



Delft University of Technology

Privacy-Preserving Data Aggregation with Public Verifiability Against Internal Adversaries

Palazzo, Marco; Dekker, Florine W.; Brighente, Alessandro; Conti, Mauro; Erkin, Zekeriya

Publication date

2024

Document Version

Final published version

Published in

Proceedings of the 33rd USENIX Security Symposium

Citation (APA)

Palazzo, M., Dekker, F. W., Brighente, A., Conti, M., & Erkin, Z. (2024). Privacy-Preserving Data Aggregation with Public Verifiability Against Internal Adversaries. In *Proceedings of the 33rd USENIX Security Symposium* (pp. 6957-6974). (Proceedings of the 33rd USENIX Security Symposium). USENIX Association.

Important note

To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright

Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy

Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.



Privacy-Preserving Data Aggregation with Public Verifiability Against Internal Adversaries

Marco Palazzo and Florine W. Dekker, *Cyber Security Group, Delft University of Technology*; Alessandro Brighente, *SPRITZ Security and Privacy Research Group, Università di Padova*; Mauro Conti, *SPRITZ Security and Privacy Research Group, Università di Padova & Cyber Security Group, Delft University of Technology*; Zekeriya Erkin, *Cyber Security Group, Delft University of Technology*

<https://www.usenix.org/conference/usenixsecurity24/presentation/palazzo>

This paper is included in the Proceedings of the
33rd USENIX Security Symposium.

August 14-16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Proceedings of the
33rd USENIX Security Symposium
is sponsored by USENIX.

Privacy-Preserving Data Aggregation with Public Verifiability Against Internal Adversaries

Marco Palazzo*, Florine W. Dekker*, Alessandro Brighente†, Mauro Conti†*, Zekeriya Erkin*

* *Cyber Security Group, Delft University of Technology*

† *SPRITZ Security and Privacy Research Group, Università di Padova*

Abstract

We consider the problem of publicly verifiable privacy-preserving data aggregation in the presence of a malicious aggregator colluding with malicious users. State-of-the-art solutions either split the aggregator into two parties under the assumption that they do not collude, or require many rounds of interactivity and have non-constant verification time.

In this work, we propose mPVAS, the first publicly verifiable privacy-preserving data aggregation protocol that allows arbitrary collusion, without relying on trusted third parties during execution, where verification runs in constant time. We also show three extensions to mPVAS: mPVAS+, for improved communication complexity, mPVAS-IV, for the identification of malicious users, and mPVAS-UD, for graceful handling of reduced user availability without the need to redo the setup. We show that our schemes achieve the desired confidentiality, integrity, and authenticity. Finally, through both theoretical and experimental evaluations, we show that our schemes are feasible for real-world applications.

1 Introduction

Data aggregation refers to the collection of data from one or more sources and its processing by a central *aggregator* for statistical analysis. These protocols find applications in many situations concerning sensitive data, such as automated power delivery and balancing mechanisms in smart grids [15, 11], patient monitoring [44], and mobile computing [23, 41]. While the benefits brought by data aggregation are evident, without proper countermeasures, they may represent a threat to the privacy of users [17, 33, 24]. Malicious actors could exploit the collected data to profile or track users' activities, social status, religious beliefs, and medical conditions [17, 33, 24]. Therefore, data aggregation protocols should guarantee users' privacy in the presence of malicious actors.

Drawbacks of current solutions. “Classical” privacy-preserving data summation protocols [13, 40, 28] assume that participants are honest-but-curious: They follow the protocol

but may try to infer others' sensitive information. When users may also attempt to influence the correctness of the output, zero-knowledge proofs can be used to prove that users' inputs are well-formed [14, 26]. When aggregators may influence correctness, the problem is more difficult. The aforementioned protocols typically assume that the legal and reputational consequences of malicious behavior deter the aggregator from publishing a forged aggregate. However, without efficient methods to actually detect such tampering, these protocols may fail to prevent malicious behaviour.

A simple technique to detect tampering by the aggregator is to have each user sign their submitted value. However, this incurs high verification costs as possibly thousands of signatures need to be verified in applications with many users. A more common technique adopted in the literature is to make only the result of the aggregation verifiable using privacy-preserving verifiable summation protocols [30, 35, 3, 21, 45, 22]. Unfortunately, both solutions fail to guarantee unforgeability when the malicious aggregator may collude with malicious users.

To the best of our knowledge, only three works have considered privacy-preserving verifiable summation against a malicious aggregator colluding with malicious users [34, 29, 37]. Leontiadis and Li [29] and Mouris and Tsoutsos [34] both assume a two-aggregator model in which at most one aggregator may collude with malicious users. Unfortunately, as we describe in Section 3.3, the work by Leontiadis and Li [29] contains a mistake that breaks unforgeability. Finally, Ren et al. [37] propose a single-server protocol in which the aggregate is hidden from the aggregator. Unfortunately, their protocol fails to provide confidentiality for small plaintext spaces, and verification time is linear in the number of users.

Contributions. We present mPVAS, the first privacy-preserving publicly verifiable summation protocol that allows for arbitrary collusions between a malicious aggregator and malicious users, requiring only a single server and constant-time verification. Note that data poisoning attacks from the users are outside the scope of this paper.

Our contributions can be summarized as follows.

- We propose a publicly verifiable aggregate signature scheme considering malicious users and aggregators (mPVAS), a novel signature scheme that allows users to sign their reports and compute a signature over the sum of the private values.
- We present three extensions to mPVAS. mPVAS+ reduces communication overhead in a slightly weaker adversarial model, mPVAS-IV allows the detection and removal of malicious users, and mPVAS-UD allows users to exit the protocol without necessitating a new setup phase for the other users.
- We provide theoretical evaluations of the security and performance of our protocols, as well as a practical analysis of their performance using a proof-of-concept implementation. Our results show that mPVAS and its extensions are practical for real-world scenarios.

Outline. In [Section 2](#), we present the system model and our assumptions. In [Section 3](#), we discuss related works. In [Section 4](#), we introduce the building blocks of our schemes. In [Section 5](#), we introduce mPVAS. In [Section 6](#), we introduce mPVAS+, which reduces communication complexity. In [Section 7](#), we introduce mPVAS-IV, which adds input validation to combat malicious users. In [Section 8](#), we introduce mPVAS-UD, which adds support for user dropouts. In [Section 9](#), we evaluate all four protocols. Finally, in [Section 10](#), we present our conclusions.

2 System Model and Assumptions

The goal of our protocol is to publish the (authenticated) sum of all users' private values in round t , subject to two properties: individual users' values remain unknown to other parties (*confidentiality*), and the published sum is guaranteed to match the true sum (*unforgeability*). Note that unforgeability implies both integrity and authenticity [42].

We assume all adversaries are probabilistic and polynomially time-bounded. Furthermore, similar to related work [3, 29], we assume availability: Parties do not intentionally try to make the protocol fail (denial of service), and do not unexpectedly drop out. We loosen this assumption in [Section 7](#) and [Section 8](#), where we provide extensions for availability.

The following parties participate in the protocol.

Aggregator. The aggregator collects users' inputs and signatures, and publishes the input sum and an aggregate signature of the sum. The aggregator is malicious and may collude with other malicious parties. That is, the aggregator may deviate from the protocol in arbitrary ways, for example to learn users' private values, tamper with signatures, or publish an incorrect aggregate.

Users. We consider a set of n users $\mathbb{U} = \{1, 2, \dots, n\}$. In any round t , each user $i \in \mathbb{U}$ holds some private integer $x_{i,t}$. We assume at most $k \leq n - 2$ users are malicious and may

collude with the aggregator. The remaining $n - k \geq 2$ users are honest-but-curious; these users follow the protocol, but may still try to obtain private data without colluding. Finally, all users have access to a synchronized clock indicating the current round t .

Verifier. Verifiers check that the aggregator's published output is correct. Any party may be a verifier; this includes external auditors, the aggregator, users, the dealer, and system administrators. We assume there is at least one verifier.

Dealer. We require a trusted dealer to set up the system, similar to nearly all related works [30, 35, 3, 21, 45, 22, 29, 37]. Though a fully trusted party is a strong assumption, we argue that it is feasible in relevant applications such as smart grids and medical data sharing, where the role can be fulfilled by a trusted institution or hardware manufacturer. The dealer is tasked with generating and distributing the public and private parameters to the other parties. After the setup, the dealer exits the protocol.

Communication. The dealer and aggregator both have direct communication channels with all users and verifiers, and with each other. These channels provide secrecy, authenticity, and integrity. Users cannot interact with each other directly but can ask the aggregator to forward messages for them.

3 Related Work

There is a large body of work on privacy-preserving computation. In this section, we discuss why these works cannot be trivially adapted to the adversarial model presented in [Section 2](#). In [Section 3.1](#), we briefly discuss protocols for general verifiable computation. Then, in [Section 3.2](#), we discuss protocols for non-verifiable summation. Finally, in [Section 3.3](#), we discuss protocols for verifiable summation, which we also summarise in [Table 1](#).

3.1 General verifiable computation

In general verifiable computation [18], the aggregator computes an arbitrary function over users' data, while learning neither the function nor its inputs. Users can verify that the output is correct, without learning others' values.

Gordon et al. [19] prove that general verifiable computation is impossible if the aggregator colludes with users, even when only a single user colludes, this user is honest-but-curious, and the protocol uses a trusted setup. Therefore, general verifiable computation is not a suitable solution for our adversarial model. Note that since the above impossibility result requires that the function remains private, this does not preclude verifiable privacy-preserving summation in this adversarial model.

3.2 Non-verifiable summation

Privacy-preserving summation [28, 40, 7, 5] ensures confidentiality and availability in a variety of adversarial models.

Table 1: Overview of privacy-preserving verifiable summation works and their properties. The symbols \bigcirc , \bigcirc , and \bullet respectively denote that a property is not, is partially, or is fully achieved by a particular work. The symbol – denotes that a property is not applicable. We abbreviate “aggregator” to “agg.”

	Trusted setup ¹	Single agg.	Malicious users	Verifiable by			Agg. colludes with	
				Users	Agg.	Verifiers	Users	Verifiers
[30]	\bullet	\bullet	\bigcirc	\bullet	\bullet	\bullet	\bigcirc	\bigcirc
[35]	\bullet ²	\bullet	\bigcirc	\bigcirc	\bigcirc	\bigcirc ³	\bigcirc	\bigcirc
[3]	\bullet	\bullet	\bigcirc	\bullet	\bullet	\bullet	\bigcirc	\bigcirc
[21]	\bullet	\bullet	\bigcirc	\bullet	\bullet	\bigcirc	\bigcirc	–
[45]	\bullet	\bigcirc	\bigcirc	\bullet	\bullet	\bigcirc	\bigcirc	–
[22]	\bullet	\bullet	\bigcirc	\bullet	\bullet	\bigcirc	\bigcirc	–
[29]	\bullet	\bigcirc	\bullet	\bigcirc	\bigcirc	\bigcirc ³	\bigcirc ⁴	\bigcirc
[34]	\bigcirc ⁵	\bigcirc	\bullet	\bullet	\bullet	\bullet	\bigcirc ⁴	\bigcirc ⁴
[37]	\bullet	\bullet	\bullet	\bullet	\bigcirc	\bigcirc	\bullet	–
mPVAS	\bullet	\bullet	\bullet	\bullet	\bullet	\bullet	\bullet	\bullet
mPVAS+	\bullet	\bullet	\bullet	\bullet	\bullet	\bullet	\bullet	\bullet
mPVAS-UD	\bullet	\bullet	\bullet	\bullet	\bullet	\bullet	\bullet	\bullet
mPVAS-IV	\bullet	\bullet	\bullet	\bullet	\bullet	\bullet	\bullet	\bullet

¹ Also includes public-key infrastructure and common reference string. ² Requires trusted party in all phases of protocol. ³ Exactly one (trusted) verifier. ⁴ May collude with at most one aggregator. ⁵ Requires public ledger in all phases of protocol.

However, these works are not verifiable. That is, if a malicious aggregator publishes an arbitrary value as the sum, this cannot be detected by other parties. Therefore, these protocols are insufficient when the aggregator has an incentive to lie.

3.3 Verifiable summation

With privacy-preserving verifiable summation, the aggregator’s output can be proven to be the sum of users’ inputs.

We first discuss protocols for honest-but-curious users and then discuss protocols for malicious users. The aggregator is necessarily assumed malicious. We restrict our discussion to the verification techniques, ignoring the protocols’ summation mechanisms. We summarise our results in Table 1.

Honest-but-curious users. Given honest-but-curious users and a malicious aggregator, the aggregator must prove that the published sum indeed corresponds to the users’ inputs.

Early works [30, 35, 3] rely on a shared secret between the users and the verifier to ensure only authenticated parties can sign, and rely on a signature key that is secret-shared between the users to ensure a signature is valid only if all users are included. These protocols cannot ensure unforgeability when the aggregator colludes with honest-but-curious users, because if a user sends the aggregator the shared secret used for authentication, the aggregator can homomorphically modify valid signatures.

Recent works [21, 45, 22] use the same high-level ideas, but combine this with the non-verifiable summation protocol

of Bonawitz et al. [7] to achieve reliability when users unexpectedly drop out. Each of these works similarly cannot ensure unforgeability when the aggregator colludes with users. We point out to interested readers that two of the above works have received security fixes [20, 32]. There are more works that achieve privacy-preserving verifiable summation with honest-but-curious users, but none that do not fit the above general descriptions.

Malicious aggregator and malicious users. To the best of our knowledge, only a few works tackle the problem of privacy-preserving verifiable summation with a malicious aggregator and malicious users.

Leontiadis and Li [29] propose the addition of a new honest-but-curious party, the converter. Users work with the converter to create homomorphic commitments of their data based on shares of the verifier’s secret key. The aggregator then aggregates users’ private data and their commitments (respectively), and sends both to the verifier. Finally, the verifier checks that the aggregation was done correctly. Unfortunately, this protocol is not truly publicly verifiable, since the verification key cannot be shared with users of the protocol. Furthermore, if a user, aggregator, and converter collude, unforgeability no longer holds. Finally, it appears that the protocol is flawed: If a malicious user sends the converter a commitment to 0 and the user then forwards the converter’s response to the aggregator, the aggregator can create arbitrary valid signatures.

Mouris and Tsoutsos [34] propose splitting the aggregator into two parties: a curator and an analyst. Both may be

malicious, but they do not collude, and only the analyst has the decryption key for the aggregate. Users homomorphically encrypt their data and send it to the curator, and publish a homomorphic commitment to the ciphertext on a public ledger. The curator verifies that the received ciphertexts correspond to the commitments on the public ledger, and then publishes an aggregate ciphertext and an aggregate commitment on the public ledger. Finally, the analyst verifies the aggregate commitment, decrypts the aggregate ciphertext, and publishes the aggregate data together with a proof of correct decryption on the public ledger. The protocol requires that the curator and analyst do not collude; otherwise, the protocol cannot guarantee confidentiality and unforgeability. Furthermore, users colluding with the curator may affect correctness.

Ren et al. [37] propose a summation protocol that provides confidentiality, unforgeability, and availability against a malicious aggregator colluding with a malicious subset of clients. The protocol has four major drawbacks. First, only the users learn the obtained sum, whereas the aggregator learns nothing. Second, only participating users can verify the obtained sum, and there is no trivial extension to allow external parties to learn and verify the sum. Third, if the plaintext space is small (as in verifiable summation for smart meters [40, 30]), confidentiality can be broken by brute-forcing commitments. Finally, verification time is linear in the number of users.

4 Preliminaries

Before we present mPVAS in Section 5, we introduce its basic building blocks. We follow the definitions in [27].

Bilinear pairings. Given cyclic groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, each of the same prime order p , a bilinear pairing is a function $e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ such that, for any $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$,

$$e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}. \quad (1)$$

Furthermore, $e(g_1, g_2)$ should be a generator of \mathbb{G}_T , and e should be efficiently computable. This excludes so-called degenerate bilinear pairings, in which $e(g_1, g_2) = 1$ for all $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. Finally, we assume that the SXDH assumption [2] holds, i.e. that the decisional Diffie-Hellman assumption holds in both \mathbb{G}_1 and \mathbb{G}_2 , and that there exist no efficiently computable homomorphisms between the two.

Zero-knowledge proof of equality between commitments. We describe ZKPEq, a zero-knowledge proof that two different Pedersen commitments share the same committed value. Formally, given commitments $C(x, r_1)$ and $C(x, r_2)$, ZKPEq proves the relation

$$\{(x, y, z): S = g_1^x h_1^{r_1} \wedge T = g_2^x h_2^{r_2}\}. \quad (2)$$

This proof can be implemented as an EQ-composition on the common witness x of two Okamoto protocols [36] running in parallel [39]. ZKPEq can be made non-interactive using the Fiat-Shamir heuristic [16].

5 mPVAS: Publicly Verifiable Aggregate Signatures with Malicious Users and a Malicious Aggregator

We present mPVAS, a novel aggregate signature scheme for summations. mPVAS can be used to verify that the output of a separate summation protocol was not tampered with by the aggregator. The core idea behind mPVAS is to create commitment-like signatures of the inputs and wrap each signature under a common secret exponent s , similar to other verifiable schemes [3, 30, 31]. Unlike other schemes, however, we allow users to collude with the aggregator by revealing their private parameters. mPVAS guarantees unforgeability of the aggregate signature given at most k malicious users. To achieve this, we use Shamir secret sharing over s with a threshold of $k + 1$.

mPVAS runs in *four phases*: setup, signing, aggregation, and verification. During setup, the participants interactively determine the scheme's public and private parameters. During signing, users cooperatively calculate signatures of their inputs to a separate summation protocol. During aggregation, the aggregator combines users' signatures into a single signature. Finally, during verification, verifiers compare the aggregate signature with the summation protocol's output.

mPVAS provides only an aggregate signature and, for large plaintext spaces, must operate adjacent to a separate privacy-preserving summation scheme. The order of operations is that mPVAS runs up to (but excluding) verification, then the summation protocol reveals the sum and, finally, mPVAS verifies correctness. Alternatively, if the plaintext space is small enough, the sum can be extracted in polynomial time from the aggregate signature itself by repeated verification on all possible values.

Data poisoning attacks are outside the scope of this paper. However, range proofs can be used during the signing phase of the protocol to partially mitigate these types of attacks.

5.1 Setup

During the setup, the trusted dealer chooses and publishes the public parameters $pp = (H, H_1, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, p, n, k)$, generated according to a strong security parameter λ . Each \mathbb{G}_i is a cyclic group of order p , where p is a large prime number. g_1 and g_2 are random generators of \mathbb{G}_1 and \mathbb{G}_2 , respectively. $e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a type-3 bilinear pairing in which the Symmetric External Diffie-Hellman (SXDH) [2] assumption holds. Furthermore, $H: \{0, 1\}^* \rightarrow \mathbb{G}_1$ and $H_1: \{0, 1\}^* \rightarrow \mathbb{G}_1$ are two distinct and cryptographically-secure hash functions. Finally, n is the number of users, and $k \leq n - 2$ is the maximum number of malicious users.

The dealer assigns each user a unique identifier $i \in \{1 \dots n\}$, chooses a secret $s \leftarrow \mathbb{Z}_p$, and creates n secret shares $[s]_i$ using $(k + 1)$ -out-of- n Shamir secret sharing. Recall that each Shamir secret share consists of a coordinate (x_i, y_i) on the 2D

plane. The dealer ensures that the x -coordinates correspond exactly to the user identifiers, and defines $[s]_i = y_i$. Next, for each $i \in \{1 \dots n\}$ and each $j \in \{1 \dots k+1\}$, the dealer chooses encryption key $ek_{i,j} \leftarrow \mathbb{Z}_p$, but sets the last encryption key to

$$ek_{n,k+1} = -\sum_{i=1}^n \sum_{j=1}^k ek_{i,j}. \quad (3)$$

It follows that the sum of all encryption keys is 0. The dealer then sends $(pp, i, [s]_i, \{ek_{i,j} \mid 1 \leq j \leq k+1\})$ to the corresponding user i .

Each user i generates a signature key $sk_i \leftarrow \mathbb{Z}_p$ and sends it to the dealer.

Once all n signature keys have been received, the dealer calculates the verification key tuple

$$vk = \left((g_2^s)^{\sum_{i=1}^n sk_i}, g_2^s \right), \quad (4)$$

sends (pp, vk) to each verifier, and then leaves the protocol.

5.2 Signing

User i creates a signature of their private input $x_{i,t}$ in round t of the summation protocol using the following interactive four-step procedure.

1) Create initial signature. User i computes their initial signature for round t as

$$\sigma_{i,t}^1 = H(t)^{sk_i} g_1^{x_{i,t}} \in \mathbb{G}_1. \quad (5)$$

and sends it to the aggregator.

2) Create partial signatures. The aggregator forwards the initial signature tuple of each user i to an arbitrary¹ set \mathcal{U}_i of k users other than user i . We call \mathcal{U}_i the signing set of user i . The aggregator also sends the list of identifiers \mathcal{U}_i to user i . After receiving the initial signature of user i , each user $j \in \mathcal{U}_i$ computes

$$\sigma_{i,t}^{2,j} = H_1(t)^{ek_{j,i}} (\sigma_{i,t}^1)^{[s]_j^*} \quad (6)$$

$$= H_1(t)^{ek_{j,i}} (H(t)^{sk_i} g_1^{x_{i,t}})^{[s]_j^*} \in \mathbb{G}_1, \quad (7)$$

where (in a minor abuse of notation) $ek_{j,i}$ denotes the encryption key that user j chooses to uniquely associate with user i , and $[s]_j^*$ denotes the partial reconstruction of user j 's Shamir secret share of s , computed via interpolation by multiplying $[s]_j$ with the Lagrange basis polynomial corresponding to user j . User j then sends $\sigma_{i,t}^{2,j}$ to the aggregator.

3) Sum secret shares. Once the aggregator has received k partial signatures $\sigma_{i,t}^{2,j}$ for user i , the aggregator combines the shares in the exponent by computing

$$\sigma_{i,t}^3 = \prod_{j \in \mathcal{U}_i} (\sigma_{i,t}^{2,j}) = H_1(t)^{\sum_{j \in \mathcal{U}_i} ek_{j,i}} (\sigma_{i,t}^1)^{\sum_{j \in \mathcal{U}_i} [s]_j^*} \quad (8)$$

$$= H_1(t)^{\sum_{j \in \mathcal{U}_i} ek_{j,i}} (H(t)^{sk_i} g_1^{x_{i,t}})^{\sum_{j \in \mathcal{U}_i} [s]_j^*} \in \mathbb{G}_1. \quad (9)$$

¹In our evaluation, we assume that the signing set consists of the next k users after user i when ordered by their numerical identifier.

The aggregator then sends $\sigma_{i,t}^3$ back to user i .

4) Compute final user signature. At this point, k secret shares of s have been added to the exponent. Adding one more secret share therefore reconstructs s in the exponent. User i computes the final user signature as

$$\sigma_{i,t} = H_1(t)^{ek_{i,i}} \cdot \sigma_{i,t}^3 \cdot (H(t)^{sk_i} g_1^{x_{i,t}})^{[s]_i^*} \quad (10)$$

$$= H_1(t)^{ek_{i,i} + \sum_{j \in \mathcal{U}_i} ek_{j,i}} (H(t)^{sk_i} g_1^{x_{i,t}})^s \in \mathbb{G}_1, \quad (11)$$

where $ek_{i,i}$ is the single remaining unused encryption key. User i submits their final user signature $\sigma_{i,t}$ to the aggregator.

Note that this signature cannot be verified using the verification key, because this key only works for aggregated signatures. This is intentional, as verifying individual user signatures would trivially allow an adversary to learn the private input of a user by brute force.

5.3 Signature Aggregation

After having received the final user signatures of all users for round t , the aggregator computes the aggregate signature

$$\sigma_t = \prod_{i=1}^n \sigma_{i,t} \quad (12)$$

$$= H_1(t)^{\sum_{i=1}^n \sum_{j=1}^{k+1} ek_{i,j}} (H(t)^s)^{\sum_{i=1}^n sk_i} (g_1^s)^{\sum_{i=1}^n x_{i,t}} \quad (13)$$

$$= (H(t)^s)^{\sum_{i=1}^n sk_i} (g_1^s)^{\sum_{i=1}^n x_{i,t}} \in \mathbb{G}_1. \quad (14)$$

The aggregator sends σ_t to each verifier.

Only at this point should the adjacent summation protocol reveal the sum of users' inputs.

5.4 Verification

Once the aggregator has published the sum of all $x_{i,t}$ and the aggregate signature σ_t , each verifier checks the equation

$$e(H(t), vk_1) e\left(g_1^{\sum_{i=1}^n x_{i,t}}, vk_2\right) \quad (15)$$

$$= e\left(H(t), (g_2^s)^{\sum_{i=1}^n sk_i}\right) e\left(g_1^{\sum_{i=1}^n x_{i,t}}, g_2^s\right) \quad (16)$$

$$\stackrel{?}{=} e(\sigma_t, g_2). \quad (17)$$

5.5 Security Analysis of mPVAS

We show that the verification procedure is correct, does not leak private data, and cannot be fooled into accepting an incorrect signature.

To see that verification succeeds for a correct signature, observe that

$$e(\sigma_t, g_2) = e\left((H(t)^s)^{\sum_{i=1}^n sk_i} (g_1^s)^{\sum_{i=1}^n x_{i,t}}, g_2\right) \quad (18)$$


$$= e\left((H(t)^s)^{\sum_{i=1}^n sk_i}, g_2\right) e\left((g_1^s)^{\sum_{i=1}^n x_{i,t}}, g_2\right) \quad (19)$$

$$= e\left(H(t), (g_2^s)^{\sum_{i=1}^n sk_i}\right) e\left(g_1^{\sum_{i=1}^n x_{i,t}}, g_2^s\right) \in \mathbb{G}_T. \quad (20)$$

Theorem 1. *mPVAS is Aggregator Oblivious.*

Proof. See [Appendix A](#). 

Theorem 2. *mPVAS is Aggregate Unforgeable against Type-I and Type-II forgeries.*

Proof. Intuitively, because the aggregator does not know s , they cannot create a correct signature for a sum other than the published one. For the full proof, see [Appendix B](#). 

6 mPVAS+: mPVAS with Lower Communication Overhead

In mPVAS (see [Section 5](#)), communication complexity is linear in the number of malicious users k . Though we assume malicious users to be in the minority, this level of interactivity may be too high for some applications. We present mPVAS+, an extension of mPVAS to significantly reduce communication complexity. Recall that we provide a runtime analysis of mPVAS and all extensions in [Section 9](#).

We show that we can significantly decrease the communication complexity using a divide-and-conquer strategy. Intuitively, our solution works by dividing users into random groups of size $c \leq k$ and providing each group with an independent set of secret shares of s . Since each group can now individually reconstruct s in the exponent, we can eliminate cross-group communication. As long as at least one user in each group is non-malicious, adversaries cannot reconstruct s . We provide a statistical analysis that this holds in [Section 6.3](#).

6.1 Modifications in mPVAS+

We describe how mPVAS+ differs from mPVAS.

Setup. The key difference with the setup of mPVAS (see [Section 5.1](#)) is that in mPVAS+, instead of creating a single sharing over all users, the dealer randomly assigns users to groups of size $c \leq k$ and, for each group, generates c -out-of- c Shamir secret shares of s . If c does not divide n , then $n \bmod c$ arbitrary groups should have one additional user, and the secret sharing threshold of this group is adjusted accordingly. After having chosen the random secret $s \leftarrow \mathbb{Z}_p$ (as in mPVAS), the dealer creates separate c -out-of- c Shamir secret shares of s for each group. Because a separate set of shares is created for each group, shares from different groups cannot be combined together. As in mPVAS, each share $[s]_i$ is sent to the corresponding user i . Furthermore, the dealer creates only nc encryption keys instead of $n(k+1)$. In mPVAS+, the list of other users in the group is additionally sent to user i .


Signing. In this phase, the only difference with mPVAS (see [Section 5.2](#)) is that the aggregator sends the initial signature $\sigma_{i,t}^1$ of each user i to the $c-1$ other users in user i 's group, rather than sending them to k arbitrary other users.

Aggregation. The aggregation phase remains unchanged (see [Section 5.3](#)).


Verification. The verification phase remains unchanged (see [Section 5.4](#)).

6.2 Security Analysis of mPVAS+

Theorem 3. *mPVAS+ is Aggregator Oblivious.*

Proof. The mPVAS+ extension changes the behavior of the base mPVAS scheme. When c malicious users end up in the same group, they can collectively reconstruct s and share it with the aggregator, thus allowing it to tamper with the signatures of honest users. However, note that even with knowledge of s , the aggregator still cannot learn the private values of individual users because they are also blinded by the secret factor $H(t)^{sk_i}$. When s is known by the aggregator, the mPVAS scheme directly reduces to the PPATS scheme of [\[40\]](#), which is Aggregator Oblivious. 

Theorem 4. *mPVAS+ is Aggregate Unforgeable against Type-I and Type-II forgeries.*

Proof. The mPVAS+ scheme can be seen as multiple instances of the regular mPVAS scheme running on multiple groups of users, but with different instances of Shamir secret sharing used to generate the secret shares of s . Thus, Aggregate Unforgeability still holds following the same logic presented in [Section 5.5](#) for the mPVAS scheme, as long as each group contains at least one honest user. We provide a statistical analysis that this requirement holds in [Section 6.3](#). 

We emphasise that mPVAS+ provides Aggregate Unforgeability with k malicious users only if we assume non-adaptive corruptions. Otherwise, the security of mPVAS+ is downgraded to that of an mPVAS instance with $k = c - 1$.

6.3 Statistical Analysis of mPVAS+

The communication complexity of mPVAS+ is better than that of mPVAS only if $c \leq k$. However, unlike mPVAS, in mPVAS+ it is possible that at least one group consists of adversaries only. These adversaries may then collude to retrieve private key material. We give an exact formula for this probability, and show that it can be made negligibly small.

Let n be the number of users, k the number of malicious users, c the group size, and $d = \text{floor}(n/c)$ the number of groups. We assume that c divides n exactly. Otherwise, $n - cd$ groups should be given one user more, and the following calculations give an upper bound rather than an exact value.

We calculate the probability using a combinatorial counting argument. We model the process of dividing users into groups as first dividing all n users into groups, and then randomly (non-adaptively) corrupting k users. We count the number of instances in which at least one group is fully corrupted, and

divide this by the total number of instances.² The total number of instances (the denominator) is simply $\binom{n}{k}$ (i.e. the binomial coefficient “ n choose k ”), but the number of problematic instances (the numerator) is harder to compute.

Intuitively, the numerator is the number of combinations in which *exactly* one group is fully compromised (which is d , since there are d groups), multiplied by the number of ways in which the remaining $n - c$ users can contain $k - c$ corruptions (which is $\binom{n-c}{k-c}$), *seemingly* giving the probability

$$\frac{d \binom{n-c}{k-c}}{\binom{n}{k}}. \quad (21)$$

However, this is inaccurate, because if the remaining users also fully corrupt a group, that case is counted twice. In fact, duplicates are counted twice, triplicates are counted thrice, and, in general, r -replicates are counted r times. Luckily, by the inclusion-exclusion principle, it suffices to separately count and subtract these cases.

Let $R = \text{floor}(\frac{k}{c})$ denote the “replicity”, which is the highest order of replication. To determine the number of r -replicates, we first define a helper function that counts the number of r -replicates *after fixing which r groups are fully corrupted*:

$$\text{rep}(r) = \binom{n-rc}{k-rc} - \sum_{i=r+1}^R \left(\binom{d-r}{i-r} \cdot \text{rep}(i) \right). \quad (22)$$

This function counts the number of ways to corrupt remaining users and then recursively subtracts higher-order replicates. The number of recursive i -replicates is found by first fixing $r - i$ additional groups and then multiplying by $\text{rep}(i)$.

We conclude that the probability that at least one group is fully corrupted is exactly

$$\frac{d \cdot \binom{n-c}{k-c} - \sum_{r=2}^R ((r-1) \cdot \binom{d}{r} \cdot \text{rep}(r))}{\binom{n}{k}} \quad (23)$$

if c divides n , and is a strict upper bound otherwise. As in (22), for each order of replication, we multiply by the number of ways to choose r groups, and additionally multiply by $r - 1$ to actually the multiply-counted items.

We visualize (23) for various values of n , k , and c in Figure 1. The figure shows that, given sufficient users, the probability of accidentally assigning only malicious users to a group can be made negligible with an appropriate choice of c . For example, with 50 users, of which 20% malicious, choosing $c = 7$ gives a probability of approximately 0.000841%, and can be made even smaller.

²We calculate the probability as a combinatorial problem. Modeling this as a permutation instead would require counting all possible ways to assign identity to users after fixing a specific combination. This can be done by multiplying both the numerator and denominator by $(n - k)!k!$. Therefore, both methods give the same result.

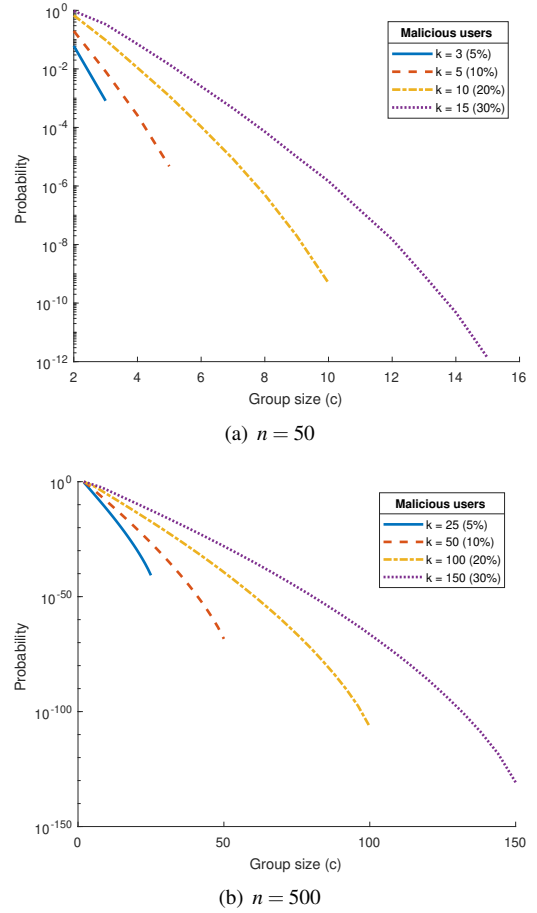


Figure 1: Probability that at least one group is fully corrupted in mPVAS+. Due to the logarithmic y-axis, lines end when they reach zero.

7 mPVAS-IV: mPVAS with Input Validation

We have thus far assumed that parties do not try to make the protocol fail. In this section, we present mPVAS-IV, an extension to mPVAS (see Section 5) to allow the aggregator to identify and remove users that attempt to cause an invalid aggregate signature. mPVAS-IV is fully compatible with both mPVAS+ (see Section 6) and mPVAS-UD (see Section 8).

In terms of our adversarial model (see Section 2), we loosen our assumptions on users, who may now send ill-formed messages with the intent of causing verification to fail. We model the aggregator as a service provider, who may attempt to obtain users’ private data or output a falsified signature, but is expected to ensure well-formed outputs so as to not disrupt users reliant on their services. As such, the malicious aggregator may still collude with malicious users against confidentiality and unforgeability, but the aggregator acts honestly with regard to availability. Furthermore, we assume that the adjacent summation protocol ensures availability in this model, including *verified* commitments to users’ summation inputs.

mPVAS-IV validates that users' inputs are well-formed, and pinpoints which user caused the invalid input. That user is then barred from participating in future instances of the protocol, and the protocol is restarted from scratch with the remaining users. In the worst case, malicious users are removed one at a time, requiring k restarts. After that, all malicious users have been removed, and users can continue additional rounds indefinitely. Therefore, whereas in mPVAS adversaries could prevent all output without end, mPVAS-IV reduces this adversarial capability to a linearly bounded overhead.

At its core, mPVAS-IV adds a mechanism to validate individual final user signatures when verification fails. However, since users may tamper with others' signatures, we must also add a detection mechanism there. Note that we do not need to validate the initial signature, since either the subsequent final user signature is valid and there is no problem to begin with, or the final user signature is invalid and is detected as such.

7.1 Modifications in mPVAS-IV

We describe how mPVAS-IV differs from mPVAS.

Setup. The setup phase of mPVAS-IV starts by running the setup phase of mPVAS (see Section 5.1). The dealer then generates additional information to perform input validation.

First, the dealer chooses g_T and h_T as random generators of \mathbb{G}_T , and, for each user i , generates $r_i \leftarrow \mathbb{Z}_p$, and the set of values $EK_{j,i} = g_2^{ek_{j,i}}$. The dealer sends r_i , $EK_{j,i} = g_2^{ek_{j,i}}$, and g_2^s to each user i .

Next, for each user i , the dealer generates $SK_i = g_T^{sk_i} h_T^{r_i}$, $SS_i = g_2^{[s]_i}$, $EK_{j,i} = g_2^{ek_{j,i}}$, for $j \in \{1 \dots k\}$, $EKS_{j,i} = g_2^{\frac{ek_{j,i}}{s}}$, for $j \in \{1 \dots k+1\}$, and sends these values to the aggregator, along with $v = g_2^{\frac{1}{s}}$.

Finally, the dealer sends the additional public parameters (g_T, h_T) to all participants.

Signing. We require additional operations before and after the regular signing phase of mPVAS (see Section 5.2).

Before the regular signing phase, we require that the adjacent summation scheme outputs Pedersen commitments of the users' inputs to that summation scheme. This is to ensure consistency of the inputs between the two schemes. If mPVAS-IV is used without a separate summation scheme, the committed values should be validated using range proofs instead. Either way, let $C(x_{i,t})$ for each user i denote these commitments.

After the regular signing phase is complete, each user checks their final user signature for tampering. If tampering is detected, the user informs the aggregator, who then validates the corresponding partial signatures. If the aggregator also detects tampering, the aggregator marks the user(s) who sent that partial signature as malicious. Otherwise, if the aggregator does not detect tampering, the reporting user is instead marked as malicious. After that, the protocol restarts without the detected malicious users.

User i checks their final user signature $\sigma_{i,t}$, calculated in (10), for tampering by checking that

$$e(H_1(t), g_2^{\varepsilon_i}) e(H(t), (g_2^s)^{sk_i}) e(g_1^{x_{i,t}}, g_2^s) \stackrel{?}{=} e(\sigma_{i,t}, g_2) \quad (24)$$

holds, where $\varepsilon_i = \sum_{j=1}^{k+1} ek_{j,i}$. In a nutshell, the left-hand side re-calculates user i 's expected final user signature under a bilinear mapping (compare with (10)), while the right-hand side bilinearly maps the actual final user signature. If (24) does not hold, the user reports this to the aggregator.

Since the aggregator forwards all partial signatures, the aggregator possesses the initial signature $\sigma_{i,t}^1$, as well as the partial signature $\sigma_{i,t}^{2,j}$ of each user $j \in \mathcal{U}_i$. Upon receiving user i 's claim that their signature was tampered with, the aggregator computes for each user $j \in \mathcal{U}_i$ the value

$$\sigma_{i,t}^{2,j*} = e\left(H_1(t), g_2^{ek_{j,i}}\right) e\left(\sigma_{i,t}^1, g_2^{[s]_j^*}\right), \quad (25)$$

where $[s]_j^*$ is as in (6), i.e. the partial reconstruction of user j 's Shamir secret share of s , here calculated in the exponent of g_2 using the set SS . Finally, the aggregator validates the partial signature $\sigma_{i,t}^{2,j}$ by checking

$$\sigma_{i,t}^{2,j*} \stackrel{?}{=} e\left(\sigma_{i,t}^{2,j}, g_2\right) \quad (26)$$

$$= e\left(H_1(t)^{ek_{j,i}} \left(H(t)^{sk_i} g_1^{x_{i,t}}\right)^{[s]_j^*}, g_2\right) \quad (27)$$

$$= e\left(H_1(t)^{ek_{j,i}}, g_2\right) e\left(\left(H(t)^{sk_i} g_1^{x_{i,t}}\right)^{[s]_j^*}, g_2\right) \quad (28)$$

$$= e\left(H_1(t), g_2^{ek_{j,i}}\right) e\left(\sigma_{i,t}^1, g_2^{[s]_j^*}\right) \quad (29)$$

If this holds, then user j did not act maliciously; otherwise, user j is marked as malicious and expelled from the protocol. The aggregator repeats this process for all users in \mathcal{U}_i , as there can be more than one user behaving maliciously in a single signing set.

Aggregation. The aggregation phase remains unchanged (see Section 5.3).

We assume that the adjacent summation scheme aborts no later than this point if the Pedersen commitment $C(x_{i,t})$ of any user i does not correspond to that user's real input to the summation scheme.

Verification. The verification phase of mPVAS-IV starts by running the verification phase of mPVAS (see Section 5.4). If verification fails, the aggregator tries to find the culprit by verifying that the final user signature $\sigma_{i,t}$ of each user i matches (10). This verification entails removing the term containing the encryption keys ek and then asking user i for a zero-knowledge proof of equality between the expected and the actual value of the remaining term.

To remove the term with the encryption keys, the aggregator first computes

$$g_2^{\frac{\varepsilon_i}{s}} = \prod_{j=1}^{k+1} EK_{j,i} = \prod_{j=1}^{k+1} g_2^{\frac{ek_{j,i}}{s}} \in \mathbb{G}_2, \quad (30)$$

and then removes the term by computing

$$\sigma'_{i,t} = \frac{e\left(\sigma_{i,t}, g_2^{\frac{1}{s}}\right)}{e\left(H_1(t), g_2^{\frac{e_i}{s}}\right)} \quad (31)$$

$$= \frac{e\left(H_1(t)^{e_i} (H(t)^{sk'_i} g_1^{x'_{i,t}})^s, g_2^{\frac{1}{s}}\right)}{e(H_1(t), g_2)^{\frac{e_i}{s}}} \quad (32)$$

$$= \frac{e\left(H_1(t)^{e_i}, g_2^{\frac{1}{s}}\right) e\left(H(t)^{sk'_i} g_1^{x'_{i,t}}, g_2\right)}{e(H_1(t), g_2)^{\frac{e_i}{s}}} \quad (33)$$

$$= e(H(t), g_2)^{sk'_i} e(g_1, g_2)^{x'_{i,t}} \in \mathbb{G}_T, \quad (34)$$

where the values of sk'_i and $x'_{i,t}$ are implied. Finally, the aggregator asks user i to prove that $sk'_i = sk_i$ and $x'_{i,t} = x_{i,t}$. User i does so by interpreting (34) as a Pedersen commitment and providing two ZKPEq proofs (see Section 4): one for proving the equality $sk'_i = sk_i$ between $\sigma'_{i,t}$ and SK_i , and another for proving equality of $x'_{i,t} = x_{i,t}$ between $\sigma'_{i,t}$ and $C(x_{i,t})$.

If mPVAS-IV is used without an adjacent summation protocol, user i must also provide a range proof (such as a Bulletproof [9]) of $x_{i,t}$ to show that their input lies in a restricted range, and that extraction of the sum from σ_t is tractable. Users that fail to send valid proofs are removed from the protocol and subsequent executions.

7.2 Security Analysis

Theorem 5. *mPVAS-IV is Aggregator Oblivious.*

Proof. The additional information received by the aggregator does not yield any advantage to breaking Aggregator Obliviousness. In fact, the secret shares the aggregator receives in the set SS cannot be efficiently extracted due to the hardness of the DLP in \mathbb{G}_2 . The commitments contained in the set SK are hiding, thus the aggregator cannot extract the signing keys either. Furthermore, as with mPVAS, all initial and final user signatures $\sigma^1_{i,t}, \sigma^{2,j}_{i,t}, \sigma^3_{i,t}, \sigma_{i,t}$ contain the secret factor $H(t)^{sk_i}$, which perfectly hides $x_{i,t}$ in \mathbb{G}_1 and prevents the aggregator from exploiting the verification algorithm in (24) to find $x_{i,t}$.

The intermediate value $e(H(t), g_2)^{sk'_i} e(g_1, g_2)^{x'_{i,t}}$ from (34) is also hiding under the random oracle model. Finally, the proof ZKPEq does not leak any information about the private witness due to its zero-knowledge property. We conclude that the aggregator cannot learn the private value of honest users, and thus mPVAS-IV is Aggregator Oblivious. \square

Theorem 6. *mPVAS-IV is Aggregate Unforgeable against Type-I and Type-II forgeries.*

Proof. See Appendix C. \square

8 mPVAS-UD: mPVAS with User Dropouts

Requiring that all users are always online is not feasible for some applications. In this section, we present mPVAS-UD, an extension to mPVAS (see Section 5) to allow users to choose a set of rounds in which they will not participate by sending one or more recovery keys containing the necessary material that would otherwise be missing from those rounds. As with the base mPVAS protocol, mPVAS-UD works as long as at least $k + 2$ users do not drop out of the protocol. mPVAS-UD is fully compatible with both mPVAS+ (see Section 6) and mPVAS-IV (see Section 7).

8.1 Modifications in mPVAS-UD

We describe how mPVAS-UD differs from mPVAS.

Setup. In addition to the regular setup of mPVAS (see Section 5.1), the dealer also sends $EK_j = g_2^{\sum_{i=1}^k ek_{j,i}}$, for each $j \in \{1 \dots n\}$, to all verifiers.

If mPVAS-UD is combined with mPVAS+, then the setup should be adjusted to use c -out-of- c' secret sharing instead of c -out-of- c secret sharing, where $c' \geq c$. This ensures that at most $c' - c$ users in each group can drop out without resulting in incomplete signatures. The statistical analysis in Section 6.3 still applies to c .

Signing. In any round t , before running the regular signing phase (see Section 5.2), each user i has the option of dropping out for a set of rounds \mathcal{T} , possibly including the remainder of the current round t . For each round $\tau \in \mathcal{T}$ from which user i would like to drop out, user i calculates a recovery key

$$rk_{i,\tau} = e\left(H(\tau)^{-sk_i}, g_2^s\right) \in \mathbb{G}_T. \quad (35)$$

User i then sends $m_i = (i \parallel \tau \parallel rk_{i,\tau})$ to the aggregator, who forwards both to the verifiers. Note that the aggregator can aggregate all recovery keys $rk_{i,t}$ together before sending them to the verifiers to save space and reduce the communication overhead.

If mPVAS-UD is used in the adversarial model of mPVAS-IV (see Section 7), we must additionally ensure that user i cannot invalidate signatures of rounds \mathcal{T} . Therefore, user i must prove that the recovery key is well-formed using a zero-knowledge proof that sk_i in $rk_{i,\tau}$ is the same as in the commitment $g_T^{sk_i} h_T^{r_i}$ from the setup of mPVAS-IV. Concretely, user i proves the relation

$$\{(x, y) : S = g_1^x h_1^y \wedge T = g_2^x\}, \quad (36)$$

which can be implemented and made non-interactive similar to ZKPEq (see Section 4).

The signing phase then proceeds as normal, but without the users who have opted to drop out of round t .

Aggregation. The aggregation phase remains unchanged (see Section 5.3).

Verification. The verification phase of mPVAS-UD replaces that of regular mPVAS (see Section 5.4). In round t , let \mathcal{D}_t be the set of users that dropped out, and let \mathcal{R}_t be the set of the remaining users. Since users \mathcal{D}_t do not participate in the adjacent summation protocol of round t , the published sum should be $x_t = \sum_{i \in \mathcal{R}_t} x_{i,t}$, and the aggregate signature should similarly be over that sum. To verify that the signature σ_t is correct, the verifier uses the dropped-out users' recovery keys and checks

$$e(g_1^{x_t}, vk_2) e(H(t), vk_1) \prod_{i \in \mathcal{D}_t} rk_{i,t} \stackrel{?}{=} \frac{e(\sigma_t, g_2)}{e(H_1(t), \prod_{i \in \mathcal{R}_t} EK_i)}. \quad (37)$$

This is essentially a modification of (15) wherein the verifier assumes that dropped-out users input $x_{i,t} = 0$, while compensating for missing information using the recovery material. To see that correctness holds, let $\sigma'_t = (H(t)^{\sum_{i \in \mathcal{R}_t} sk_i} g_1^{x_t})^s$ be the desired signature (see (14)), recall the definition of vk from (4), and observe that on the left-hand side of (37) we find

$$e(g_1^{x_t}, vk_2) e(H(t), vk_1) \prod_{i \in \mathcal{D}_t} rk_{i,t} \quad (38)$$

$$= e(g_1^{x_t}, g_2^s) e(H(t), vk_1) e(H(t), g_2^s)^{-\sum_{i \in \mathcal{D}_t} sk_i} \quad (39)$$

$$= e(g_1^{x_t}, g_2^s) e(H(t), g_2^s)^{\sum_{i \in \mathcal{R}_t} sk_i} \quad (40)$$

$$= e\left((H(t)^{\sum_{i \in \mathcal{R}_t} sk_i} g_1^{x_t})^s, g_2\right) = e(\sigma'_t, g_2). \quad (41)$$

Similarly, on the right-hand side of (37), we find

$$\frac{e(\sigma_t, g_2)}{e(H_1(t), \prod_{i \in \mathcal{R}_t} EK_i)} \quad (42)$$

$$= \frac{e\left(H_1(t)^{\sum_{i \in \mathcal{R}_t} \sum_{j=1}^{k+1} ek_{j,i}} (H(t)^{\sum_{i \in \mathcal{R}_t} sk_i} g_1^{x_t})^s, g_2\right)}{e(H_1(t), \prod_{i \in \mathcal{R}_t} EK_i)} \quad (43)$$

$$= \frac{e(H_1(t), g_2)^{\sum_{i \in \mathcal{R}_t} \sum_{j=1}^{k+1} ek_{j,i}} e(\sigma'_t, g_2)}{e(H_1(t), g_2)^{\sum_{i \in \mathcal{R}_t} \sum_{j=1}^{k+1} ek_{j,i}}} \quad (44)$$

$$= e(\sigma'_t, g_2). \quad (45)$$

8.2 Security Analysis

Theorem 7. *mPVAS-UD is Aggregator Oblivious.*

Proof. See Appendix D. \square

Theorem 8. *mPVAS-UD is Aggregate Unforgeable against Type-I and Type-II forgeries.*

Proof. See Appendix E. \square

9 Complexity Analysis of mPVAS, mPVAS+, mPVAS-IV, and mPVAS-UD

We evaluate the complexity of mPVAS and each of its extensions. In Section 9.1, we present the asymptotic communication complexity of our schemes, and compare this with a selection of related works. In Section 9.2, we describe our experimental setup for empirically determining runtime complexity. After that, we present the results of this analysis for mPVAS in Section 9.3, for mPVAS+ in Section 9.4, for mPVAS-IV in Section 9.5, and for mPVAS-UD in Section 9.6.

9.1 Asymptotic Communication Complexity

In Table 2, we summarize the asymptotic communication complexity of all proposed schemes and compare them with state-of-the-art protocols that consider malicious users. The dealer has to share information with every user, which leads to a complexity of $O(n)$ for all signature schemes. Users only communicate within their own signing set, for a complexity of $O(k)$, or $O(c)$ in the mPVAS+ scheme. The aggregator needs to relay messages between each user and their signing set, which leads to a complexity of $O(kn)$ for the mPVAS and mPVAS-IV schemes, and $O(cn)$ for the mPVAS+ scheme. Verifiers do not actively participate in the protocol.

Table 2: Asymptotic communication complexity per party in related works and in the mPVAS family.

Protocol	Dealer	Agg.	User	Verifier	Ledger
[29]	$O(n)$	$O(1)$	$O(1)$	-	no
[34]	$O(1)$	$O(1)$	$O(1)$	-	$O(n)$
[37]	$O(n)$	$O(n^2)$	$O(n)$	-	no
mPVAS	$O(n)$	$O(kn)$	$O(k)$	-	no
mPVAS+	$O(n)$	$O(cn)$	$O(c)$	-	no
mPVAS-IV	$O(n)$	$O(kn)$	$O(k)$	-	no
mPVAS-UD	$O(n)$	$O(kn)$	$O(k)$	-	no

The mPVAS family of protocols enjoys reduced communication complexity compared to [37], but increased communication complexity compared to [29, 34]. We note that similar schemes such as [3, 30, 29, 34], work in a different system and adversarial model where there is little to no interaction between the participants except for the initial setup. As such, the communication complexities for these schemes is $O(1)$ for both the aggregator and the users. (Similarly, the computation complexity is $O(1)$ for the users and $O(n)$ for the aggregator.) While this is better than any of the mPVAS variants, the adversarial model in these related works is also weaker than those used in our work. As discussed in Section 3, the compared schemes either assume honest behavior from the users [30, 3], no collusions between the aggregator and the users [34], or they rely on a semi-trusted party during protocol execution [34, 29]. That said, mPVAS can trivially be generalized to these alternative scenarios. For example, honest users can be simulated by choosing $k = 0$, which leads to a non-interactive scheme with $O(1)$ communication complexity and a computation complexity nearly identical to that of the

PUDA scheme [30]. Similarly, choosing $k = 1$ for mPVAS corresponds to the scheme presented in [29], which entrusts a semi-trusted third party with the secret signing key, and similarly leads to constant communication complexity.

9.2 Experimental Setup

We describe the experimental setup with which we empirically determine the runtime complexity of our protocols.

We use the Charm framework [6] to develop a proof-of-concept implementation of mPVAS and its extensions. This framework is widely used for the prototyping and benchmarking of cryptographic schemes, e.g. [1, 38].

The experiments are performed over the MNT224 elliptic curve, which is pairing friendly, provides 112 bits of security [10], and allows for type-3 pairings, which are necessary for the SXDH assumption [3]. This is the most secure curve provided by the Charm framework that is compatible with our schemes. While the current recommendation is to use curves that provide 128 bits of security as a conservative choice, 112 bits is the minimum security level required by NIST for the US Federal Government [4]. In this curve, elements in \mathbb{G}_1 are 56 bytes, in \mathbb{G}_2 are 168 bytes, in \mathbb{G}_T are 168 bytes, and in \mathbb{Z}_p are 28 bytes [25], which we verified experimentally. The size of the elements influences the performance of the various algebraic operations performed in each group.

The experiments were run on a Threadripper 7970X CPU with 256GB of RAM. The protocol was executed sequentially on a single core without special optimizations. Messages are assumed to be delivered instantaneously, so that the measured runtime represents only the computational complexity.

9.3 mPVAS Runtime

We show the runtime of mPVAS (see Section 5) in Figure 2.

Figure 2(a) shows that, even when there are 1000 users and $k = 30\%$ of all users are malicious, the runtime is only around 0.36 seconds for a single user. As expected from an asymptotic complexity $O(k)$, the runtime decreases with the number of malicious users k .

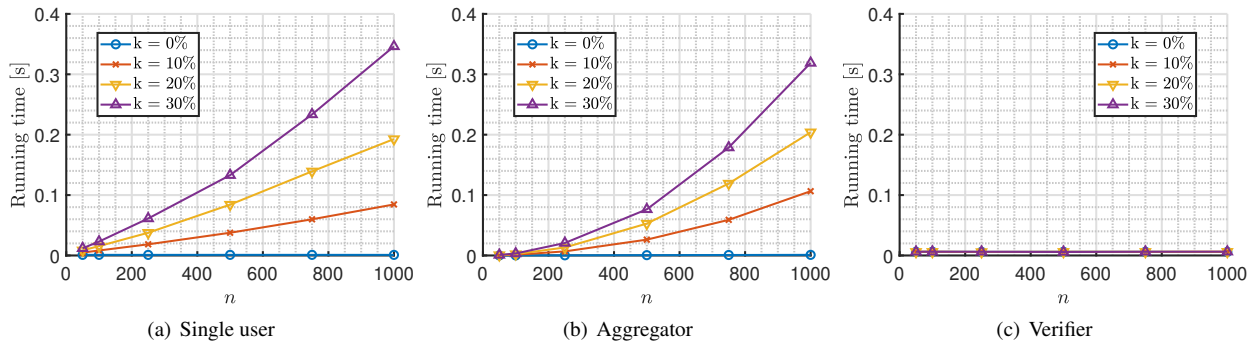


Figure 2: Experimental runtime of mPVAS.

Figure 2(b) shows a similar trend for the aggregator. Moreover, when $k = 0$, the aggregator does not have to combine partial secret sharing for every user, and the runtime dips below even that of a single user.

Finally, Figure 2(c) shows the runtime for verifiers. As expected, since a verifier only needs to compute three pairings, regardless of the number of users, runtime is constant.

9.4 mPVAS+ Runtime

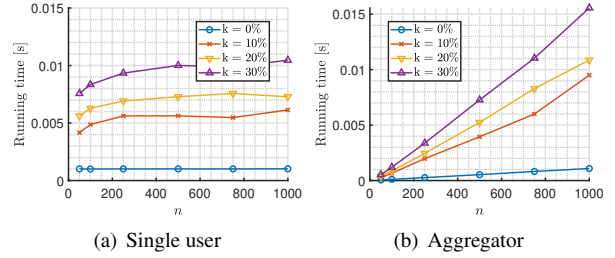


Figure 3: Experimental runtime of mPVAS+.

mPVAS+ (see Section 6) reduces computational complexity under well-defined probabilistic assumptions, assuming non-adaptive corruptions. We present its runtime in Figure 3.

We note that, at $k = 0\%$, mPVAS+ reduces directly to mPVAS, and complexity is independent of the number of users. For experiments with $k > 0\%$, we choose the smallest group size c such that the probability that at least one group is fully compromised is at most 10^{-5} , using our combinatorial formula in (23), giving us group sizes ranging from 5 up to 14.

We see in Figure 3(a) that, compared to mPVAS, user runtime is reduced by an order of magnitude for our choice of parameters. Since the runtime depends only on the constant c , this leads to an upper bound on the runtime. As n continues to grow, the runtime becomes apparently constant.

We see in Figure 3(b) that the speedup for the aggregator is similar. As in regular mPVAS, the main bottleneck for the aggregator is combining the partial user signatures as

in Equation 8. (Combining the final user signatures requires negligible runtime.) Reducing the group size c affects this bottleneck directly. For example, with 1000 users, of which 30% is malicious, setting $c = 14$ means the aggregator must only aggregate $c - 1 = 13$ values per user instead of $k = 300$, reducing complexity in this part by a factor of 23.

9.5 mPVAS-IV Runtime

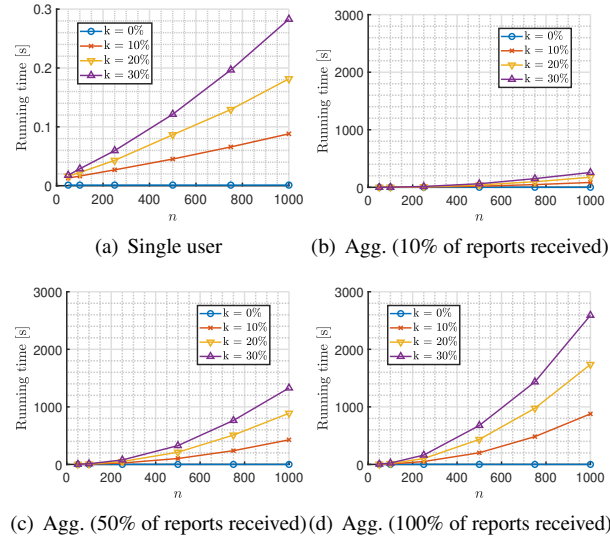


Figure 4: Experimental runtime of mPVAS-IV with final user signature tampering.

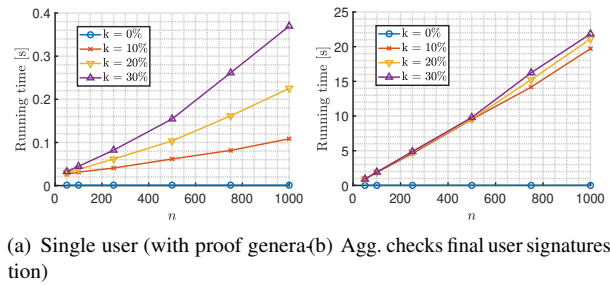


Figure 5: Experimental runtime of mPVAS-IV with aggregate signature tampering.

mPVAS-IV (see Section 6) deals with malicious users who submit malformed signatures to the aggregator. The aggregator identifies these users through several additional checks.

Since the aggregator does not know the number of malicious users beforehand, they will check all signatures in order to find every user that acted maliciously during each round.

First, we consider the case in which users obtain invalid final user signatures during the signing phase and report these to the aggregator. Figure 4(a) shows the runtime for a single

user. We see that the total runtime for a single user is comparable to that of the mPVAS scheme, as there are no additional steps required from users at this stage. For the aggregator, instead, the runtime is dependent on the number of reports received and the size k of each signing set. This dependence is clearly shown in Figure 4(b), Figure 4(c), and Figure 4(d), in which we consider three cases where 10%, 50%, and 100% of users submit a report to the aggregator. In all cases, the runtime is noticeably higher than in the base mPVAS protocol. The reason for this steep increase is the additional exponentiations and pairings required to check whether each signature $\sigma_{j,i}^2$ is well formed. Moreover, these checks must be repeated for every user in a signing set in order to find every possible instance of tampering or whether the report was actually false. We remark that our implementation does not include any specific optimizations, such as parallelization. Since verification is embarrassingly parallel for the aggregator, we expect this can be sped up linearly in the number of cores.

Next, we consider the case in which malicious users submit malformed final user signatures to the aggregator. When this happens, verification of the aggregate signature fails, and the aggregator starts a procedure to identify the malicious users. The checks in this procedure must be performed on all n users, thus giving a linear complexity for the aggregator. Figure 5(a) and Figure 5(b) show the runtime for a single user and for the aggregator, respectively. We see that the runtime for the aggregator can reach up to 22 seconds on our machine. Despite the increased runtime for the aggregator, recall from Section 7 that this identification procedure is necessary only after malicious behavior has occurred. Our experiments represent the cumulative worst-case “denial of service” that malicious users can inflict on the aggregator and other users.

9.6 mPVAS-UD Runtime

In mPVAS-UD (see Section 8) there are changes in the signing phase for users that drop out and in the verification phase for the verifiers. We, therefore, focus on the runtime of those specific parties in Figure 6. In our experiments, we fixed the number of user dropouts to be 10%, 30%, and 50% of the total number of users.

Figure 6(a) shows the runtime for each dropped-out user. We find that the runtime is independent of the number of malicious users and the number of dropped-out users, which is expected since the protocol is non-interactive for these users and the recovery material can be computed in constant time.

From Figure 6(b), Figure 6(c), and Figure 6(d) we see that the runtime for the verifier is not constant anymore, unlike all other variants of mPVAS. In mPVAS-UD, the verifier’s runtime is linear in the number of remaining users because it must compute the product of the masking factors EK_j for every remaining user j , as described in (37). Note that the product of the recovery keys $rk_{i,t}$ is precomputed by the aggregator before being sent to the verifiers to save bandwidth. Still,

these multiplications not particularly expensive and, even in the worst case we consider, with 1000 users and only 10% dropouts, the total running time is below 0.0165 seconds.

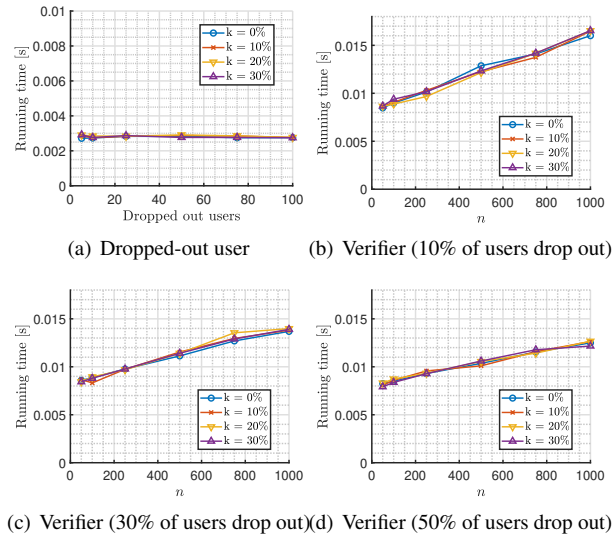


Figure 6: Experimental runtime of mPVAS-UD.

10 Conclusions

mPVAS and its extensions ensure the confidentiality of the input values and the integrity and authenticity of the aggregate even in the presence of a malicious aggregator and a subset of malicious users that collaborate to tamper with the result of the aggregation. Ensuring not only confidentiality but also integrity and authenticity even in the presence of malicious adversaries helps to develop more trust in the results of privacy-preserving schemes and make such schemes appealing to a wider range of scenarios.

Acknowledgments

This work is partly supported by the European Union's Horizon Europe research and innovation program under grant agreement No. 101094901, the SEPTON project (Security Protection Tools for Networked Medical Devices).

References

- [1] Anees Ara, Mznah Al-Rodhaan, Yuan Tian, and Abdullah Al-Dhelaan. A secure privacy-preserving data aggregation scheme based on bilinear elgamal cryptosystem for remote health monitoring systems. *IEEE Access*, 5:12601–12617, 2017. doi:[10.1109/ACCESS.2017.2716439](https://doi.org/10.1109/ACCESS.2017.2716439).
- [2] Giuseppe Ateniese, Jan Camenisch, Susan Hohenberger, and Breno de Medeiros. Practical group signatures without random oracles. *IACR Cryptol. ePrint Arch.*, page 385, 2005. URL <http://eprint.iacr.org/2005/385>.
- [3] Bence Gabor Bakondi, Andreas Peter, Maarten H. Everts, Pieter H. Hartel, and Willem Jonker. Publicly verifiable private aggregation of time-series data. In *10th International Conference on Availability, Reliability and Security, ARES 2015, Toulouse, France, August 24-27, 2015*, pages 50–59. IEEE Computer Society, 2015. doi:[10.1109/ARES.2015.82](https://doi.org/10.1109/ARES.2015.82).
- [4] Elaine Barker and Allen Roginsky. Transitioning the use of cryptographic algorithms and key lengths. Technical Report NIST Special Publication 800-131A Revision 2, National Institute of Standards and Technology, Gaithersburg, MD, 2018.
- [5] James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly)logarithmic overhead. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 1253–1269. ACM, 2020. doi:[10.1145/3372297.3417885](https://doi.org/10.1145/3372297.3417885).
- [6] G. R. Blakley. Safeguarding cryptographic keys. pages 313–318, 1979. doi:[10.1109/MARK.1979.8817296](https://doi.org/10.1109/MARK.1979.8817296).
- [7] Kallista A. 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 Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1175–1191. ACM, 2017. doi:[10.1145/3133956.3133982](https://doi.org/10.1145/3133956.3133982).
- [8] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *J. Cryptol.*, 17(4): 297–319, 2004. doi:[10.1007/s00145-004-0314-9](https://doi.org/10.1007/s00145-004-0314-9).
- [9] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bullet-proofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 315–334. IEEE Computer Society, 2018. doi:[10.1109/SP.2018.00020](https://doi.org/10.1109/SP.2018.00020).
- [10] Hui Cui, Zhiguo Wan, Robert H. Deng, Guilin Wang, and Yingjiu Li. Efficient and expressive keyword

- search over encrypted data in cloud. *IEEE Trans. Dependable Secur. Comput.*, 15(3):409–422, 2018. doi:[10.1109/TDSC.2016.2599883](https://doi.org/10.1109/TDSC.2016.2599883).
- [11] Sanket S. Dhruva, Joseph S. Ross, Joseph G. Akar, Brittany Caldwell, Karla Childers, Wing Chow, Laura Ciacchio, Paul Coplan, Jun Dong, Hayley J. Dykhoff, Stephen Johnston, Todd Kellogg, Cynthia Long, Peter A. Noseworthy, Kurt Roberts, Anindita Saha, Andrew Yoo, and Nilay D. Shah. Aggregating multiple real-world data sources using a patient-centered health-data-sharing platform. *npj Digit. Medicine*, 3, 2020. doi:[10.1038/s41746-020-0265-z](https://doi.org/10.1038/s41746-020-0265-z).
 - [12] Keita Emura, Hayato Kimura, Toshihiro Ohigashi, and Tatsuya Suzuki. Privacy-preserving aggregation of time-series data with public verifiability from simple assumptions and its implementations. volume 62, pages 614–630. 2019. doi:[10.1093/comjnl/bxy135](https://doi.org/10.1093/comjnl/bxy135).
 - [13] Zekeriya Erkin and Gene Tsudik. Private computation of spatial and temporal power consumption with smart meters. In Feng Bao, Pierangela Samarati, and Jianying Zhou, editors, *Applied Cryptography and Network Security - 10th International Conference, ACNS 2012, Singapore, June 26-29, 2012. Proceedings*, volume 7341 of *Lecture Notes in Computer Science*, pages 561–577. Springer, 2012. doi:[10.1007/978-3-642-31284-7_33](https://doi.org/10.1007/978-3-642-31284-7_33).
 - [14] Jingyao Fan, Qinghua Li, and Guohong Cao. Privacy-aware and trustworthy data aggregation in mobile sensing. In *2015 IEEE Conference on Communications and Network Security, CNS 2015, Florence, Italy, September 28-30, 2015*, pages 31–39. IEEE, 2015. doi:[10.1109/CNS.2015.7346807](https://doi.org/10.1109/CNS.2015.7346807).
 - [15] Xi Fang, Satyajayant Misra, Guoliang Xue, and Dejun Yang. Smart grid - the new and improved power grid: A survey. *IEEE Commun. Surv. Tutorials*, 14(4):944–980, 2012. doi:[10.1109/SURV.2011.101911.00087](https://doi.org/10.1109/SURV.2011.101911.00087).
 - [16] 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, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986. doi:[10.1007/3-540-47721-7_12](https://doi.org/10.1007/3-540-47721-7_12).
 - [17] Flavio D. Garcia and Bart Jacobs. Privacy-friendly energy-metering via homomorphic encryption. In Jorge Cuéllar, Gilles Barthe, and Alexander Pretschner, editors, *Security and Trust Management - 6th International Workshop, STM 2010, Athens, Greece, September 23-24, 2010, Revised Selected Papers*, volume 6710 of *Lecture Notes in Computer Science*, pages 226–238. Springer, 2010. doi:[10.1007/978-3-642-22444-7_15](https://doi.org/10.1007/978-3-642-22444-7_15).
 - [18] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2010. doi:[10.1007/978-3-642-14623-7_25](https://doi.org/10.1007/978-3-642-14623-7_25).
 - [19] S. Dov Gordon, Jonathan Katz, Feng-Hao Liu, Elaine Shi, and Hong-Sheng Zhou. Multi-client verifiable computation with stronger security guarantees. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 144–168. Springer, 2015. doi:[10.1007/978-3-662-46497-7_6](https://doi.org/10.1007/978-3-662-46497-7_6).
 - [20] Xiaojie Guo. Fixing issues and achieving maliciously secure verifiable aggregation in "VeriFL: Communication-efficient and fast verifiable aggregation for federated learning". *IACR Cryptol. ePrint Arch.*, page 1073, 2022. URL <https://eprint.iacr.org/2022/1073>.
 - [21] Xiaojie Guo, Zheli Liu, Jin Li, Jiqiang Gao, Boyu Hou, Changyu Dong, and Thar Baker. VeriFL: Communication-efficient and fast verifiable aggregation for federated learning. *IEEE Trans. Inf. Forensics Secur.*, 16:1736–1751, 2021. doi:[10.1109/TIFS.2020.3043139](https://doi.org/10.1109/TIFS.2020.3043139).
 - [22] Changhee Hahn, Hodong Kim, Minjae Kim, and Junbeom Hur. VerSA: Verifiable secure aggregation for cross-device federated learning. *IEEE Trans. Dependable Secur. Comput.*, 20(1):36–52, 2023. doi:[10.1109/TDSC.2021.3126323](https://doi.org/10.1109/TDSC.2021.3126323).
 - [23] Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Goraczko, Allen Miu, Eugene Shih, Hari Balakrishnan, and Samuel Madden. CarTel: A distributed mobile sensor computing system. In Andrew T. Campbell, Philippe Bonnet, and John S. Heidemann, editors, *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, SenSys 2006, Boulder, Colorado, USA, October 31 - November 3, 2006*, pages 125–138. ACM, 2006. doi:[10.1145/1182807.1182821](https://doi.org/10.1145/1182807.1182821).
 - [24] Bonnie Kaplan. How should health data be used? *Cambridge Quarterly of Healthcare Ethics*, 25(2):312–329, 2016. ISSN 1469-2147. doi:[10.1017/S0963180115000614](https://doi.org/10.1017/S0963180115000614).
 - [25] Diptendu Mohan Kar and Indrajit Ray. Systematization of knowledge and implementation: Short identity-based signatures, 2019.

- [26] Ferhat Karakoç, Melek Önen, and Zeki Bilgin. Secure aggregation against malicious users. In Jorge Lobo, Roberto Di Pietro, Omar Chowdhury, and Hongxin Hu, editors, *SACMAT '21: The 26th ACM Symposium on Access Control Models and Technologies, Virtual Event, Spain, June 16-18, 2021*, pages 115–124. ACM, 2021. doi:[10.1145/3450569.3463572](https://doi.org/10.1145/3450569.3463572).
- [27] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014. ISBN 9781466570269.
- [28] Klaus Kursawe, George Danezis, and Markulf Kohlweiss. Privacy-friendly aggregation for the smart-grid. In Simone Fischer-Hübner and Nicholas Hopper, editors, *Privacy Enhancing Technologies - 11th International Symposium, PETS 2011, Waterloo, ON, Canada, July 27-29, 2011. Proceedings*, volume 6794 of *Lecture Notes in Computer Science*, pages 175–191. Springer, 2011. doi:[10.1007/978-3-642-22263-4_10](https://doi.org/10.1007/978-3-642-22263-4_10).
- [29] Iraklis Leontiadis and Ming Li. Secure and collusion-resistant data aggregation from convertible tags. *Int. J. Inf. Sec.*, 20(1):1–20, 2021. doi:[10.1007/s10207-019-00485-4](https://doi.org/10.1007/s10207-019-00485-4).
- [30] Iraklis Leontiadis, Kaoutar Elkhiyaoui, Melek Önen, and Refik Molva. PUDA - privacy and unforgeability for data aggregation. In Michael K. Reiter and David Naccache, editors, *Cryptology and Network Security - 14th International Conference, CANS 2015, Marrakesh, Morocco, December 10-12, 2015, Proceedings*, volume 9476 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2015. doi:[10.1007/978-3-319-26823-1_1](https://doi.org/10.1007/978-3-319-26823-1_1).
- [31] Yongkai Li, Shubo Liu, Jun Wang, and Mengjun Liu. Collusion-tolerable and efficient privacy-preserving time-series data aggregation protocol. *Int. J. Distributed Sens. Networks*, 12(9):1341606, 2016. doi:[10.1177/1550147711341606](https://doi.org/10.1177/1550147711341606).
- [32] Fucui Luo, Haiyan Wang, and Xingfu Yan. Comments on "VERSA: Verifiable secure aggregation for cross-device federated learning". *IEEE Trans. Dependable Secur. Comput.*, 21(1):499–500, 2024. doi:[10.1109/TDSC.2023.3253082](https://doi.org/10.1109/TDSC.2023.3253082).
- [33] Bradley A. Malin, Khaled El Emam, and Christine M. O’Keefe. Biomedical data privacy: problems, perspectives, and recent advances. *J. Am. Medical Informatics Assoc.*, 20(1):2–6, 2013. doi:[10.1136/amiajnl-2012-001509](https://doi.org/10.1136/amiajnl-2012-001509).
- [34] Dimitris Mouris and Nektarios Georgios Tsoutsos. Masquerade: Verifiable multi-party aggregation with secure multiplicative commitments. *IACR Cryptol. ePrint Arch.*, page 1370, 2021. URL <https://eprint.iacr.org/2021/1370>.
- [35] Jianbing Ni, Khalid Nawaf Alharbi, Xiaodong Lin, and Xuemin Shen. Security-enhanced data aggregation against malicious gateways in smart grid. In *2015 IEEE Global Communications Conference, GLOBECOM 2015, San Diego, CA, USA, December 6-10, 2015*, pages 1–6. IEEE, 2015. doi:[10.1109/GLOCOM.2014.7417140](https://doi.org/10.1109/GLOCOM.2014.7417140).
- [36] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In Ernest F. Brickell, editor, *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, volume 740 of *Lecture Notes in Computer Science*, pages 31–53. Springer, 1992. doi:[10.1007/3-540-48071-4_3](https://doi.org/10.1007/3-540-48071-4_3).
- [37] Yanli Ren, Yerong Li, Guorui Feng, and Xinpeng Zhang. Privacy-enhanced and verification-traceable aggregation for federated learning. *IEEE Internet Things J.*, 9(24):24933–24948, 2022. doi:[10.1109/JIOT.2022.3194930](https://doi.org/10.1109/JIOT.2022.3194930).
- [38] Yannis Rouselakis and Brent Waters. Practical constructions and new proof methods for large universe attribute-based encryption. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 463–474. ACM, 2013. doi:[10.1145/2508859.2516672](https://doi.org/10.1145/2508859.2516672).
- [39] Berry Schoenmakers. Lecture notes cryptographic protocols, version 1.9, February 2024. URL <https://www.win.tue.nl/~berry/CryptographicProtocols/LectureNotes.pdf>.
- [40] Elaine Shi, T.-H. Hubert Chan, Eleanor Gilbert Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011*. The Internet Society, 2011.
- [41] Katie Shilton. Four billion little brothers?: privacy, mobile phones, and ubiquitous data collection. *Commun. ACM*, 52(11):48–53, 2009. doi:[10.1145/1592761.1592778](https://doi.org/10.1145/1592761.1592778).
- [42] Nigel P. Smart. *Cryptography Made Simple*. Information Security and Cryptography. Springer, 2016. ISBN 978-3-319-21935-6. doi:[10.1007/978-3-319-21936-3](https://doi.org/10.1007/978-3-319-21936-3).
- [43] Ngoc Hieu Tran, Robert H. Deng, and HweeHwa Pang. Privacy-preserving and verifiable data aggregation. In Aditya Mathur and Abhik Roychoudhury, editors, *Proceedings of the Singapore Cyber-Security Conference (SG-CRC) 2016 - Cyber-Security by Design, Singapore*,

January 14-15, 2016, volume 14 of *Cryptology and Information Security Series*, pages 115–122. IOS Press, 2016. doi:10.3233/978-1-61499-617-0-115.

- [44] Ata Ullah, Muhammad Azeem, Humaira Ashraf, Abdulallah A. Alaboudi, Mamoona Humayun, and N. Z. Jhanjhi. Secure healthcare data aggregation and transmission in iot - A survey. *IEEE Access*, 9:16849–16865, 2021. doi:10.1109/ACCESS.2021.3052850.
- [45] Yong Wang, Aiqing Zhang, Shu Wu, and Shui Yu. VOSA: Verifiable and oblivious secure aggregation for privacy-preserving federated learning. *IEEE Trans. Dependable Secur. Comput.*, 20(5):3601–3616, 2023. doi:10.1109/TDSC.2022.3226508.

A Aggregator Obliviousness of mPVAS

We provide the proof of [Theorem 1](#). We first provide some preliminaries, and then restate the theorem as [Theorem 9](#).

We show that if a probabilistic polynomial-time adversary has a non-negligible advantage of breaking Aggregator Obliviousness of mPVAS, then it also has a non-negligible advantage of breaking Aggregator Obliviousness of the Shi et al. [40] scheme, which is proven under the Decisional Diffie-Hellman assumption. The proof of this property follows an indistinguishability-based game and it provides an adversary with access to the following oracles.

- $O_{\text{Setup}}(1^\lambda)$: Performs the setup of the mPVAS scheme using the given security parameter λ and replies with the public parameters pp and the verification key vk . The secret values of each user $([s]_i, sk_i)$ are kept secret.
- $O_{\text{Compromise}^1}(i \in \mathbb{U})$: When queried on user i , the oracle replies with the secret of user i , namely $([s]_i, sk_i)$.
- $O_{\text{Sign}}(i \in \mathbb{U}, t, x_{i,t})$: Given an input $x_{i,t}$ of user i in round t , the oracle replies with $(\sigma_{i,t}^1, \{\sigma_{i,t}^{2,j}\}_{j \in \mathcal{U}_i}, \sigma_{i,t}^3, \sigma_{i,t})$, where each $\sigma_{i,t}^\ell$, $1 \leq \ell \leq 3$ is a final user signature, and $\sigma_{i,t}$ is the final user signature.
- $O_{\text{Challenge}}(X_{i^*}^0, X_{i^*}^1)$: Given two sets of input values $X_{i^*}^0, X_{i^*}^1$ of size $|X_{i^*}^i| = n$, such that $\sum_{i \in \mathbb{U}^*} x_{i,t}^0 = \sum_{i \in \mathbb{U}^*} x_{i,t}^1$, the oracle randomly flips a coin $b \leftarrow \{0, 1\}$ and, for set $X_{i^*}^b$, it returns all the corresponding partial and final user signatures of its inputs.

Aggregator Obliviousness security game. The Aggregator Obliviousness security game is based on the game introduced by Shi et al. [40].

We can now define Aggregator Oblivious. We follow the definitions seen in [30, 29].

Definition 1 (Aggregator Oblivious). Let $\Pr[\mathcal{A}^{AO}]$ denote the probability that aggregator \mathcal{A} outputs $b^* = b$ in the AO game. A data aggregation protocol is said to be Aggregator Oblivious if, any polynomially bounded \mathcal{A} has negligible advantage $\Pr[\mathcal{A}^{AO}] \leq \frac{1}{2} + \text{negl}(\lambda)$ of winning the AO game.

Theorem 9. The mPVAS scheme is Aggregator Oblivious in the random oracle model under SXDH in \mathbb{G}_1 and \mathbb{G}_2 .

Proof. Let us assume an adversary \mathcal{A} that can win the AO game with a non-negligible advantage. We show how a polynomial time algorithm \mathcal{B} can break the AO scheme of [40], henceforth referred to as PPATS, which is provably secure under the DDH assumption, by using \mathcal{A} as a subroutine. We refer to the following oracles provided by the PPATS scheme. $O_{\text{Setup}}^{\text{PPATS}}$ returns the public parameters. $O_{\text{Encrypt}}^{\text{PPATS}}$ returns the ciphertext $c_{i,t}$ of a given input $x_{i,t}$ in round t using the PPATS scheme. $O_{\text{Compromise}}^{\text{PPATS}}$ returns the secret encryption key sk'_i of a specified user $i \in \mathbb{U}$. Finally, $O_{\text{Challenge}}^{\text{PPATS}}$, only called once during the game, randomly flips a coin $b \leftarrow \{0, 1\}$ and, similarly to the challenge phase described above, encrypts one of the two plaintext sets chosen by the adversary $X_{i^*}^b = \{x_{i,t^*}\}_{i \in \mathbb{U}^*}$.

We follow the AO security game and show how \mathcal{B} reacts to the queries of \mathcal{A} .

1. **Setup.** When \mathcal{A} queries the $O_{\text{Setup}}(1^\lambda)$ oracle, \mathcal{B} queries $O_{\text{Setup}}^{\text{PPATS}}(1^\lambda)$. The latter returns the public parameters $pp_{\text{PPATS}} = (H, \mathbb{G}_1, g_1, p)$. \mathcal{B} also queries the $O_{\text{Compromise}}^{\text{PPATS}}(0)$, which returns the secret key of the aggregator $sk_A = -\sum_{i=1}^n sk'_i$. \mathcal{B} will additionally choose the remaining public parameters of the mPVAS scheme $pp = (H, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, k)$. Finally, \mathcal{B} also chooses the secret keys $(s, \{sk_i\}_{i \in \mathbb{U}}, \{\{ek_{i,j}\}_{1 \leq j \leq k}\}_{i \in \mathbb{U}})$, creates n secret shares $[s]_i$ using $(k+1, n)$ -Shamir Secret Sharing, and creates the verification key as $vk = ((g_2^s)^{-sk_A}, g_2^s) = (g_2^{\sum_{i=1}^n sk'_i}, g_2^s)$. Finally, \mathcal{B} returns pp and vk to \mathcal{A} .

2. **Learning. Compromise.** When \mathcal{A} queries the $O_{\text{Compromise}^1}(i \in \mathbb{U})$ oracle, \mathcal{B} will, in turn, query $O_{\text{Compromise}}^{\text{PPATS}}(i \in \mathbb{U})$ and return the corresponding secret key sk'_i of user i . Additionally, the secret share $[s]_i$ is also sent to \mathcal{A} .

Sign. When \mathcal{A} calls $O_{\text{Sign}}(i \in \mathbb{U}, t, x_{i,t})$, \mathcal{B} queries $O_{\text{Encrypt}}^{\text{PPATS}}(i \in \mathbb{U}, t, x_{i,t})$ to obtain $c_{i,t}^{\text{PPATS}} = H(t)^{sk'_i} g_1^{x_{i,t}}$. \mathcal{B} then computes:

$$\begin{aligned} \sigma_{i,t}^1 &= c_{i,t}^{\text{PPATS}} = H(t)^{sk'_i} g_1^{x_{i,t}}; \\ \sigma_{i,t}^{2,j} &= H_1(t)^{ek_{j,i}} \left(\sigma_{i,t}^1 \right)^{[s]_j^*}; \\ \sigma_{i,t}^3 &= \prod_{j \in \mathcal{U}_i} \sigma_{i,t}^{2,j}; \\ \sigma_{i,t} &= H_1(t)^{ek_{i,i}} \cdot \left(\prod_{j \in \mathcal{U}_i} \sigma_{i,t}^{2,j} \right) \cdot \left(\sigma_{i,t}^0 \right)^{[s]_i^*} = \\ &= H_1(t)^{ek_{i,i} + \sum_{j \in \mathcal{U}_i} ek_{j,i}} \cdot \left(H(t)^{sk'_i} g_1^{x_{i,t}} \right)^s. \end{aligned}$$

Notice how each partial signature and the final user signature $\sigma_{i,t}$ are constructed from the ciphertext output by the encryption algorithm of PPATS but perfectly simulates a partial or final user signature mPVAS scheme. Finally, \mathcal{B} returns $(\sigma_{i,t}^1, \{\sigma_{i,t}^{2,j}\}_{j \in \mathcal{U}_i}, \sigma_{i,t})$ to \mathcal{A} .

Verify. \mathcal{A} can test the correctness of an aggregate sum using the verification key vk obtained during the setup according to (15).

3. **Challenge.** \mathcal{A} chooses a set of uncompromised users $U^* \subseteq \mathbb{U}$, with $|U^*| \geq 2$ and an aggregation round t^* for which no sign queries were made in the learning phase. Then, \mathcal{A} also chooses two sets of ciphertexts $X_{t^*}^0 = \{x_{i,t^*}^0\}_{i \in U^*}$ and $X_{t^*}^1 = \{x_{i,t^*}^1\}_{i \in U^*}$ such that $\sum_{i \in U^*} x_{i,t^*}^0 = \sum_{i \in U^*} x_{i,t^*}^1$. When \mathcal{A} calls the $O_{\text{Challenge}}(X_{t^*}^0, X_{t^*}^1)$ oracle, \mathcal{B} queries $O_{\text{Challenge}}^{\text{PPATS}}(X_{t^*}^0, X_{t^*}^1)$. The oracle flips a coin $b \leftarrow \{0, 1\}$ and returns the encrypted ciphertexts of the b^{th} set $\{c_{i,t^*}^{\text{PPATS}^b}\}_{i \in U^*}$. \mathcal{B} computes the partial and final user signatures using the O_{Sign} oracle, returning $(\{\sigma_{i,t^*}^{1,b}\}_{i \in U^*}, \{\{\sigma_{i,t^*}^{2,b,j}\}_{j \in \mathcal{U}_i}\}_{i \in U^*}, \{\sigma_{i,t^*}^{3,b}\}_{i \in U^*}, \{\sigma_{i,t^*}^{b,b}\}_{i \in U^*})$ to \mathcal{A} . In particular, the final user signature is $\sigma_{i,t^*}^b = H_1(t^*)^{ek_{i,i} + \sum_{j \in \mathcal{U}_i} ek_{j,i}} \cdot (H(t^*)^{sk_i} g_1^{x_{i,t^*}^b})^s$, for $i \in U^*$. Notice how σ_{i,t^*}^b , and all partial signatures are computed from the ciphertexts output by the encryption algorithm of the PPATS scheme and perfectly simulate the ciphertexts, partial and final user signatures of the mPVAS scheme. The aggregation of all such final user signatures is also valid and correctly verified using the verification key vk as $\sigma_{t^*}^b = \prod_{i \in U^*} \sigma_{i,t^*}^b = (H(t^*)^s)^{\sum_{i \in U^*} sk_i} (g_1^s)^{\sum_{i \in U^*} x_{i,t^*}^b}$.

If \mathcal{A} has a non-negligible advantage ϵ of guessing the correct bit b^* in the AO game of the mPVAS scheme, then \mathcal{B} can also win the AO game of the PPATS scheme with the same non-negligible advantage ϵ by guessing the same bit b^* . This would contradict the DDH assumption in \mathbb{G}_1 , because the security of the PPATS scheme relies on this assumption. Additionally, if the DDH does not hold in \mathbb{G}_1 , then the SXDH assumption does not hold either since it requires that the DDH problem be hard in \mathbb{G}_1 . Therefore, the mPVAS scheme is Aggregator Oblivious in the random oracle model under the SXDH assumption. \square

B Aggregate Unforgeability of mPVAS

We provide the proof of [Theorem 2](#).

Proof. Type-I Unforgeability. A Type-I forgery [3, 30, 29, 12, 43] occurs when the aggregator outputs a valid aggregate

signature σ_t in a round t without receiving any users' signatures. Thus, the aggregator can only use knowledge from previous rounds or by colluding with users. First, we note how signatures from different rounds are incompatible with each other. Assuming a cryptographically-secure hash function $H: \{0, 1\}^* \rightarrow \mathbb{G}_1$ under the random oracle model its output can be considered random. As such, in each round t , each signature has a different random factor $H(t)$. Even assuming the aggregator chooses the round identifier t , because of the collision resistance property of H , it has a negligible probability of finding two different round identifiers t, t' such that $H(t) = H(t')$. Similarly, because of the second pre-image resistance property of H , given t , the aggregator has negligible probability of finding another t' such that $H(t) = H(t')$. The same arguments hold for the other hash function $H_1: \{0, 1\}^* \rightarrow \mathbb{G}_1$ used in the protocol. It follows that the aggregator cannot reuse signatures from previous rounds. The only other option left for the aggregator is to construct new signatures itself. However, in order to do so, all secret signing keys sk_i are required but, assuming colluding users, the aggregator has only access to at most k of them. The aggregator also needs the secret exponent s to compute a valid signature, but it has only access to at most k secret shares of s , which are not enough to reconstruct s .

Type-II Unforgeability. [3, 30, 29, 12, 43] There are two pieces of information that can allow the aggregator to successfully forge an aggregate signature in a round in which it received all signatures from the users: the secret exponent s or the factor g_1^s . The exponent s is secret-shared by all users using $(k+1, n)$ -Shamir Secret Sharing. Since we assume at most k malicious users who collude with each other and k shares leak no information about the underlying secret s , then no dishonest party can directly learn s from exchanging their shares. Additionally, if $g^s \in \mathbb{G}_1$ is known, for any $g \in \mathbb{G}_1$, recovering s is considered computationally infeasible because DLP is assumed to be hard in \mathbb{G}_1 . The same argument applies to the value $g_2^s \in \mathbb{G}_2$, which is part of the verification key and known by every verifier. Note that, while a malicious actor may know g_2^s , since the Co-Computational Diffie-Hellman (Co-CDH) [8] problem is assumed to be hard in \mathbb{G}_1 and \mathbb{G}_2 , obtaining g_1^s is still considered hard.

In a malicious setting where users can behave arbitrarily, sending malformed signatures may allow them to gain additional information that will allow them to break the Aggregate Unforgeability property.

Each signature starts with the form $\sigma_{i,t}^1 = H(t)^{sk_i} g_1^{x_{i,t}}$. Clearly, any malicious party can immediately tamper with this signature since g_1 is public. However, the aggregator is required to send a *Compute final user signature* request to every user in order for the aggregate signature to be successfully verified. From (10), any $\sigma_{i,t}^3$ that was not computed using user i 's original $\sigma_{i,t}^1$ will lead to an invalid $\sigma_{i,t}$, because the bases of the two factors will not match. This, in turn, will lead to an invalid aggregate signature σ_t .

In order to prevent malicious actors from manipulating any of the partial signatures to obtain g_1^s , each user j is required to further mask any response to a *Create partial signature* or *Compute final user signature* request with a fresh masking factor $H_1(t)^{ek_{j,i}}$. Assuming at most k malicious users, any signing request from each user i will result in a $\sigma_{i,t}$ containing at least one such factor, since either user i itself or a user in its signing set must be honest by assumption. Therefore, all that malicious actors can learn by deviating from the protocol is of the form $H_1(t)^\epsilon g^s$ or $H_1(t)^\epsilon g^{[s]_j^*}$, where $g \in \mathbb{G}_1$ and ϵ indicates the sum of any non-empty subset of masking exponents $ek_{j,i}$. None of these values can be used to successfully forge a valid aggregate signature, as they would introduce extra masking exponents that would not sum up to 0 anymore. As a result, the verification algorithm will fail. \square

C Aggregate Unforgeability of mPVAS-IV

We provide the proof of [Theorem 6](#).

Proof. In the mPVAS-IV extension, the aggregator is trusted to detect users who may attempt disrupt the normal execution of the protocol and not to disrupt the protocol itself. However, the aggregator is still considered malicious with respect to the unforgeability property of the mPVAS-IV extension. This extension provides the aggregator with additional knowledge that is not available in the main scheme. As such, in this section, we provide additional arguments to show why Aggregate Unforgeability is still maintained in the mPVAS-IV extension.

Type-I Forgeries. The new pieces of information that the aggregator is handed in the mPVAS-IV extension are the sets $SS_i = g_2^{[s]_i}$, $SK_i = g_T^{sk_i} h_T^{r_i}$, $EK_{j,i} = g_2^{ek_{j,i}}$, $EKS_{j,i} = g_2^{\frac{ek_{j,i}}{s}}$, and the value $v = g_2^{\frac{1}{s}}$. Since DLP is assumed to be intractable in \mathbb{G}_2 , the aggregator has a negligible probability of obtaining the secret share $[s]_i$ or the masks $ek_{j,i}$ of a user i from $g_2^{[s]_i}$ and $g_2^{ek_{j,i}}$, respectively. Similarly, finding $\frac{1}{s}$ from v is also hard. Additionally, because of the perfect hiding property of Pedersen commitments, the aggregator cannot learn any information about the signing key sk_i from its corresponding commitment in SK_i . Hence, the additional information that is handed to the aggregator in the mPVAS-IV extension gives the aggregator no advantage of learning the necessary information to create Type-I forgeries.

Type-II Forgeries. There is no additional piece of information handed to the aggregator in the mPVAS-IV extension that could allow it to create Type-II forgeries. Intuitively, this is because all of the additional values are members of either \mathbb{G}_2 or \mathbb{G}_T , but the signatures are elements of \mathbb{G}_1 . As such, there is no additional information that could be used by the aggregator to tamper with the signatures in \mathbb{G}_1 , assuming the SXDH assumption holds in the chosen pairing group. \square

D Aggregator Obliviousness of mPVAS-UD

We provide the proof of [Theorem 7](#).

Proof. In the mPVAS-UD protocol, the signing phase is identical to that of the main mPVAS protocol for all remaining users. As such, the signature of each remaining user perfectly hides the input value, as proven in [Theorem 1](#).

The signing phase is, however, different for dropped-out users. Each user is required to send a recovery key $rk_{i,t}$ for every round t it wish to drop out of. This value could, in turn, be plugged in (24), using $EK_{j,i}$, to find user i 's secret input value by brute force. Fortunately, the signing key sk_i is bound to the generator $H(t)$ and, thus, can only be successfully used in round t , during which dropped-out users do not submit any data. Additionally, if malicious users collaborate to create k additional recovery factors $rk_{i,t}$, then, as long as at least 2 honest users do not drop out during round t , only the sum of their input data can be computed, but not the individual values.

As a result, the input data of both remaining and dropped-out users remain private during every step of the protocol. \square

E Aggregate Unforgeability of mPVAS-UD

We provide the proof of [Theorem 8](#).

Proof. The main addition introduced by the mPVAS-UD extension is the recovery key $rk_{i,t}$ that users that wish to exit the protocol during some round t submit to the aggregator. The recovery key is computed over \mathbb{G}_T , so it cannot be used to directly affect the signatures, which are elements of \mathbb{G}_1 .

The aggregator cannot lie about the set of users that drop out during any given round t and cannot publish more than one valid aggregate signature. Forcing a subset of users out of the protocol would lead to a failed verification, since the aggregator cannot provide the verifiers with valid recovery keys for the missing users on its own. Similarly, the aggregator cannot force a dropped-out user i in the protocol as it does not possess its signing key sk_i . Assuming a subset of users colludes with the aggregator and provides it with valid recovery keys, we identify two cases.

If any of these users actually engage in the protocol, then their recovery key alone would not suffice anymore, because their final user signature would contain at least one masking factor $H_1(t)^{ek_{j,i}}$, with $ek_{j,i}$ belonging to an honest user j , which has not been redistributed among the remaining users. As such, the check in (37) would fail.

If they do not submit anything, then the aggregator is forced to forward their recovery keys, indicating they have indeed dropped out, otherwise, the verification would fail.

As for Type-I forgeries, the same arguments presented in [Theorem 2](#) apply to mPVAS-UD. \square