input per client. Our work focuses on protecting data confidentiality, rather than participant anonymity.
Case 1: Malicious Curator, Honest Analyst. A *malicious* curator may have incentives to deviate from the protocol either to extract information about the participants' sensitive data or to disproportionately

influence the result. The curator may try to skip adding some participants' encrypted data from the result or even over-count some others. Thus, it is crucial to be able to detect such malicious behavior and provide public guarantees to the analyst and any auditors about the integrity of the computation. In both our protocols, participants' inputs are protected using a secure probabilistic PHE scheme (our protocol relies on the Paillier cryptosystem with a 2048-bit modulus N). As PHE does not offer native integrity protections, we introduce a novel multiplicative homomorphic commitment scheme (section 4) that provides public verifiability without compromising the participants' privacy. The analyst employs these public commitments to verify the computational integrity of the homomorphic result.
Case 2: Semi-honest Analyst, Honest Curator. We assume a semi-honest analyst that is interested in learning the aggregation results but has incentives to extract information about individuals' data. Similarly to the curator, our goal is to also provide public verifiability that the analyst has followed the protocol specification and enable convincing the participants that the result was decrypted correctly. We consider a bulletin board as an online append-only ledger that is verifiable by anyone (e.g., public auditors) and is maintained by the analyst using standard consensus methods [43] or using a public blockchain network. The analyst posts a proof of correct decryption to the bulletin board providing public verifiability.

3.3.2 zk-Masquerade (zero-knowledge-proofs-based) Threat Model

Malicious Participants. This protocol assumes *malicious* participants that may attempt to disproportionately affect the analysis by encrypting invalid inputs (e.g., a large negative number). To address this threat, we adopt non-interactive ZKP protocols that allow the participants to prove to the curator that the provided encrypted data correspond to a valid message. Both the curator and the analyst verify the participants' proofs and only consider the private inputs for which the ZKP verification is successful. This guarantees that each participant submits a valid input (i.e., within a pre-agreed range) and cannot detect if clients submitted valid but false data (e.g., lie about their age). It is crucial that both the curator and the analyst verify the proofs since one of them may also be malicious and this way we guarantee that the other party catches it. Finally, similarly to Masquerade, this protocol is robust against participant dropouts.
Case 1: Malicious Curator, Honest Analyst. Similarly to Masquerade, zk-Masquerade assumes a *malicious* curator that may have incentives to deviate from the protocol either to extract information about the participants' sensitive data or to disproportionately influence the result. The curator may try to skip adding some participants' encrypted data from the result or even over-count some others. Thus, it is crucial to be able to detect such malicious behavior and provide public guarantees to the analyst and any auditors about the integrity of the computation. In zk-Masquerade, we use threshold encryption in which both the curator and the analyst need to agree in order to decrypt the final result. For that reason, in this protocol, we publish both the ciphertexts and the participants' ZKPs to a bulletin board such that both parties as well as any auditor can verify the correctness of the computation. Since the ciphertexts and the proofs are public, we no longer need the homomorphic commitments.
Case 2: Malicious Analyst, Honest Curator. Lastly, we assume a *malicious* analyst that may deviate from the protocol specification in order to learn individual participants' private data or to change the final aggregation result. Our goal is to also provide public verifiability that the analyst has not cheated, similarly to the curator. Both parties proceed with threshold decryption only if the homomorphic results match. Finally, both the analyst and the curator must post proof of correct decryption to the bulletin board providing public verifiability.

4 Our Multiplicative Commitment Scheme

Here we propose a new commitment scheme based on RSA [46] with multiplicative homomorphism. Recall from section 2.3 that the Pedersen commitment scheme has additive homomorphism [45]. Let N be a public RSA modulus such that $N = pq$, where p, q are secret primes, let e be a public prime that satisfies $e > N^2$

and $\mathsf{GCD}(e, \phi(N^2)) = 1$, and let g_m be a public element of high order in $\mathbb{Z}^{\times}_{N^2}$. To bind to a secret message $m \in \mathbb{Z}^{\times}_{N^2}$, the sender $\mathcal{S}$ generates a random secret $r \in \mathbb{Z}^{\times}_{N^2}$ and calculates c as:

$$c = \mathsf{Com}(m, r) = m^e g_m^r \bmod N^2. \tag{3}$$

Assuming that the RSA problem with modulus N^2 and public exponent e is hard to invert, this commitment scheme (intuitively) satisfies the hiding and binding requirements. During the reveal phase, both the secret message m and the randomness r are published, and the verifier checks if they produce the same commitment as c.

Theorem 1. *The scheme with multiplicative homomorphism from Eq. 3 satisfies the hiding and binding properties.*

Proof. First, we show that the scheme is perfectly hiding. Sender $\mathcal{S}$ confirms that $N^2 < e$ and e is a prime, so that $\mathsf{GCD}(e, \phi(N^2)) = 1$ and e is invertible $\bmod \phi(N^2)$. We will show that $c = m^e g_m^r \bmod N^2$ is indistinguishable from random values in $\mathbb{Z}^{\times}_{N^2}$: Since g_m is an element of high order in $\mathbb{Z}^{\times}_{N^2}$ (by construction) and r is uniformly random in $\mathbb{Z}^{\times}_{N^2}$, then $g_m^r \bmod N^2$ is uniformly distributed in a subgroup of $\mathbb{Z}^{\times}_{N^2}$ with order $|g_m|$. Likewise, the product $m^e g_m^r \bmod N^2$ is also uniformly distributed in a subgroup of $\mathbb{Z}^{\times}_{N^2}$ with order $|g_m|$, so it is indistinguishable from a random element of $\mathbb{Z}^{\times}_{N^2}$ as the order of g_m is high; thus, no information can be learned about m from c.

Next, we prove that our scheme is computationally binding under the RSA assumption (i.e., the problem of computing e^{th} roots modulo a composite N^2 is computationally hard). Our goal is to show that if a $\mathcal{S}$ can find two different messages m, m' with corresponding randomness r, r' that collide to the same commitment $c = c'$, this would imply we can easily compute e^{th} roots modulo N^2 so that the RSA problem would be easy (contradiction). To show this, we assume it is possible to find $m \neq m'$ and $r \neq r'$, so that $c = m^e {g_m}^r = c' = {m'}^e {g_m}^{r'} \bmod N^2$. In this case, $m^e {g_m}^r = {m'}^e {g_m}^{r'} \bmod N^2 \iff g_m^{r-r'} = ({}^{m'}/_{m})^e \bmod N^2$.

Using the fact that $r, r' \in \mathbb{Z}^{\times}_{N^2}$ and e is a prime larger than N^2, it holds that $r - r' < e$ and also $\mathsf{GCD}(r - r', e) = 1$. Thus, from Bèzout's identity, there exist integers A, B (that can be computed in polynomial time using the extended Euclidean algorithm), so that $A(r - r') + Be = \mathsf{GCD}(r - r', e) = 1$. Then, we have:

$$\begin{aligned} g_m^1 = g_m^{A(r-r')+Be} &= \left(g_m^{(r-r')}\right)^A g_m^{Be} \bmod N^2 \\ &= \left(({}^{m'}/_{m})^e\right)^A \left(g_m^B\right)^e \bmod N^2 \\ &= \left(({}^{m'}/_{m})^A\, g_m^B\right)^e \bmod N^2, \end{aligned}$$

and thus, $g_m^{1/e} = ({}^{m'}/_{m})^A\, g_m^B \bmod N^2$.

The above steps can efficiently compute the e^{th} root of $g_m \bmod N^2$, which contradicts the hardness assumption of RSA for a sufficiently large composite N^2. Therefore, the original assumption that $\mathcal{S}$ can bind a commitment c to more than one message is false, as expected.

From Eq. 3 we observe that our commitment has a multiplicative homomorphic property; we can compute a commitment for the multiplication of two messages m_1 and m_2 by knowing the individual commitments to these two messages. To open this new commitment, the randomness r_1 and r_2 used for each individual commitment should be added. Note that this is distributed identically to a fresh commitment to $m_1 \cdot m_2$. More formally:

$$\begin{aligned} \mathsf{Com}(m_1, r_1) \cdot \mathsf{Com}(m_2, r_2) \bmod N^2 &= \left({m_1}^e \cdot g_m^{r_1}\right) \cdot \left({m_2}^e \cdot g_m^{r_2}\right) \bmod N^2 \\ &= \left((m_1 \cdot m_2)^e \cdot g_m^{r_1+r_2}\right) \bmod N^2 \\ &= \mathsf{Com}(m_1 \cdot m_2, r_1 + r_2), \end{aligned}$$

which can be extended to ℓ messages as follows:

$$\textstyle\prod_{i=1}^{\ell} \mathsf{Com}(m_i, r_i) = \mathsf{Com}(\prod_{i=1}^{\ell} m_i, \sum_{i=1}^{\ell} r_i) \bmod N^2. \tag{4}$$

In the next section, we describe how the analyst in Masquerade leverages this property to generate a commitment over the encrypted sum without accessing the individual encrypted messages, relying solely on the public commitments of each encrypted message. By the *hiding* property (Theorem 1), each commitment to a message *does not reveal anything* to the analyst about the message itself. Additionally, the *binding* property guarantees the *computational integrity* of ciphertext aggregation, since an error in the aggregation result would prevent an auditor from successfully opening the aggregate commitment. Masquerade offers public verifiability by publishing all the commitments to a ledger and enabling any party to be able to audit them.

5 Publicly Verifiable Private Data Aggregation (PVPDA) Protocols

In this section, we describe Masquerade and zk-Masquerade, our cryptographic constructions for secure PVPDA over the Internet. Our protocols support quantitative data *by design* and we extend our constructions to support categorical data (Section 6).

5.1 Masquerade: Homomorphic Commitments on Homomorphic Data

Each participant (a) locally encrypts their private data using Paillier, (b) publishes in a bulletin board a multiplicative commitment on the ciphertext, and (c) sends their encrypted data to the curator. The curator first verifies the correctness of the commitment, and then homomorphically adds the participant's data to an encrypted accumulator if the commitment is correct; otherwise, the curator rejects the participant's data. As soon as all participants have sent their data, the curator publishes the encrypted sum.

The analyst audits the protocol by accumulating the homomorphic commitments from the ledger and verifying that they open with the encrypted sum. Since the commitments and the final sum are public, any auditor (or even a participant) can repeat the same process and be convinced that their private data were included in the computation, as well as verify the correctness of the encrypted aggregation. We remark that the need for public verifiability remains critical in real-world applications. Masquerade explicitly enables the participants and the analyst to verify that no errors were introduced in the aggregation result (e.g., by omitted inputs or over-counting) since the analyst asserts that the aggregated commitment opens successfully to the encrypted sum. Upon verifying that the curator's result is correct, the analyst decrypts and publishes the result along with a proof of correct decryption of the final result.

5.1.1 Benefits

A crucial property of our scheme is that it does not require interactive communication between the participants and the curator. Each participant generates and sends their encrypted data and their commitment locally and then disconnects from the protocol. Since our commitment scheme does not reveal anything about the participants' data, any public ledger that does not rely on a trusted setup is sufficient.

Furthermore, if some clients deliberately stop participating in the protocol (i.e., participant *dropout*), our scheme remains resilient since we do not rely on any user secret for decryption of the final result. In contrast, in earlier PDA protocols that are based on secret-sharing a single client can easily corrupt the protocol by just withdrawing their participation. We compare with related works in section 9.

5.1.2 Protocol

We now discuss our PVPDA protocol, depicted in Fig. 2. First, the analyst generates a Paillier key-pair sk, pk as well as the random prime e and the maximum-order element g_m required for our commitment scheme (section 4) and publishes the pk (including modulus N^2), e, and g_m. Notably, we use the same N^2 both as the public *commitment* modulus and as Paillier's *encryption* modulus. Then, each participant probabilistically encrypts their private message using Paillier and commits to the generated ciphertext (recall, these commitments do not reveal anything about either the ciphertext or the plaintext). Then, the participant sends their ciphertext to the curator along with the randomness used for the commitment and publishes the commitment value to the ledger. The curator calculates the homomorphic addition of the participants' messages, adds the individual commitment randomness r_i, and publishes the Paillier ciphertext along with the sum of all r_i's to the bulletin board.

Public Verifiability for Curator. Finally, the analyst verifies the integrity of the homomorphic result using the formula of Eq. 4 to ensure the multiplication of the client commitments along with the aggregated randomness equals the commitment over the encrypted sum.

```
1: ① Key-Generation                                                        (Analyst)
2:    $(sk, pk) \xleftarrow{\$} \mathsf{Gen}(1^\kappa)$ ▷ Paillier key generation. Security level $\kappa$ in bits.
3:    $N = p \cdot q$ using $p, q$ from $sk$
4:    $N^2$ is used both for Paillier and commitment
5:    Randomly select prime $e$, s.t. $e > N^2$.
6:    $g_m \xleftarrow{\$}$ maximal order $\mathbb{Z}^\times_{N^2}$
7:    Publish to the ledger: $pk$, $g_m$, $e$
8:    Private Analyst Output: $sk$
```

```
1: ② Encrypt-Prove-Commit                                                  (Participant)
2:    $ct_i \leftarrow \mathsf{Enc}_{pk}(m_i, \rho_i)$                      ▷ $\rho_i \xleftarrow{\$} \mathbb{Z}^\times_{N^2}$
3:    $c_i = \mathsf{Com}(ct_i, r_i)$                                       ▷ $r_i \xleftarrow{\$} \mathbb{Z}^\times_{N^2}$
4:    Publish to the ledger: $c_i$
5:    Privately transmit to the Curator: $ct_i$, $r_i$
```

```
1: ③ Aggregate                                                             (Curator)
2:    Assert $c_i \overset{?}{=} \mathsf{Com}(ct_i, r_i)$                   ▷ Verify the commitment
3:    $ct_{sum} \leftarrow \prod_{i=1}^{\ell} ct_i \bmod N^2$               ▷ Aggregate ciphertexts
4:    $r_{sum} \leftarrow \sum_{i=1}^{\ell} r_i$                            ▷ Aggregate randomness
5:    Publish to the ledger $ct_{sum}$, $r_{sum}$
```

```
1: ④ Audit-Decrypt                                                         (Analyst)
2:    Assert $\prod_{i=1}^{\ell} c_i \bmod N^2 \overset{?}{=} \mathsf{Com}(ct_{sum}, r_{sum})$ ▷ Verify commits
3:    $sum \leftarrow \mathsf{Dec}_{sk}(ct_{sum})$                          ▷ Decrypt result
4:    $ct'_{sum} \leftarrow \mathsf{Enc}_{pk}(sum, \rho)$                   ▷ $\rho, \rho' \xleftarrow{\$} \mathbb{Z}^\times_{N^2}$
5:    $ct_0 \leftarrow ct_{sum} * (ct'_{sum})^{-1}$                         ▷ Encryption of zero
6:    $\pi_0 \xleftarrow{\$} \mathsf{Prove}(ct_0, \mathsf{Enc}_{pk}(0, \rho'))$ ▷ Prove $ct_0 = \mathsf{Enc}_{pk}(0, \rho')$
7:    Publish to the ledger $sum$, $ct_0$, $ct'_{sum}$, $\rho$, $\rho'$, $\pi_0$
```

```
1: ⑤ Public Verification                                                   (Auditor)
2:    Assert $\prod_{i=1}^{\ell} c_i \bmod N^2 \overset{?}{=} \mathsf{Com}(ct_{sum}, r_{sum})$ ▷ Verify commits
3:    Assert $ct'_{sum} \overset{?}{=} \mathsf{Enc}_{pk}(sum, \rho)$        ▷ Verify $ct'_{sum}$
4:    Assert $ct_0 \overset{?}{=} ct_{sum} * (ct'_{sum})^{-1}$              ▷ Verify $ct_0$
5:    Accept/Reject $\leftarrow \mathsf{Verify}(pk, \pi_0, ct_0)$
```

Fig. 2. Masquerade protocol for PVPDA. In the top right (inside the parentheses), we indicate which party runs each algorithm. Com stands for the commitment algorithm.

For example, given two participant messages m_1 and m_2, the curator receives two ciphertexts $\mathsf{Enc}_{pk}(m_1)$ and $\mathsf{Enc}_{pk}(m_2)$ and computes $\mathsf{Enc}_{pk}(m_1) \cdot \mathsf{Enc}_{pk}(m_2) \bmod N^2$ (here, we omit the randomness of Paillier encryption). Moreover, the curator sums the *commitments' randomness* $(r_1 + r_2)$. The analyst reads the published commitments to these two ciphertexts and can verify the commitment to the encrypted sum without having access to the individual ciphertexts as:

$$\begin{aligned}
&\mathsf{Com}\big(\mathsf{Enc}_{pk}(m_1), r_1\big) \cdot \mathsf{Com}\big(\mathsf{Enc}_{pk}(m_2), r_2\big) \bmod N^2 \\
&= \big(\mathsf{Enc}_{pk}(m_1)^e \cdot g_m^{r_1}\big) \cdot \big(\mathsf{Enc}_{pk}(m_2)^e \cdot g_m^{r_2}\big) \bmod N^2 = \big(\mathsf{Enc}_{pk}(m_1) \cdot \mathsf{Enc}_{pk}(m_2)\big)^e \cdot \big(g_m^{r_1+r_2}\big) \bmod N^2 \\
&= \mathsf{Com}\big(\mathsf{Enc}_{pk}(m_1) \cdot \mathsf{Enc}_{pk}(m_2) \bmod N^2, r_1 + r_2\big) = \mathsf{Com}\big(\mathsf{Enc}_{pk}(m_1 + m_2), r_1 + r_2\big).
\end{aligned}$$

Therefore, the analyst and any auditor can compute and open the commitment for the encrypted sum received from the curator by using the accumulated randomness. In our implementation, the analyst acts as an auditor, but we also enable any party to further verify the protocol by inspecting the public ledger. With a successful commitment opening, an auditor is assured that no errors occurred during the aggregation.

Public Verifiability for Analyst. Masquerade enables any auditor and the participants to audit not only the curator but also the analyst, efficiently verifying that the published sum is the correct decryption of the ciphertext that the curator published to the ledger. We also adopt the ZKP protocol for proving ciphertext equality over N^{s+1} modulus with $s = 1$ from [17] and we apply the Fiat-Shamir heuristic in the random oracle model to convert the ZKP into a non-interactive variant.

Let ct_{sum} be the encrypted result and sum be the decryption published by the analyst. In this case, we can provide public verifiability that the analyst decrypted and published the correct result: The analyst first encrypts sum using randomness ρ into ct'_{sum}, and then computes a ciphertext ct_0, which is the homomorphic result of ct_{sum} minus ct'_{sum}, and posts ct_0, ct'_{sum}, along with the randomness ρ to the ledger. By construction, if ct_{sum} is an encryption of sum, then ct_0 is an encryption of zero. Finally, the analyst uses the aforementioned non-interactive ZKP (Appendix A, Protocol 1) and offers public verifiability that ct_0 is an encryption of zero, which means that ct_{sum} is an encryption of sum. Notably, it is not possible for the analyst to cheat and publish a wrong sum, as ct_0 would not be an encryption of zero, which breaks the soundness of the ZKP [17].

5.1.3 Weighted Arithmetic Mean

Next, we extend this core protocol to support computing the weighted arithmetic mean based on both the participants' private inputs and some public weights. From Eq. 2, it follows that the curator can multiply encrypted participant inputs with weight values $w_i \in \mathbb{Z}_N$, where $i \in [1, \ell]$. Based on the previous example, for ciphertexts $\mathsf{Enc}_{pk}(m_1)$ and $\mathsf{Enc}_{pk}(m_2)$, the curator computes $\mathsf{Enc}_{pk}(m_1)^{w_1} \cdot \mathsf{Enc}_{pk}(m_2)^{w_2} \bmod N^2 = \mathsf{Enc}_{pk}(w_1 m_1) \cdot \mathsf{Enc}_{pk}(w_2 m_2) \bmod N^2$, and also sums the commitments' randomness. Finally, the analyst incorporates the weights in the commitments received from the participants by applying the following formula: ${w_i}^e \cdot \mathsf{Com}\big(\mathsf{Enc}_{pk}(m_i), r_i\big) = \big(w_i \cdot \mathsf{Enc}_{pk}(m_i)\big)^e g_m^{r_i} = \mathsf{Com}\big(\mathsf{Enc}_{pk}(w_i m_i), r_i\big)$.

5.2 zk-Masquerade: Thwarting Malicious Participants with ZKPs

We now introduce our zero-knowledge variant of Masquerade, which has a more strict threat model and relies on non-interactive ZKPs instead of multiplicative commitments. Each participant (a) encrypts their private data locally using Paillier, (b) generates a non-interactive ZKP that enables its recipient to verify that the ciphertext is the encryption of a message that lies in a range of valid messages, and (c) publishes their encrypted data along with a proof to the bulletin board. Notably, our approach supports arbitrary ranges based on the target application. Moreover, a core difference from Masquerade is that zk-Masquerade uses threshold Paillier, with the curator and the analyst holding shares of the decryption key. Thus, as long as at least one of them is honest, the participants can publish their encrypted data.

In our threat model, we assume that one computing party can be malicious (either only the curator or the analyst). Therefore, the protocol asks both to verify the validity of the ZKPs and then homomorphically add the participant's data to an encrypted accumulator if the proof is correct; otherwise, the participant's data is rejected. As soon as all participants have sent their inputs, the encrypted sum is published and both the analyst and the curator verify that they computed the same sum. This can be followed by any auditor that also wants to verify the computation. The analyst and the curator proceed with threshold decryption only if the homomorphic sums match.

5.2.1 Protocol

We now discuss our zk-Masquerade protocol, depicted in Fig. 3. First, the analyst and the curator jointly generate a threshold Paillier key-pair $(d_A, d_C), pk$ with unknown factorization and they publish the pk [28]. Then, each participant probabilistically encrypts their private message using Paillier and creates a ZKP proving that their private plaintext that corresponds to their public ciphertext is well-formed without revealing the plaintext. Then, each participant publishes both the ciphertext and the ZKP to the ledger. The curator first checks the proof and then accumulates ciphertext if the ZKP verification was successful, otherwise, the curator skips the malicious client. Finally, the curator helps the analyst to decrypt the encrypted result and provide proofs of correct decryption, as in [28].

As participants contribute their data encrypted, the curator has no way of detecting if a malicious client performs a data pollution attack by sending an encryption of an overwhelmingly large number. Our zk-Masquerade protocol limits each participant's influence in the aggregate results by restricting the values they can input into the protocol. Specifically, to mitigate such attacks, we incorporate *non-interactive* ZKP protocols, namely a *range proof* and a *set-membership proof.* The former enables the participants to prove that their Paillier ciphertext encrypts a message m that lies within a valid range, i.e., $m \in [0, \ell]$ [35,10], while

```
1: ① KEY-GENERATION                                         (Analyst & Curator)
2:     $((d_A, d_C), pk) \xleftarrow{\$} \mathsf{Gen}(1^\kappa)$   ▷ Threshold Paillier distributed
   key generation. Security level $\kappa$ in bits.
3:     $N = p \cdot q$ with unknown factorization.
4:     Publish to the ledger $pk$
5:     Private Analyst Output $d_A$
6:     Private Curator Output $d_C$
```

```
1: ② ENCRYPT-PROVE-COMMIT                                   (Participant)
2:     $ct_i \leftarrow \mathsf{Enc}_{pk}(m_i, \rho_i)$                ▷ $\rho_i \xleftarrow{\$} \mathbb{Z}^\times_{N^2}$
3:     $\pi_i \xleftarrow{\$} \mathsf{Prove}(ct_i, m_i)$      ▷ Prove $ct_i$ is an encryption of $m_i$
4:     Publish to the ledger $ct_i$, $\pi_i$
```

```
1: ③ AGGREGATE                                              (Curator)
2:     Accept/Reject ← $\mathsf{Verify}(pk, \pi_i, ct_i)$       ▷ Verify the proof
3:     $ct_{sum} \leftarrow \prod_{i=1}^{\ell} ct_i \bmod N^2$      ▷ Aggregate ciphertexts
4:     Publish to the ledger $ct_{sum}$
```

```
1: ④ AUDIT-DECRYPT                                          (Analyst & Curator)
2:     Analyst: Accept/Reject ← $\mathsf{Verify}(pk, \pi_i, ct_i)$   ▷ Verify proofs
   for $i \in \ell$.
3:     $ct'_{sum} \leftarrow \prod_{i=1}^{\ell} ct_i \bmod N^2$      ▷ Aggregate ciphertexts for
   which the proofs were accepted
4:     Analyst checks: $ct'_{sum} \overset{?}{=} ct_{sum}$           ▷ Verify $ct_{sum}$
5:     Analyst: $ct_A \leftarrow (ct_{sum})^{d_A} \bmod N^2$ and prove correct
   decryption using commitment to $d_A$.
6:     Curator: $ct_C \leftarrow (ct_{sum})^{d_C} \bmod N^2$ and prove correct
   decryption using commitment to $d_C$.
7:     Curator: Send $ct_C$ and proof of correct decryption to an-
   alyst.
8:     Both: Check the proofs of correct decryption. Abort if ver-
   ification fails.
9:     Analyst: $sum = ((ct_A \cdot ct_C) \bmod N^2 - 1)/N$
10:    Publish to the ledger $sum$
```

```
1: ⑤ PUBLIC VERIFICATION                                    (Auditor)
2:     Accept/Reject ← $\mathsf{Verify}(pk, \pi_i, ct_i)$   ▷ Verify proofs for $i \in \ell$.
3:     $ct'_{sum} \leftarrow \prod_{i=1}^{\ell} ct_i \bmod N^2$      ▷ Aggregate ciphertexts for
   which the proofs were accepted
4:     Assert $ct'_{sum} \overset{?}{=} ct_{sum}$                   ▷ Verify $ct_{sum}$
5:     Check proofs of correct decryption.
```

Fig. 3. zk-Masquerade protocol for PVPDA. In the top right (inside the parentheses), we indicate which party runs each algorithm. Prove and Verify stand for the prover and the verifier algorithms, respectively, which are discussed in Section 5.2.

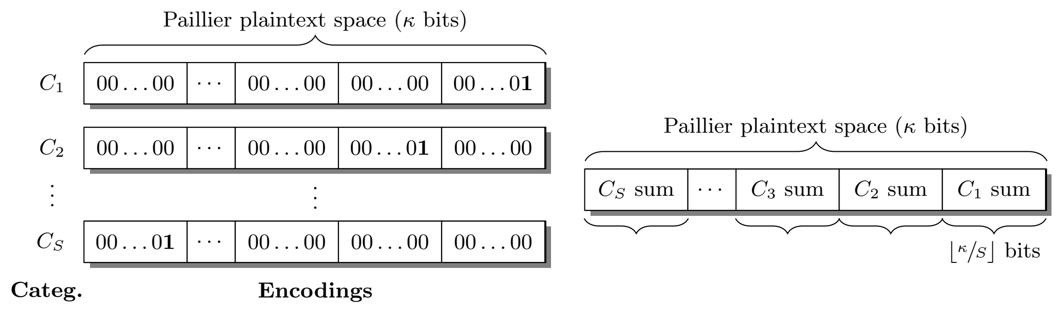
the latter proves that m lies within a set of messages, i.e., $m \in \{m_1, m_2, \dots, m_k\}$ [4,17]. In both cases, the client manages to convince the curator by only showing the Paillier encryption of m without leaking any information about m. We include these protocols for completeness in Appendix A (Protocols 2 and 3).

As discussed in Section 2.4, ZKP protocols must satisfy three basic properties: *completeness*, *soundness*, and *zero-knowledge*. Completeness guarantees that an honest participant ($\mathcal{P}$) can always convince the curator ($\mathcal{V}$) about a valid statement. Soundness guarantees that if the curator is honest and the participant's private data are not well-formed (i.e., are not within a pre-specified range or a set of public messages), then the curator will reject the proof with overwhelming probability. We formalize the soundness by introducing a security parameter t: the probability of an adversary convincing an honest verifier in range proofs is 2^{-t}, and in set-membership proofs it is A^{-t}, where A is the size of the randomness used for the protocol. Finally, zero-knowledge guarantees that the proof does not reveal anything about the private participant's data, except that they are well-formed.

We utilize the zero-knowledge range proof protocol for quantitative aggregations, while the set-membership proof is used to assert that the participants select a message from some predefined categories (described in section 6). We instantiate these ZKP protocols as public coins in the random oracle model [6] via the Fiat-Shamir heuristic (discussed in Section 2.5) to eliminate the interaction and transform the interactive proofs to their *non-interactive* variants [22]. Therefore, the curator verifies the correctness of the proof for the ciphertext sent by each participant and homomorphically increments the sum if and only if the proof was accepted. In Fig. 3, the proof for the participant i is depicted as π_i, whereas the prover and verifier algorithms are Prove and Verify, respectively.

6 Enabling Categorical Data Aggregation

Both our protocols naturally support the aggregation of quantitative data. On the other hand, categorical variables represent types of data that may be divided into groups, where each group should be encoded and accumulated separately. For instance, an aggregation based on four age groups (e.g., "infant", "child", "adolescent", and "adult") requires four counters, one for each category. In categorical data, each participant's influence is limited since they can contribute at most one to the sum of a category. For example, if the attribute type is "age group", each participant may only contribute to one of the categories and increment that counter (e.g., "adult").


Paillier plaintext space (κ bits)
C_1
00...00 | ··· | 00...00 | 00...00 | 00...01
C_2
00...00 | ··· | 00...00 | 00...01 | 00...00
C_S
00...01 | ··· | 00...00 | 00...00 | 00...00
Categ.
Encodings
Paillier plaintext space (κ bits)
C_S sum | ··· | C_3 sum | C_2 sum | C_1 sum
$\lfloor \kappa/S \rfloor$ bits

(a) Encodings for S categories. (b) Accumulated sums for each category.

Fig. 4. Histograms Overview. We divide the plaintext space into different sections, each of which corresponds to a different category. Homomorphic addition of ciphertexts results in the summation of the bits from each category.

Taking these two requirements into account, we design a specialized encoding to incorporate different categories into one encrypted message. A possible solution for categorical variables would be to use multiple invocations of our quantitative PVPDA protocols so that every invocation would aggregate one category. Each participant could contribute an encryption of either a "one" or a "zero", depending on their selection, and also prove in zero-knowledge that the submitted ciphertext is an encryption of a binary number. Unfortunately, in cases where the response should be "one-hot encoded" (e.g., "age group"), this solution is not sufficient, as it enables malicious participants to contribute to more than one category and tamper with the aggregation result. Although in each invocation they could submit a valid encryption and a proof, the set of their responses could be invalid (e.g., submit multiple encryptions of True).

We introduce an encoding that enforces the participants to select 1-out-of-S categories using a single message that encodes both the selected category as well as the remaining $S-1$ non-selected categories. We encode each category as a power $2^{\lfloor \kappa/S \rfloor}$, where κ are the bits of the plaintext: specifically, the first category is represented as $(2^{\lfloor \kappa/S \rfloor})^0 = 1$, the second category is represented as $(2^{\lfloor \kappa/S \rfloor})^1 = 2^{\lfloor \kappa/S \rfloor}$, and so on. In Fig. 4 (a) we illustrate these encodings. We remark that performing homomorphic addition between ciphertexts from the same category will increment the $\lfloor \kappa/S \rfloor$-bit counter for the specific category, and leave the counters for the $S-1$ remaining categories unmodified. The accumulations for each selected attribute create a one-dimensional histogram with S categories, as illustrated in Fig. 4 (b). We complement our zk-Masquerade protocol with the non-interactive set-membership proof described in section 5.2 to guarantee benevolent client behavior; in this case, each participant encrypts one of S possible messages (i.e., a power $2^{\lfloor \kappa/S \rfloor}$) and creates a zero-knowledge set membership proof that the ciphertext is the encryption of one of these powers.

Using the same number of plaintext bits, our protocols also support responses that enable the participants to select multiple categories at the same time, if those are valid answers. To support such messages, the set membership proof includes all the possible combinations of valid responses. For instance, a response for both C_1 and C_3 from Fig. 4 (a) would be in the form $00\ldots00 \mid \cdots \mid 00\ldots01 \mid 00\ldots00 \mid 00\ldots01$. Adding the above response with another response for C_1 will result in $00\ldots00 \mid \cdots \mid 00\ldots01 \mid 00\ldots00 \mid 00\ldots10$, which accumulates both responses.

Notably, by having large plaintext sizes our encoding can also support multidimensional histograms. For instance, extending the example of age groups, the analyst can also consider if a participant is a smartphone user. We use these two variables (i.e., "age groups" and "smartphone user") to form a two-dimensional array with all combinations and encode each index in the array into a unique category from Fig. 4 (a). Since "age groups" has four possible values and "smartphone user" has two, the number of categories is $S = 8$. To increase the number of categories and preserve enough bits for each accumulation, a larger plaintext space can be used by increasing κ. This introduces a trade-off between the number of categories and the performance of our protocol since larger modulus sizes may increase performance overheads. Additional examples of realistic Internet applications of Masquerade are shown in Table 1.

Table 1. Example applications of Masquerade including the application type (i.e., numerical or categorical), as well as potential entities for the curator, the analyst, and the participants.

Application	**Application Type**	**Participants**	**Curator**	**Analyst**
Course-evaluations	Numerical/Categorical	Students	US Department of Education, State, etc.	University
Smart-metering	Numerical	Households	AWS	Utility company
Auctions	Categorical	Participants	AWS	Auctioneer
Healthcare	Numerical/Categorical	Patients	Insurance company, federal agency, etc.	Hospital
Banking	Numerical/Categorical	Customers	Federal/state department	Bank

7 Security Analysis

We prove the security of our protocols in the real-ideal world paradigm by describing a simulator Sim that can indistinguishably simulate the view of the honest parties [12]. Let $\mathsf{Real}^{Party}(\{m_i\}_{i\in\ell})$ be a random variable representing the view of $Party \in \{Curator, Analyst, P_i \text{ for } i \in \ell\}$ in the real protocol execution, and ranges over the randomness of the parties.

We define the security of our protocols by comparing what an adversary can do in the real protocol execution versus what it can do in an ideal execution that involves a trusted third party that receives the inputs. In the ideal world, the trusted party is assumed to be honest and computes the functionality on the parties' inputs, and returns the result to each party. An adversary in the ideal world, also called a simulator, that interacts with the functionality must be able to generate a view that is similar to the real-world view. In the real world, an adversary that interacts with the real protocol should not be able to do anything that it could not do in the ideal world (e.g., learn private inputs or alter the expected outputs). In other words, a protocol is secure if it is possible in the ideal world to generate something indistinguishable from the real-world adversary's view.

7.1 Security Analysis of Masquerade

Theorem 2. *Assuming that Eq. 3 is a secure commitment scheme with multiplicative homomorphism and satisfies the hiding and binding properties and the Paillier cryptosystem is semantically secure against chosen-plaintext attacks (IND-CPA), then the Masquerade protocol (Fig. 2) securely implements a PVPDA protocol against a malicious curator or corruption of a semi-honest analyst with semi-honest participants.*

Proof. **Semi-honest Corruption of Participants and Analyst.** We assume that a set $\mathcal{L}' = \{1, \ldots, \ell'\}$ of $\ell' < \ell$ participants are corrupt as well as the analyst. We denote the remaining honest participants as $\mathcal{L} = \{j, \ldots, \ell\}$, where $j = \ell' + 1$. Without loss of generality, we assume that the corrupt participants are the first ℓ' ones. The analyst receives the final encrypted sum as well as the aggregated commitment randomness. Then, the analyst checks that the product of the commitments opens to the commitment of the encrypted sum and the aggregated randomness. Observe that as long as the Paillier encryption is secure, nothing is revealed about each honest participant's individual data points. We define a PPT simulator such that for all participant inputs $\{m_i\}_{i\in\ell}$,

$$\mathsf{Real}^{Analyst,\mathcal{L}'}(\{m_i\}_{i\in\ell}) \approx \mathsf{Sim}^{Analyst,\mathcal{L}'}(sum, \{m_i\}_{i\in\mathcal{L}'}).$$

$\mathsf{Sim}^{Analyst,\mathcal{L}'}$ simulates the messages sent by the honest parties as follows: First, the simulator generates $\hat{ct}_j = \mathsf{Enc}_{pk}(sum - \sum_{i=1}^{\ell'} m_i)$, which is an encryption of the final sum minus the inputs of the corrupt participants for $i \in \mathcal{L}'$. Then, the simulator publishes a commitment $\hat{c}_j$ to this ciphertext as the commitment of the first honest participant and saves the randomness of this commitment $\hat{r}_j$. Next, the simulator replaces every other honest participant's encrypted message ct_i with a fresh encryption of zero $\hat{ct}_i = \mathsf{Enc}_{pk}(0)$, follows the protocol specification, and generates commitments $\hat{c_i} = \mathsf{Com}(\hat{ct_i}, \hat{r_i})$ to these encryptions and publishes them. Then, it creates ct_{sum} as the product of the products of ciphertexts of the corrupt $(\prod_{i=1}^{\ell'} ct_i)$ and the honest parties $(\prod_{i=j}^{\ell} \hat{ct_i})$. It can be seen that ct_{sum} is an encryption of the correct sum. The simulator computes r_{sum} by following the protocol and adding up the randomness of the corrupt parties $(\sum_{i=1}^{\ell'} r_i)$ with the randomness of the honest parties $(\sum_{i=j}^{\ell} \hat{r_i})$. Finally, the simulator sends $ct_{sum} = \hat{ct_j} \cdot \prod_{i=j+1}^{\ell} \mathsf{Enc}_{pk}(0) \cdot \prod_{i=1}^{\ell'} ct_i = \mathsf{Enc}_{pk}(sum)$ and $r_{sum} = \hat{r_j} + \sum_{i=j+1}^{\ell} \hat{r_i} \cdot \prod_{i=1}^{\ell'} r_i$ to the analyst. The latter checks that:

$$
\begin{aligned}
&\mathsf{Com}(\hat{ct_j}, \hat{r_j}) \cdot \prod_{i=j+1}^{\ell} \mathsf{Com}(\mathsf{Enc}_{pk}(0), \hat{r_i}) \cdot \prod_{i=1}^{\ell'} \mathsf{Com}(ct_i, r_i) \\
&= \mathsf{Com}(\mathsf{Enc}_{pk}(sum - \sum_{i=1}^{\ell'} m_i), \hat{r_j}) \cdot \prod_{i=j+1}^{\ell} \mathsf{Com}(\mathsf{Enc}_{pk}(0), \hat{r_i}) \cdot \prod_{i=1}^{\ell'} \mathsf{Com}(\mathsf{Enc}_{pk}(m_i), r_i) \mod N^2 \\
&= \mathsf{Com}(\mathsf{Enc}_{pk}(sum - \sum_{i=1}^{\ell'} m_i) \cdot \prod_{i=j+1}^{\ell} \mathsf{Enc}_{pk}(0) \cdot \prod_{i=1}^{\ell'} \mathsf{Enc}_{pk}(m_i), \hat{r_j} + \sum_{i=j+1}^{\ell} \hat{r_i} + \sum_{i=1}^{\ell'} r_i) \\
&= \mathsf{Com}(\mathsf{Enc}_{pk}(sum - \sum_{i=1}^{\ell'} m_i + \sum_{i=j+1}^{\ell} 0 + \sum_{i=1}^{\ell'} m_i), \hat{r_1} + \sum_{i=j+1}^{\ell} \hat{r_i} + \sum_{i=1}^{\ell'} r_i) \\
&= \mathsf{Com}(\mathsf{Enc}_{pk}(sum), r_{sum}) = \mathsf{Com}(ct_{sum}, r_{sum}).
\end{aligned}
$$

We now argue that each successive pair of hybrids below is indistinguishable:

– HYB_0: The view of the corrupt parties in a real execution of the protocol. This includes ct_i, c_i, and r_i for $i \in \mathcal{L}'$, c_i for $i \in \mathcal{L}$, as well as ct_{sum} and r_{sum}.
– HYB_1: The same as HYB_0, except, that we replace the message of the first honest participant m_j with $sum - \sum_{i=1}^{\ell'} m_i$ for $i \in \mathcal{L}'$, as well as every other message of honest participants $\{m_i\}_{i \in \{j+1, \ell\}}$ with zero, i.e., $ct_i = \mathsf{Enc}_{pk}(m_i)$ becomes $ct_i = \mathsf{Enc}_{pk}(0)$ and thus $c_i = \mathsf{Com}(\mathsf{Enc}_{pk}(0), r_i)$ for $r_i \xleftarrow{\$} \mathbb{Z}_{N^2}^{\times}$.
– HYB_2: The view of the corrupt parties output by $\mathsf{Sim}^{Analyst, \mathcal{L}'}$.

Observe that HYB_0 and HYB_1, are indistinguishable by the CPA security of the Paillier encryption scheme. Finally, observe that HYB_1 and HYB_2 are identical.

Malicious Corruption of Curator. The curator receives the ciphertexts from the participants, as well as the commitments, and the commitment randomness. Then, the curator computes the encrypted sum and the aggregated randomness. It is straightforward to see that as long as the Paillier encryption is secure, nothing is revealed about each participant's individual data points. We define a PPT simulator such that for all participant inputs $\{m_i\}_{i \in \ell}$,

$$\mathsf{Real}^{Curator}(\{m_i\}_{i \in \ell}) \approx \mathsf{Sim}^{Curator}().$$

Note that the simulator does not need to access individual client messages or even the final result. $\mathsf{Sim}^{Curator}$ simulates the messages sent by the honest parties as follows: The simulator replaces every participant's encrypted message ct_i with a fresh encryption of zero. Next, the simulator follows the protocol specification and generates commitments to these encryptions and sends them to the curator. Finally, after the simulator receives $ct_{sum} = \sum_{i=1}^{\ell} 0$ and $r_{sum} = \sum_{i=1}^{\ell} r_i$ from the curator, it checks that $\mathsf{Com}(ct_{sum}, r_{sum}) = \prod_{i=1}^{\ell} c_i$. If this check passes, the simulator decrypts ct_{sum}. Otherwise (if the check does not pass), the simulator detects malicious activity and discards it. Our homomorphic commitment provides public verifiability without compromising the participants' privacy and ensures any auditor that the curator computed the encrypted sum correctly.

We now argue that each successive pair of hybrids below is indistinguishable:

– HYB_0: The view of the curator in a real execution of the protocol. This includes ct_i, c_i, and r_i for $i \in \{1, \ldots, \ell\}$, as well as ct_{sum} and r_{sum}.
– HYB_1: The same as HYB_0, except, that we replace every $\{m_i\}_{i\in\ell}$ with zero, i.e., $ct_i = \mathsf{Enc}_{pk}(m_i)$ becomes $ct_i = \mathsf{Enc}_{pk}(0)$ and thus $c_i = \mathsf{Com}(\mathsf{Enc}_{pk}(0), r_i)$ for $r_i \xleftarrow{\$} \mathbb{Z}^{\times}_{N^2}$.
– HYB_2: The view of the curator output by $\mathsf{Sim}^{Curator}$.

Observe that HYB_0 and HYB_1, are indistinguishable by the CPA security of the Paillier encryption scheme. Finally, observe that HYB_1 and HYB_2 are identical.

7.2 Security Analysis of zk-Masquerade

Theorem 3. *Assuming that the threshold Paillier cryptosystem is semantically secure against chosen plaintext attacks (IND-CPA) and the ZKPs satisfy the three properties from section 2.4, then the zk-Masquerade protocol (Fig. 3) securely implements a PVPDA protocol against malicious corruption of ℓ' clients with either the curator or the analyst.*

Proof. **Malicious Corruption of Curator with Multiple Participants.** Observe that the honest participants' data are protected against a malicious curator since neither Paillier ciphertexts nor the ZKPs reveal anything about the plaintexts. We also assume that a set $\mathcal{L}' = \{1, \ldots, \ell'\}$ of $\ell' < \ell$ participants are corrupt. We denote the remaining honest participants as $\mathcal{L} = \{j, \ldots, \ell\}$, where $j = \ell' + 1$.

We define a PPT simulator such that for all honest participant inputs $\{m_j\}_{j\in\mathcal{L}}$,

$$\mathsf{Real}^{Curator,\mathcal{L}'}(\{m_i,\}_{i\in\ell}) \approx \mathsf{Sim}^{Curator,\mathcal{L}'}().$$

We can simulate this case by replacing the messages of all honest parties with encryptions of zero and valid ZKPs to these encryptions. Note that [28] fully simulates the threshold Paillier scheme in the two-party setting with security against malicious behavior, thus, for brevity we omit the simulations of the distributed key generation and the distributed decryption. This case is trivial as neither the curator nor any of the corrupted participants $\mathcal{L}'$ receive any output, thus the simulator can send encryptions of zero and valid ZKPs to the curator. $\mathsf{Sim}^{Curator,\mathcal{L}'}$ simulates the messages sent by the honest parties by replacing every honest participant's encrypted message ct_i with a fresh encryption of zero and a valid ZKP to this encryption and sends them to the curator. Each successive pair of hybrids below is indistinguishable:

– HYB_0: The view of the corrupt parties in a real execution of the protocol.
– HYB_1: The same as HYB_0, except, that we replace every $\{m_i\}_{i\in\ell}$ with zero, i.e., $ct_i = \mathsf{Enc}_{pk}(m_i)$ becomes $ct_i = \mathsf{Enc}_{pk}(0)$ and thus $\pi_i = \mathsf{Prove}(ct_i, 0)$. Indistinguishable from the previous by the CPA security of the Paillier encryption scheme.
– HYB_2: The view of the corrupt parties output by $\mathsf{Sim}^{Curator,\mathcal{L}'}$. Identical to the previous hybrid.

Malicious Corruption of Analyst with Multiple Participants. Since the malicious analyst only learns the encrypted sum (and thus the final result), nothing is revealed about each participant's individual data points. We define a PPT simulator such that for all participant inputs $\{m_j\}_{j\in\ell}$,

$$\mathsf{Real}^{Analyst,\mathcal{L}'}(\{m_i\}_{i\in\ell}) \approx \mathsf{Sim}^{Analyst,\mathcal{L}'}(sum),$$

where $sum = \sum_{i=1}^{\ell} m_i$ is the final aggregation result. Note that the simulator needs a way to extract the malicious parties' inputs to invoke the functionality for computing the data aggregation, which can be achieved from the ZKPs (as shown in [35] for the range proof and in [4] for the set-membership proof). Next, the simulator computes $sum - \sum_{i=1}^{\ell'} m_i$ and replaces the honest parties inputs with random additive shares of $sum - \sum_{i=1}^{\ell'} m_i$, s.t., each input is less than the ZKP threshold and encrypts them. The simulator creates valid ZKPs for these encryptions and sends them to the analyst (as if they were posted in the public ledger). The simulator follows computes ct_{sum}, partially decrypts it, creates a proof of correct decryption, and sends all these values to the analyst. Finally, a corrupt analyst that does not decrypt ct_{sum} correctly cannot create a valid proof of decryption. In this case, the auditors will detect a corrupt analyst. We argue indistinguishability as follows:

- HYB_0: The view of the corrupt parties in a real execution of the protocol.
- HYB_1: The same as HYB_0, except, that we replace the messages of the honest participants with random additive shares of $sum - \sum_{i=1}^{\ell'} m_i$ for $i \in \mathcal{L}'$ s.t., each input is less than the ZKP threshold and encrypts them. The simulator creates valid ZKPs for these encryptions and sends them to the analyst. This hybrid is indistinguishable from HYB_0 by the CPA security of the Paillier encryption scheme.
- HYB_2: The view of the corrupt parties output by $\mathsf{Sim}^{Analyst,\mathcal{L}'}$. Identical to HYB_1.

8 Experimental Evaluations

Our evaluations assess the performance of our protocols and examine how they scale with an increasing number of participants. We expect both protocols to scale linearly with the number of clients, with the zero-knowledge protocols (i.e., zk-Masquerade) dominating the execution time, especially as the soundness parameter t increases. Finally, the set-membership proof is more computationally intensive than the range proof, and thus we expect the quantitative studies to be faster than the categorical ones.

8.1 Experimental Setup

For our experimental evaluation of the curator and the analyst, we used two t3.2xlarge AWS EC2 instances running with eight virtual processors up to 2.5 GHz and 32 GB RAM. For the experiments evaluating the participants, we used a Lenovo laptop with an i7-8650U CPU running at 1.90 GHz, and we implemented our Masquerade framework in Go 1.16. In our experiments, the latency between the curator and the analyst was negligible (about 0.014 ms) as both are hosted in AWS instances. The communication between the participants and the AWS instances (curator and analyst) is 9.65 ms on average. The back-end of the curator is fully parallelizable and can take advantage of all the available threads of the host. In zk-Masquerade, for every participant, the curator spawns a new thread to verify the zero-knowledge proof of the participant and aggregates their private data only if the ZKP verification was successful. This optimization provides a speed-up proportional (roughly) to the number of parallel cores of the curator. In Masquerade, we still use the same parallelization technique but as the aggregation is significantly faster than the ZKP verification, this does not offer significant speedup. Without loss of generality, in our implementation, the analyst maintains a public append-only ledger. After our protocols have finished execution, we offer a website for inspecting the ledger. Note that the ledger can also be maintained by any trusted third party, or be distributed between the analyst and the curator. Finally, both the Paillier cryptosystem and our commitment scheme are instantiated with a 4096-bit modulus N^2 following the key size recommendations of [3].

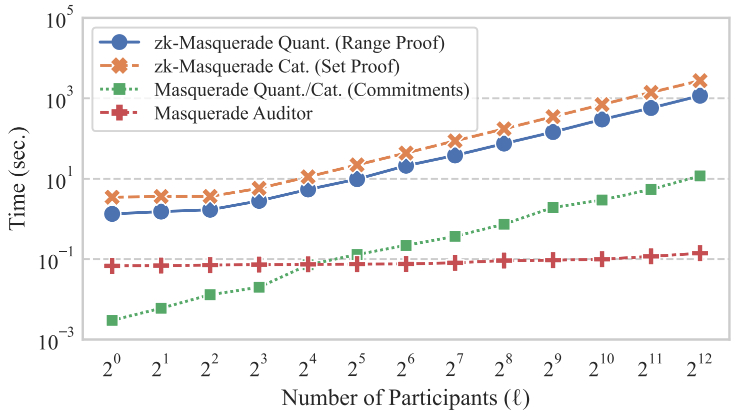


Fig. 5. Total protocol timings in seconds for an increasing number of participants with $t = 60$ and $K = 4$. Here we report the online protocol time, i.e., from the moment the participants send their data until the final result is aggregated.

8.2 Performance Evaluation

In Fig. 5 we present the *online protocol costs* of Masquerade and zk-Masquerade assuming both honest and malicious participants. In the former case, the participants' ciphertexts are well-formed and the curator accepts all the encrypted data directly. We use the timings of Masquerade as a baseline to demonstrate the trade-off of the zero-knowledge protocols for the quantitative and categorical studies (note, the honest participant cost does not depend on the type of study). Fig. 5 demonstrates the experimental timings for quantitative variables of zk-Masquerade (i.e., blue circles), categorical variables (i.e., orange "X"s trend), Masquerade for both quantitative and categorical values (i.e., green squares), and the timing for an auditor (i.e., red "+"s trend) with an increasing number of participants. For the ZKPs, we used soundness parameter $t = 60$ bits.

The experimental timings include the computation performed by the analyst and the curator until the result of the aggregation is published by the analyst. We note that Fig. 5 does not include the offline cost for key generation or participant cost (these are discussed later in this Section) since all the participants can precompute their ciphertexts, proofs, and commitments before engaging in the protocol. Finally, the timings also include latency, which has a negligible performance cost. For the reported cost of the categorical study in Fig. 5 we used a set with $S = 4$ classes. As expected, for both types of studies we report a linear increase in the total aggregation cost as the number of participants grows. Additionally, for up to 8 participants the cost of the quantitative and categorical *remains almost constant* for zk-Masquerade, due to the thread parallelization in the curator's back-end. When we compare both trends that use ZKPs with the trend of Masquerade (which only uses commitments), we observe that protecting against malicious participants incurs about two orders of magnitude additional overhead. Likewise, the quantitative study is about half an order of magnitude faster than the categorical one, which is attributed to the complexity of the set-membership proof [4]. Finally, we observe that the time to audit the ledger for Masquerade is almost constant with an increasing number of participants as it mostly comprises modular multiplications. Next, we analyze the individual performance of the participants, the curator, the analyst, and the auditor and discuss how each one affects the total performance of our protocol.

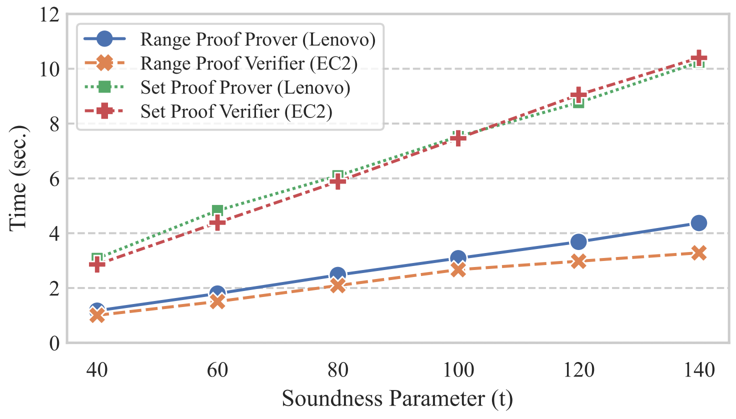


Fig. 6. Time measurements in seconds for both range and set-membership proofs for the prover and the verifier with an increasing soundness parameter t in bits and $K = 4$.

Participant Perspective. The computation costs for each participant include the Paillier encryption cost, the ZKP cost (only in zk-Masquerade), and the commitment cost (only in Masquerade). Notably, our protocol allows the participants to pre-compute everything offline. Both the Paillier encryption cost and the cost of committing to encrypted data are negligible compared to the ZKP generation cost. Fig. 6 shows the proof generation time for participants for both *range* and *set-membership* proofs for varying the soundness parameter t. Similarly, Table 2 reports the ZKP size in KBytes with an increasing t for range and set-membership proofs.

Both the computation and space costs of the range proofs scale linearly to the soundness parameter t. Similarly, the computation and space costs of the set-membership proofs scale linearly to t, but are also affected by the number k of messages in the set. In Table 3 we show the timings for $\mathcal{P}$ and $\mathcal{V}$ for the set-membership proof with increasing set sizes. Our results show that the performance cost of both $\mathcal{P}$ and $\mathcal{V}$ increases linearly to the set size.

Table 2. Size of the Zero-knowledge Proof Protocols for zk-Masquerade.

Soundness Parameter (t)	**40**	**60**	**80**	**100**	**120**	**140**
Range Proof (KB)	36.0	57.4	78.1	103.1	118.4	142.2
Set Proof (KB)	309.6	464.5	619.3	774.1	929.0	1083.9

Smartphone participants. This is the first work to further evaluate the participant costs using a smartphone edge device; in particular, we use a smartphone equipped with a Qualcomm Snapdragon 865 at 2.84 GHz. In our analysis we use soundness $t = 40$, and observer that the participant cost is dominated by $\mathcal{P}$. Overall, the cost for a range proof is 2.14 seconds, while a set-membership proof with $S = 4$ takes 4.78 seconds (i.e., about 0.014% of the phone's effective battery life). In all cases, the commitment cost was negligible at just 78 ms. This analysis highlights that using a modern smartphone instead of a laptop (c.f. Fig. 6 prover costs for $t = 60$) incurs very similar overheads.

Table 3. Set-membership proof timings with an increasing number of set elements for soundness $t = 60$ for zk-Masquerade.

Set Size (S)	**2**	**4**	**8**	**16**	**32**	**64**
Prover Time (Lenovo) (sec.)	1.92	2.94	7.92	14.61	30.55	63.83
Verifier Time (EC2) (sec.)	2.08	2.61	6.97	13.26	31.93	65.14

Curator Perspective. Assuming ℓ participants, the total curator overhead corresponds to: ℓ homomorphic additions (Eq. 1) and, in zk-Masquerade, the verification cost for ℓ proofs. The curator in Masquerade is significantly faster than zk-Masquerade; below we focus on the latter and analyze the cost of having ZKPs. As shown in Fig. 6 for the proof generation, the verification cost of both proof types scales linearly to the soundness parameter. As shown in Table 3, set-membership verification incurs a linear overhead to the size of the set of valid messages. Moreover, the verification cost for set-membership is about half an order of magnitude higher compared to range ZKP due to the increased complexity of [4]. This difference justifies why quantitative studies are faster than categorical (as reported in Fig. 5).

In either study type, the curator verifies one proof for each of the ℓ participants and performs one homomorphic addition if the proof is verified successfully. The overall communication cost of the curator is ℓ times the proof size (Table 2), as well as the cost of transmitting the final results to the analyst (i.e., ct_{sum} and r_{sum}). Likewise, without any optimizations, the curator's performance overhead is ℓ times the cost for one participant; using 8 parallel threads, the curator's performance is about 5.3 times faster. We remark that the total performance of the protocol is dominated by the curator's overhead.

Analyst Perspective. In our implementation, the analyst maintains the public ledger. In Masquerade, the timing cost of the analyst can be attributed to: (1) the key generation, (2) the ℓ multiplications and additions of the commitments and the random parameters, respectively, and (3) the constant costs of opening the commitment and decrypting the aggregation result. The key generation offline cost involves generating two large prime numbers and a group element of maximum order. For 2048, 4096, and 8192-bit plaintexts (i.e., N bitsize), key generation takes 43.1 milliseconds, 202 milliseconds, and 1.27 seconds, respectively. In Masquerade, in addition to decrypting the encrypted sum, the only additional cost of the analyst is to support public verifiability by aggregating all participant commitments and showing that it correctly opens with the homomorphic sum (Alg. 4 in Fig. 2).

Table 4. A comparison with existing PDA schemes based on the cryptographic technique they utilize, the type of variables they support and their robustness against dropping participants and malicious inputs.

Approach	**Cryptographic Technique**	**Quant. Vars**	**Categ. Vars**	**Robust (Dropouts)**	**Public Verifiability**	**Robust (Malicious Inputs)**	**Proof Scaling Quant./Categ.**
Shi *et al.* [51]	THE[*] & Diff. Privacy	●	○	○	○	○	N/A
Chan *et al.* [14]	THE & Diff. Privacy	●	○	●	○	○	N/A
Joye *et al.* [30]	THE	●	○	○	○	○	N/A
Danezis *et al.* [18]	THE	●	○	○	○	○	N/A
Leontiadis *et al.* [32]	THE	●	○	●	○	○	N/A
PUDA [33]	THE	●	○	○	●	○	N/A
Bonawitz *et al.* [9]	Pairwise Masking	●	○	●	○	○	N/A
SEPIA [11]	S.Sharing, Generic MPC	●	○	●	○	●	Quadratic / N/A
Giannopoulos *et al.* [26]	S.Sharing, Generic MPC	●	●	●	○	○	N/A
zkLedger[†] [42]	Pedersen Commitments	●	○	●	●	●	Quadratic / N/A
Prio[‡] [15]	S.Sharing	●	●	●	○	●	Linear / Linear
Masquerade	PHE & Commitments	●	●	●	●	○	N/A
zk-Masquerade	THE & ZKP	●	●	●	●	●	Constant / Linear

[†] zkLedger is the only related work that achieves both public verifiability and being robust against malicious inputs, however, the transaction verification is quadratic to the number of participants causing major scalability issues (the authors only experiment with at most 20 participants). Additionally, zkLedger does not support categorical variables.

[‡] Prio introduces a mechanism called affine-aggregatable encodings ("AFEs") to encode a series of bits as quantitative or categorical variables. Prio does not guarantee public verifiability and each client only communicates with the servers (no support for a distributed ledger).

[*] Threshold Homomorphic Encryption (THE): The private key is shared between n participating parties and in order to decrypt a message at least t-out-of-n parties are required.

In zk-Masquerade, the timing cost of the analyst is very similar to the one of the curator as the analyst audits the curator. The communication cost of the analyst in Masquerade entails receiving the individual commitments from each participant, while in zk-Masquerade the analyst audits the proofs from the ledger and also computes its local encrypted sum to audit the curator.

Auditor Perspective: Finally, any auditor can access a copy of the ledger in order to verify that the protocol was executed correctly, which allows public verifiability. The auditor in Masquerade uses the accumulated randomness and verifies that the participants' homomorphic commitments open with the encrypted sum. This verification is similar to the one that the analyst performs in Fig. 2. Lastly, as mentioned in section 5.1, the auditor verifies that the published decrypted sum is indeed a correct decryption of the ciphertext that the curator published to the ledger. We measured the auditor timings in Fig. 5 (i.e., yellow circles) with a modern Lenovo laptop (the same machine was also used for the participants) and we observe almost constant overhead. Auditing zk-Masquerade is more expensive as the auditor has to essentially repeat the computation of the analyst.

9 Related Work

In this section, we discuss several recent works in PDA that rely on cryptographic techniques such as HE, secure multiparty computation (MPC), and differential privacy. Existing solutions introduce different trade-offs between computation and communication costs, depending on the underlying privacy technique they employ. Interestingly, some protocols provide robustness against dropouts, while only a few are secure against malicious clients that provide malformed inputs. Our protocols and only two other works allow the participants to verify the outcome of the protocol. Next, we discuss these existing works and how they compare with Masquerade and zk-Masquerade.

In MPC, multiple parties want to jointly compute a function over their private data while keeping those inputs private [53]. Most common constructions are either based on garbled circuits or on HE and secret sharing [11,2,36,26,31]. The latter is more computationally efficient than the former but incurs high communication costs. Such generic MPC protocols can be used for PVPDA but since they are not optimized solely for this task, they are not as efficient as customized PVPDA solutions such as our protocols. Likewise, generic schemes allow the participants to provide *any input* to the aggregation, whereas Masquerade protects against malicious inputs using the two non-interactive ZKPs discussed in section 5.2. Both MPC and the works in [18,33] that use thresholding-based cryptography (such as Shamir's secret-sharing [49]) remain susceptible to participant dropouts as the participating parties need to be online to partially decrypt the result. Conversely, Masquerade and zk-Masquerade is immune by design to participants intentionally leaving the computation and offers public verifiability, where third parties can audit the result.

Shi *et al.* [51] proposed a PDA methodology based on random shares and distributed noise generation. The authors of [14] and [30] extended Shi's protocol by handling client dropouts and enabling larger plaintext spaces, respectively. Likewise, the work in [18] is also based on secret-sharing techniques for privacy-preserving smart meter readings, yet it cannot tolerate participant dropouts. PUDA [33] is a pairing-based scheme that uses a *trusted dealer* to set up secret keys and enables public verification using homomorphic tags. While these aforementioned protocols support only quantitative variables our protocols enable both categorical and quantitative studies. Additionally, in these works, any malicious client can corrupt the results, while zk-Masquerade offers robust protection against malicious clients. The work in [9] features an efficient t-out-of-n secret sharing protocol, however, like most related works, it cannot tolerate malformed inputs from malicious participants. Trinocchio [48] outsources a computation *from a single client* to multiple workers in a privacy-preserving way and generates verifiable guarantees of the correct computation. Trinocchio additionally describes a multi-client version but this variant requires a special key generation and also is susceptible to malicious inputs: if any client provides incorrect information, the computation is aborted. Moreover, among the aforementioned works, only PUDA and our protocols offer public verifiability.

Prio [15] introduces a technique called secret-shared non-interactive proofs (SNIPs) to validate that clients' inputs are the encryption of either a "0" or a "1". It is based on multiple non-colluding servers and is the most closely related to zk-Masquerade. To aggregate longer values than just one bit, Prio encodes the participants' private data as a sequence of bits, where each bit is verified by a different SNIP, and computes the sum of these encodings. Therefore, since SNIPs inherently prove one bit, to prove messages of bigger sizes the proof size increases linearly to the number of bits. Notably, the range proofs in zk-Masquerade incur *constant size and time*, whereas our set-membership proofs *scale linearly* to the set elements. For very small client inputs, Prio's design enables fast participant timings: For instance, the authors report that for 1-bit up to 4-bit integers, the participant timing varies between 0.01 to 1 second, depending on the number of the multiplication gates on the SNIP circuit. We emphasize that zk-Masquerade offers superior scalability, given that our plaintext dynamic range spans thousands of bits (e.g., 2048 bits in our experiments), allowing for the encoding of substantially more information. While the authors of Prio do not report any timings for categorical studies (which can be theoretically supported with modified proofs), their approach introduces additional overheads, considering that each participant must prove that the sum of all the bits is exactly one. Finally, Prio does not operate on a public ledger and does not offer any public verifiability protections, contrary to our approach.

In zkLedger [42], the authors propose a method that protects participants' privacy and allows public verifiability. In this case, the participants rely on Pedersen commitments [45] and Schnorr-type ZKPs [47] to publish their transactions on a public ledger, before an auditor can combine all the public commitments and verify their correctness. Likewise, in both Masquerade and zk-Masquerade we allow the analyst to audit the commitments and the proofs respectively. Notably, since both the commitments and the proofs in our protocols are public, *any participant or third party* can also audit the committed result. In terms of performance, zkLedger incurs significant overheads since the transaction verification costs are *quadratic to the number of participants*, so practicality is significantly impacted in studies with more than 10-20 participants. Conversely, our experiments report that our approach scales gracefully to thousands of participants.

Table 4 summarizes all notable existing works and compares them with ours. Interestingly, these approaches use a broad range of cryptographic techniques, yet all of them have homomorphic properties. We observe that most of these constructions are tailored to quantitative studies, while only our protocol and Prio [15] provide encodings to represent categorical variables (e.g., Likert scales). Moreover, only PUDA, zkLedger, and Masquerade allow practical auditing and public verifiability, which is an important benefit for real-world applications. Although several prior works have focused on malicious participants intentionally dropping out of the protocol, only zk-Masquerade, SEPIA, Prio, and zkLedger protect against malicious inputs using non-interactive ZKPs; notably, only zk-Masquerade offers *constant ZKP costs for quantitative studies*.

The security and threat models of the aforementioned existing works vary based on the number of computing parties, the types of studies they support, and their robustness against malicious participants. The protocols in [51,14,30] use a single untrusted aggregator, and assume the *aggregator obliviousness* model which ensures that the aggregator learns only the sum of users' inputs and nothing else. The authors of [33] extended the aggregator obliviousness model and introduce the concept of *aggregate unforgeability*, which assumes that one party performs the aggregation and another party decrypts and publishes the final result. Aggregate unforgeability ensures that the aggregator cannot convince the data analyzer to accept a bogus sum. Similarly to our work, in this model, the two parties should never collude. In [9], the authors introduce three different security models, one which ensures security only against the clients, one which ensures security only against the server, and a mixed security model for a threshold of colluding clients with the server. The work in [11] uses the honest-but-curious model and is secure as long as no more than half of the computing nodes collude, while the authors of [26] use the Sharemind MPC framework with three honest-but-curious non-colluding nodes. Lastly, the protocol in [42] is secure against malicious input providers (e.g., banks) and keeps the transactions private as long as the sender, the receiver, and the auditor do not collude, whereas the authors of [15] use a small set of servers (e.g., three) and protect the client privacy if not all the servers collude. Like Masquerade and zk-Masquerade, most related works rely on two or more non-colluding servers. However, as shown in Table 4, only zk-Masquerade offers public verifiability while protecting against malicious inputs and scales with an increasing number of participants.

10 Concluding Remarks

We present Masquerade and zk-Masquerade, two new schemes for publicly verifiable private data aggregation over the Internet that ensures that participants' inputs are kept private and the analyst only learns the final aggregate result. We have expanded our protocol to work both with quantitative and categorical variables, enabling a variety of different studies. Masquerade publishes our novel *multiplicative homomorphic* commitments on a ledger to enable public verifiability, which is an essential property for real-world applications where participants need to assess the correctness of the encrypted sum and verify that their response is included in the final result. In zk-Masquerade, we defend against malicious participants by adopting a range proof and a set-membership proof and verifying that the encrypted data are well-formed and that each participant has limited influence on the protocol. Moreover, the public verifiability property of our protocols allows the analyst and any auditor (e.g., a participant or an external party) to verify that the protocol result was computed faithfully and that the announced decryption of the encrypted aggregate is correct. Our experiments demonstrate that Masquerade and zk-Masquerade can scale gracefully to thousands of participants, and can further support smartphone clients.

References

1. Carlisle Adams and Steve Lloyd. *Understanding PKI: concepts, standards, and deployment considerations*. Addison-Wesley Professional, Boston, USA, 2003.
2. Abdelrahaman Aly, Marcel Keller, Dragos Rotaru, Peter Scholl, Nigel Smart, and Tim Wood. SCALE–MAMBA Documentation, 2021.
3. Elaine B. Barker, William C. Barker, William E. Burr, W. Timothy Polk, and Miles E. Smid. *Recommendation for Key Management: Part 1 Rev. 5*. National Institute of Standards and Technology, Technology Administration, Gaithersburg, MD, USA, 2007.
4. Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In Ajay D. Kshemkalyani and Nir Shavit, editors, *20th ACM Symposium Annual on Principles of Distributed Computing*, pages 274–283, Newport, Rhode Island, USA, August 26–29, 2001. Association for Computing Machinery.
5. Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14: 9th International Conference on Security in Communication Networks*, volume 8642 of *Lecture Notes in Computer Science*, pages 175–196, Amalfi, Italy, September 3–5, 2014. Springer, Heidelberg, Germany.
6. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.
7. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 701–732, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.
8. Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. Prochlo: Strong Privacy for Analytics in the Crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 441–459, New York, NY, USA, 2017. Association for Computing Machinery.
9. Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1175–1191, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.
10. Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany.
11. Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas A. Dimitropoulos. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In *USENIX Security 2010: 19th USENIX Security Symposium*, pages 223–240, Washington, DC, USA, August 11–13, 2010. USENIX Association.
12. Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, January 2000.
13. Ann Cavoukian, Jules Polonetsky, and Christopher Wolf. Smart privacy for the smart grid: embedding privacy into the design of electricity conservation. *Identity in the Information Society*, 3(2):275–294, 2010.
14. T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Privacy-preserving stream aggregation with fault tolerance. In Angelos D. Keromytis, editor, *FC 2012: 16th International Conference on Financial Cryptography and Data Security*, volume 7397 of *Lecture Notes in Computer Science*, pages 200–214, Kralendijk, Bonaire, February 27 – March 2, 2012. Springer, Heidelberg, Germany.
15. Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, page 259–282, USA, 2017. USENIX Association.
16. Scott E. Coull, Charles V. Wright, Fabian Monrose, Michael P. Collins, and Michael K. Reiter. Playing devil's advocate: Inferring sensitive information from anonymized network traces. In *ISOC Network and Distributed System Security Symposium – NDSS 2007*, pages 1–13, San Diego, CA, USA, February 28 – March 2, 2007. The Internet Society.

17. Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136, Cheju Island, South Korea, February 13–15, 2001. Springer, Heidelberg, Germany.
18. George Danezis, Cédric Fournet, Markulf Kohlweiss, and Santiago Zanella-Béguelin. Smart meter aggregation via secret-sharing. In *ACM Workshop on Smart Energy Grid Security*, page 75–80, New York, NY, USA, 2013. ACM.
19. John R Douceur. The Sybil Attack. In *International workshop on peer-to-peer systems*, pages 251–260, Berlin, Heidelberg, 2002. Springer, Springer Berlin Heidelberg.
20. Cynthia Dwork. Differential Privacy: A Survey of Results. In Manindra Agrawal, Dingzhu Du, Zhenhua Duan, and Angsheng Li, editors, *Theory and Applications of Models of Computation*, pages 1–19, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
21. Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. RAPPOR: Randomized aggregatable privacy-preserving ordinal response. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014: 21st Conference on Computer and Communications Security*, pages 1054–1067, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press.
22. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.
23. Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. In Yair Frankel, editor, *FC 2000: 4th International Conference on Financial Cryptography*, volume 1962 of *Lecture Notes in Computer Science*, pages 90–104, Anguilla, British West Indies, February 20–24, 2001. Springer, Heidelberg, Germany.
24. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.
25. Craig Gentry and Dan Boneh. *A fully homomorphic encryption scheme*. Stanford University, California, USA, 2009.
26. Thanos Giannopoulos and Dimitris Mouris. Privacy preserving medical data analytics using secure multi party computation. an end-to-end use case. Master's thesis, National and Kapodistrian University of Athens, 2018.
27. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.
28. Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, Tomas Toft, and Angelo Agatino Nicolosi. Efficient RSA key generation and threshold paillier in the two-party setting. *Journal of Cryptology*, 32(2):265–323, April 2019.
29. Jim Isaak and Mina J Hanna. User data privacy: Facebook, Cambridge Analytica, and privacy protection. *Computer*, 51(8):56–59, 2018.
30. Marc Joye and Benoît Libert. A scalable scheme for privacy-preserving aggregation of time-series data. In Ahmad-Reza Sadeghi, editor, *FC 2013: 17th International Conference on Financial Cryptography and Data Security*, volume 7859 of *Lecture Notes in Computer Science*, pages 111–125, Okinawa, Japan, April 1–5, 2013. Springer, Heidelberg, Germany.
31. Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1575–1590, Virtual Event, USA, November 9–13, 2020. ACM Press.
32. Iraklis Leontiadis, Kaoutar Elkhiyaoui, and Refik Molva. Private and dynamic time-series data aggregation with trust relaxation. In Dimitris Gritzalis, Aggelos Kiayias, and Ioannis G. Askoxylakis, editors, *CANS 14: 13th International Conference on Cryptology and Network Security*, volume 8813 of *Lecture Notes in Computer Science*, pages 305–320, Heraklion, Crete, Greece, October 22–24, 2014. Springer, Heidelberg, Germany.
33. Iraklis Leontiadis, Kaoutar Elkhiyaoui, Melek Önen, and Refik Molva. PUDA - privacy and unforgeability for data aggregation. In Michael Reiter and David Naccache, editors, *CANS 15: 14th International Conference on Cryptology and Network Security*, Lecture Notes in Computer Science, pages 3–18, Marrakesh, Morocco, December 10–12, 2015. Springer, Heidelberg, Germany.
34. Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1(1):1–53, 1932.
35. Yehuda Lindell. Fast secure two-party ECDSA signing. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 613–644, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

36. Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. ObliVM: A programming framework for secure computation. In *2015 IEEE Symposium on Security and Privacy*, pages 359–376, San Jose, CA, USA, May 17–21, 2015. IEEE Computer Society Press.
37. Oleg Mazonka, Nektarios Georgios Tsoutsos, and Michail Maniatakos. Cryptoleq: A heterogeneous abstract machine for encrypted and unencrypted computation. *IEEE Transactions on Information Forensics and Security*, 11(9):2123–2138, 2016.
38. Dimitris Mouris, Charles Gouert, and Nektarios Georgios Tsoutsos. Privacy-preserving ip verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(7):2010–2023, 2022.
39. Dimitris Mouris and Nektarios Georgios Tsoutsos. Pythia: Intellectual property verification in zero-knowledge. In *57th ACM/IEEE Design Automation Conference, DAC 2020, July 20-24, 2020*, pages 1–6, San Francisco, CA, USA, 2020. IEEE.
40. Dimitris Mouris and Nektarios Georgios Tsoutsos. Zilch: A Framework for Deploying Transparent Zero-Knowledge Proofs. *IEEE Transactions on Information Forensics and Security*, 16:3269–3284, 2021.
41. Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy*, pages 111–125, Oakland, CA, USA, May 18–21, 2008. IEEE Computer Society Press.
42. Neha Narula, Willy Vasquez, and Madars Virza. Zkledger: Privacy-preserving auditing for distributed ledgers. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation*, NSDI'18, page 65–80, USA, 2018. USENIX Association.
43. Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, page 305–320, USA, 2014. USENIX Association.
44. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany.
45. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany.
46. Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.
47. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany.
48. Berry Schoenmakers, Meilof Veeningen, and Niels de Vreede. Trinocchio: Privacy-preserving outsourcing by distributed verifiable computation. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 16: 14th International Conference on Applied Cryptography and Network Security*, volume 9696 of *Lecture Notes in Computer Science*, pages 346–366, Guildford, UK, June 19–22, 2016. Springer, Heidelberg, Germany.
49. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.
50. Adi Shamir, Ronald L. Rivest, and Leonard M. Adleman. *Mental Poker*, pages 37–43. Springer US, Boston, MA, 1981.
51. Elaine Shi, T.-H. Hubert Chan, Eleanor G. Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *ISOC Network and Distributed System Security Symposium – NDSS 2011*, pages 1–17, San Diego, CA, USA, February 6–9, 2011. The Internet Society.
52. Nektarios Georgios Tsoutsos and Michail Maniatakos. The HEROIC framework: Encrypted computation without shared keys. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(6):875–888, 2015.
53. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.

A Zero-Knowledge Proof Protocols

A.1 Zero-Knowledge Enc(0) Proof

Protocol 1 describes a non-interactive ZKP to prove that a given ciphertext is an encryption of zero. We adopt the protocol for proving ciphertext equality over N^{s+1} modulus with $s = 1$ from [17] and the Fiat-

Protocol 1 Zero-Knowledge Encryption-of-Zero Proof

Public Inputs: Let c be a Paillier ciphertext known to both $\mathcal{P}$ and $\mathcal{V}$. The soundness parameter t is the bit size of the verifier challenge h.
Private Inputs: $\mathcal{P}$ knows the randomness ρ such that $c = \rho^N \mod N^2$.
Goal: The prover $\mathcal{P}$ convinces the verifier $\mathcal{V}$ that c is the encryption of zero.

The protocol:

1. **Proof:**
   (a) $\mathcal{P}$ generates a random $\rho' \in \mathbb{Z}_N^*$ and computes $a \leftarrow {\rho'}^N \mod N^2$ and sends it to $\mathcal{V}$.
   (b) $\mathcal{P}$ computes $h \leftarrow \texttt{SHA256}(N, c, a)$
   (c) $\mathcal{P}$ computes $z \leftarrow \rho' * \rho^h \mod N^2$ and sends it to $\mathcal{V}$.
2. **Verification:**
   (a) $\mathcal{V}$ generates the randomness based on the transcript as $h \leftarrow \texttt{SHA256}(N, c, a)$.
   (b) $\mathcal{V}$ verifies that $z^N = a * c^h \pmod{N^2}$.

Shamir heuristic in the random oracle model to convert the ZKP to a non-interactive variant. Let c be a Paillier encryption of zero known both to $\mathcal{P}$ and $\mathcal{V}$ and ρ be the secret randomness used for c (i.e., $c = \rho^N \mod N^2$). As described by the protocol, $\mathcal{P}$ generates and sends a and z to $\mathcal{V}$, who in turn can easily check that $\mathsf{Enc}_{pk}(0, z) = z^N \mod N^2 = a * c^h \mod N^2$.

Protocol 2 Zero-Knowledge Range Proof

Public Inputs: Let $\mathcal{R} = \{0, \ldots, \ell\}$ be a range of $\ell + 1$ messages and $c = g^m \rho^N \mod N^2$ be the Paillier encryption of a secret message $m \in \{0, \ldots, \ell\}$. Let t be the soundness parameter.
Private Inputs: $\mathcal{P}$ knows secret message $m \in \mathcal{R}$.
Goal: The prover $\mathcal{P}$ convinces the verifier $\mathcal{V}$ that c is the encryption of a message in $\mathcal{R}$.

The protocol:

1. **Setup:**
   (a) For $i \in \{1, \ldots, t\}$, $\mathcal{P}$ chooses random $w_1^1, \ldots, w_1^t \leftarrow \{\ell, \ldots, 2\ell\}$ and computes $w_2^i = w_1^i - \ell$.
   (b) $\mathcal{P}$ switches the values of w_1^i and w_2^i with probability 1/2 independently for each $i \in \{1, \ldots, t\}$.
   (c) For $i \in \{1, \ldots, t\}$, $\mathcal{P}$ computes $c_1^i = \mathsf{Enc}_{pk}(w_1^i, \rho_1^i)$ and $c_2^i = \mathsf{Enc}_{pk}(w_2^i, \rho_2^i)$, where $\rho_1^i, \rho_2^i \leftarrow \mathbb{Z}_N$ are the randomness used in the Paillier encryption.
2. **Commit:** $\mathcal{P}$ generates $h \leftarrow \{0,1\}^t$ in the random oracle model as $h \leftarrow H(c_1^1, c_2^1, \ldots c_1^t, c_2^t)$.
3. **Proof:** For each $i \in \{1, \ldots, t\}$:
   (a) Depending on the value of h_i, $\mathcal{P}$ sets z_i as follows:
   $$z_i = \begin{cases} (w_1^i, \rho_1^i, w_2^i, \rho_2^i), & h_i = 0 \\ (j, m + w_j^i, \rho \cdot \rho_j^i \mod N), & \\ j \in \{1,2\} \mid m + w_j^i \in \{\ell, \ldots, 2\ell\}, & h_i = 1. \end{cases}$$
   (b) Finally, $\mathcal{P}$ sends $c_1^1, c_2^1, \ldots c_1^t, c_2^t$ and $z_1, \ldots, z_t$ to $\mathcal{V}$.
4. **Verification:** $\mathcal{V}$ generates h as in step 2) and then parses z_i according to the value of h_i. For every $i \in \{1, \ldots, t\}$:
   (a) If $h_i = 0$, $\mathcal{V}$ checks that $c_1^i = \mathsf{Enc}_{pk}(w_1^i, \rho_1^i)$ and $c_2^i = \mathsf{Enc}_{pk}(w_2^i, \rho_2^i)$ and that one of $\hat{w}_1^i, \hat{w}_2^i \in \{\ell, \ldots, 2\ell\}$ while the other is in $\{0, \ldots, \ell\}$, where $z_i = (w_1^i, \rho_1^i, w_2^i, \rho_2^i)$.
   (b) If $h_i = 1$, $\mathcal{V}$ checks that $c \oplus c_j^i = \mathsf{Enc}_{pk}(w_i, \rho_i)$ and $w_i \in \{\ell, \ldots, 2\ell\}$, where $z_i = (j, w_i, \rho_i)$.
   (c) $\mathcal{V}$ outputs 1 if and only if all of the above checks pass.

A.2 Zero-Knowledge Range Proof

Protocol 2 describes a non-interactive ZKP that a Paillier ciphertext c encrypts a message m from within a range of plaintexts $\mathcal{R} = \{0, \ldots, \ell\}$. We adopt the range proof from [35] and make it non-interactive using the Fiat-Shamir heuristic in the random oracle model to eliminate the interaction between the prover and the verifier. More specifically, in step (2), $\mathcal{P}$ generates the randomness h based on the transcript of the protocol until that point (i.e., $c_1^1, c_2^1, \ldots c_1^t, c_2^t$). In step (4), $\mathcal{V}$ generates h herself based on the public transcript and continues with the verification of the proof. The soundness parameter t corresponds to the number of queries that the prover has to answer in order to convince the verifier, i.e., the probability that it answers all is at most 2^{-t}.

A.3 Zero-Knowledge Set Membership Proof

Finally, in Protocol 3, we describe a non-interactive variant of a ZKP that asserts that a Paillier ciphertext c encrypts a message m from a public set of plaintexts $\mathcal{S} = \{m_1, \ldots, m_k\}$ [4]. Our proof uses the Fiat-Shamir heuristic in the random oracle model to eliminate the interaction between the prover and the verifier. In step (2), the prover generates the (unpredictable) challenge h as `SHA256`$(u_1, \ldots u_k)$ so that anyone can re-generate h from the u vector. If the protocol is iterated t times and $m' \notin \mathcal{S}$, then the probability that a malicious prover convinces an honest verifier that $m' \in \mathcal{S}$ is $1/A^t$, where A is the size of the challenge.

Protocol 3 Zero-Knowledge Set Membership Proof

Public Inputs: Let $\mathcal{S} = \{m_1, \ldots, m_k\}$ be a public set of k messages and $c = g^{m_i} \rho^N \mod N^2$ be the Paillier encryption of m_i, where i is secret. Let t be the soundness parameter and A the number of bits of the hash function H.
Private Inputs: $\mathcal{P}$ knows secret message $m \in \mathcal{S}$.
Goal: The prover $\mathcal{P}$ convinces the verifier $\mathcal{V}$ that c is the encryption of a message in $\mathcal{S}$.

The protocol (iterated t times):

1. **Setup:**
   (a) $\mathcal{P}$ chooses a random $\rho' \in \mathbb{Z}_N^*$ and randomly generates vectors h and v with $k-1$ values each, such that $\{h_j\}_{j \neq i} \in \mathbb{Z}_N$ and $\{v_j\}_{j \neq i} \in \mathbb{Z}_N^*$.
   (b) $\mathcal{P}$ computes vector u of size k as:
$$u_j = \begin{cases} {\rho'}^N \mod N^2, & j = i \\ v_j^N (g^{m_j}/c)^{h_j} \mod N^2, & j \neq i. \end{cases}$$
2. **Commit:** $\mathcal{P}$ generates $h \leftarrow \{0,1\}^A$ in the random oracle model as $h \leftarrow H(u_1, \ldots u_k)$.
3. **Proof:** $\mathcal{P}$ computes $h_i = h - \sum_{j \neq i} h_j \mod N$, $v_i = \rho' \rho^{h_i} g^{(h - \sum_{j \neq i} h_j) \div N} \mod N$ and sends $v_1, \ldots, v_k$, $u_1, \ldots, u_k$, and $h_1, \ldots, h_k$ to $\mathcal{V}$.
4. **Verification:**
   (a) $\mathcal{V}$ generates the randomness based on the transcript as $h \leftarrow H(u_1, \ldots u_k)$ and verifies that $h = \sum_j h_j \mod N$.
   (b) Finally, $\mathcal{V}$ verifies that $v_j^N = u_j (c/g^{m_j})^{h_j} \mod N^2$ for each $j \in \{1, \ldots, k\}$.
