Collusion Resistant Aggregation from Convertible Tags

Iraklis Leontiadis, Ming Li

University of Arizona, USA
{leontiad,lim}@email.arizona.com

Abstract. The progress in communication and hardware technology increases the computational capabilities of personal devices. Aggregators, acting as third parties, are interested in learning a statistical function as the sum over a census of data. Users are reluctant to reveal their information in cleartext, since it is treated as personal sensitive information. The paradoxical paradigm of preserving the privacy of individual data while granting an untrusted third party to learn in cleartext a function thereof, is partially addressed by the current privacy preserving aggregation protocols.
Current solutions are either focused on a honest-but-curious Aggregator who is trusted to follow the rules of the protocol or they model a malicious Aggregator with trustworthy users. In this paper we are the first to propose a protocol with fully malicious users who collude with a malicious Aggregator in order to forge a message of a trusted user. We introduce the new cryptographic primitive of *convertible tag*, that consists of a two-layer authentication tag. Users first tag their data with their secret key and then an untrusted *Converter* converts the first layer tags in a second layer. The final tags allow the Aggregator to produce a proof for the correctness of a computation over users' data. Security and privacy of the scheme is preserved against the *Converter* and the Aggregator, under the notions of *Aggregator obliviousness* and *Aggregate unforgeability* security definitions, augmented with malicious users. Our protocol is provably secure and experimental evaluations demonstrates its practicality.

Keywords: data privacy, data security, convertible tags, collusion resistant aggregation

1 Introduction

The folklore model of Alice and Bob who want to exchange messages in a secure way, has been extensively analyzed. Nowadays, with the progress of communication and computing technology, users are able to produce big amount of data, which is shared with untrusted parties. As such, the idea of locally holding the data is of the past. Users leverage the computational and storage capabilities in order to store and analyze their data. Solutions tailored for this scenario propose a new model for outsourced data computations. In the paper we are focused on secure aggregation. In a nutshell, in an aggregation protocol, untrusted parties collect individual users' data in order to compute a function over their cleartext data. The paradigm of data collection and analysis is motivated by a plethora or real world scenarios:

- Smart metering data is collected by an energy supplier in order to perform energy forecasting for cost minimizations. On the other hand users want to protect their individual privacy and apply a privacy preserving mechanism on their data.
- In a healthcare scenario patients leave their health traces to hospitals. These traces comprise health care sensitive data and a compromise thereof, affect negatively the patients: A hospital which acts as a data enclave for patients data may collude with an insurance company. The latter may decline an insurance subscription to a patience according its health care data.

In the aforementioned use cases an untrusted Aggregator computes in cleartext a function f over users' data and forwards the result to a Data Analyzer. The paradox stems from the desire of each user to protect its individual privacy while the Aggregator wants to learn in cleartext f over users' data. Existing literature is focused either on protecting individual privacy [10, 22, 25, 31] or on improving the security model with a malicious Aggregator; who will try to convince a Data Analyzer, who acts as a honest verifier, that the result of computations comes from genuine data inputs. In [26] the authors by leveraging the encryption scheme of Shi *et al.* [31] they enrich the typical data collection and analysis protocol with a proof computed by a malicious Aggregator, which allows the Data Analyzer to verify the correctness of computations. However the authors employ a rather weak model. During their analysis, users are

assumed as trustworthy and they do not collude with the Aggregator. However, this assumption is not realistic in a real world scenario in which trustworthiness is not guaranteed. Namely, users can collude with the Aggregator in order to change the protocol's messages at their need. This would have devastating consequences on users' security. In [16] the authors propose a solution in which trustworthiness of users is correlated with the validity of their produced data. Their solutions incorporates a blind commitment before the collection of the data. In between the commitment and the aggregation phase users cannot change their data. However a malicious user is able to alter its real data before the commitment phase, thus violating the validity of data.

We propose a secure aggregation protocol in the presence of untrustworthy users. In this setting users are allowed to collude with a malicious Aggregator, without affecting the security of the scheme. We only require that users send correct data and not fake information. The striking attribute of our protocol which is of independent importance is a new cryptographic primitive named *convertible tag*. Users tag their data with a convertible tag using independent randomness. This allows users to collude with a malicious Aggregator without the latter being able to forge user's data. The tag is convertible, in the sense that a semi-trusted third party with some auxiliary information computed by each user, can convert it to a another tag, which is able to be aggregated with respect to the function f. Informally, the security guarantees for *convertible tags* assure that any collusion of the user with a malicious Aggregator cannot forge non-genuine data, originating from other users. Plugging convertible tags to a secure aggregation protocol also assures unforgebility of data aggregation as formalized in [26] and Aggregator obliviousness [31]. That is, a malicious Aggregator cannot convince a honest verifier for the correctness of computation f that arises from non-genuine data inputs and moreover individual privacy of users' inputs is preserved thanks to the obliviousness property. We summarize the contributions of this paper as follows.

Contributions

- In the aim of assuring collusion resistant aggregation we come up with the cryptographic primitive of *convertible tag*. Users can choose independently their tag keys. The tags are unified under common randomness with the aid of a semi-honest third party, called hereafter the *Converter*. The convertible tags assure *obliviousness* of computations against a malicious Aggregator and a semi-honest *Converter*, without jeopardizing unforgeability.
- We extend the current security definitions of secure aggregation protocols with collusions: a) between users and Aggregator, b) between users and the *Converter*, c) between the Aggregator and the *Converter*, in case of trustworthy users. Our protocol is provably secure under standard assumptions in the random oracle model.
- Thanks to our construction, the protocol achieves constant time symmetric verification in a multi-user setting.

Outline In section 2 we introduce the problem this paper addresses and we identify the lack of a stronger security definition from existing protocols. Afterwards, in section 3 we review similar cryptographic primitives with *convertible tags*. We continue in section 4 with the core idea of our solution and in section 5 with the technical preliminaries . The protocol is presented in full details in section 6 and its cost analysis in section 7. We analyze the security of the scheme in section 8. Finally, we conclude in section 9.

2 Problem Statement

For a secure aggregation protocol, we assume a set of n users $\mathbb{U} = \{\mathcal{U}_i\}_{i=1}^{n}$, each one producing time series personal data inputs $x_{i,t}$. Users encrypt their data with an encryption algorithm, which produces ciphertexts $c_{i,t}$. Ciphertexts are collected by an Aggregator $\mathcal{A}$, whose main goal is to learn a function f in cleartext over users' data and forward the result to a trustworthy Data Analyzer $\mathcal{DA}$, who does not communicate with each user. We assume a malicious Aggregator who does not follow the rules of the protocol and seeks to infer more information from the exchanged messages of the protocol. More specifically the Aggregator will try to convince a honest verifier $\mathcal{DA}$ for the correctness of computations over non-genuine data. To protect against the malicious Aggregator users further tag their data in such a way that a proof of correct computations can be constructed by the Aggregator and will convince the verifier.

We recall in this section the syntax of a secure aggregate protocol as described in [26].

2.1 Syntax

- **Setup**$(1^\lambda) \rightarrow (\mathsf{pp}, \mathsf{sk_A}, \{\mathsf{sk}_i\}_{\mathcal{U}_i,\in\mathbb{U}}, \mathsf{vk})$: It is a randomized algorithm run by a trusted dealer $\mathcal{KD}$, which on input of a security parameter λ outputs the public parameters pp that will be used by subsequent algorithms, the Aggregator $\mathcal{A}$'s secret key $\mathsf{sk_A}$, the secret keys $\mathsf{sk_i}$ of users $\mathcal{U}_i$ and the public verification key vk.
- **EncTag**$(t, \mathsf{sk}_i, x_{i,t}) \rightarrow (c_{i,t}, \sigma_{i,t})$: It is a randomized algorithm which on inputs of time interval t, secret key sk_i of user $\mathcal{U}_i$ and data $x_{i,t}$, encrypts $x_{i,t}$ to get a ciphertext $c_{i,t}$ and computes a tag $\sigma_{i,t}$ that authenticates $x_{i,t}$.
- **Aggregate**$(\mathsf{sk_A}, \{c_{i,t}\}_{\mathcal{U}_i\in\mathbb{U}}, \{\sigma_{i,t}\}_{\mathcal{U}_i\in\mathbb{U}}) \rightarrow (\mathsf{sum_t}, \sigma_\mathsf{t})$: It is a deterministic algorithm run by the Aggregator $\mathcal{A}$. It takes as inputs Aggregator $\mathcal{A}$'s secret key $\mathsf{sk_A}$, ciphertexts $\{c_{i,t}\}_{\mathcal{U}_i\in\mathbb{U}}$ and authentication tags $\{\sigma_{i,t}\}_{\mathcal{U}_i\in\mathbb{U}}$, and outputs in cleartext the sum $\mathsf{sum_t}$ of the values $\{x_{i,t}\}_{\mathcal{U}_i\in\mathbb{U}}$. Moreover, it computes a proof σ_t assessing the correctness of $\mathsf{sum_t}$, using the authentication tags $\{\sigma_{i,t}\}_{\mathcal{U}_i\in\mathbb{U}}$.
- **Verify**$(\mathsf{vk}, \mathsf{sum_t}, \sigma_\mathsf{t}) \rightarrow \{0, 1\}$: It is a deterministic algorithm that is executed by the Data Analyzer $\mathcal{DA}$. It outputs 1 if Data Analyzer $\mathcal{DA}$ is convinced that the sum $\mathsf{sum_t} = \sum_{\mathcal{U}_i\in\mathbb{U}}\{x_{i,t}\}$; and 0 otherwise, with the aid of the verification key vk.

2.2 Security Model

We build upon the model as presented in [26] and we further assume that users are not trustworthy. We only require that each user be it trustworthy or not submits real data and not fake inputs. Notably, users can collude with the Aggregator in order to forge non-genuine tags for a legitimate user. This has a negative result on the scheme's security, since the security definition of *aggregate unforgeability* is not assured anymore. In a nutshell, *aggregate unforgeability* definition follows the classical message tag unforgeability under chosen message attack, with the difference that adversary $\mathcal{A}$ cannot forge an aggregate tag with respect to the computation f. That is, if users submit tags $\sigma_{i,t}$ for their private data inputs $x_{i,t}$ then $\mathcal{A}$ can only compute a valid aggregate tag σ_t for the sum computation over $x_{i,t}$ and nothing else. We show how the scheme in [26] does not assure *aggregate unforgeability* in the presence of non-legitimate users, who collude with a malicious Aggregator $\mathcal{A}$. A malicious user $\mathcal{U}_m$ shares the secret information a with the Aggregator. The Aggregator $\mathcal{A}$ then can forge another user's tag with a as follows: After obtaining $\sigma_{l,t} = H(t)^{\mathsf{tk}_l}(g_1^a)^{x_{l,t}}$ from a legitimate user $\mathcal{U}_l$ at time interval t, $\mathcal{A}$ computes $\sigma_{l,t}\cdot(g_1^a)^v = H(t)^{\mathsf{tk}_l}(g_1^a)^{x_{l,t}+v}$, for a value v of its choice. Thus, $\mathcal{A}$ can produce a valid proof by aggregating all tags and the forged one, for a sum that comes from non-genuine data. We also inherit the privacy definitions of Aggregator obliviousness, which protects individual privacy. A malicious Aggregator from the computation of the sum in cleartext over individual data inputs cannot jeopardize individual privacy. The privacy definition is expressed as a security game .

3 Related work

Similar cryptographic primitives have been proposed in the literature for the purpose of unforgeability with privacy. Blind signatures provide privacy by allowing the signer to sign a message blindly, without learning what it signs [11]. Group signatures [12] provide anonymity by allowing any member of an authorized group to sign on behalf of the group manager. Group signatures provide traceability and non-frameability. The traceability property requires that no adversary can compute a signature that cannot be traced to a user and non-frameability assures that a malicious group manager cannot falsely accuse a user. With proxy signatures [5, 29] and its variations (anonymous [17], private [14, 19, 20]), signing rights are delegated to a proxy who signs on behalf of a user. Proxy Re-Signatures (PRS) [2, 4, 28] translate signature for one party to another one. PRS share some properties with convertible tags. We carefully compare our new primitive with the aforementioned constructions below.

Blind signatures Chaum first introduced the notion of *blind signatures*. A user sends a blinded version of its message to the signer and the latter signs without learning the underlying message. The user then obtains the signature on the original message and sends the signature to the verifier. Apart from confidentiality, blind signatures guarantee also anonymity and they are useful for a broad range of applications, as e-cash [7] and anonymous credentials [8]. Similarly with the *convertible tags* blind signatures offer privacy on top of authentication but only for the third party who signs and not for the verifier. The verifier in a blind signature verifies the correctness of a message in cleartext. In contrast, *convertible tags*

extend this functionality with privacy, since there is not one-to-one message signature verification but verification of the correctness of an aggregate result over data. Moreover, in a multi-user setting, *convertible tags*, offer increased security, other than unlinkability, in case of collusions between a user and the signer. The user can verify the well-formness of the tag. In cotrast blind signatures assume the signer as trusted to sign correctly.

Group signatures Group signatures [6, 9] allow a member of a group to sign on behalf of a group manager in such a way that anonymity of the sender is preserved. Moreover they guarantee traceability of the signatures, non-frameability and coalition resistance. The model of *convertible tags* differ from group signatures in the sense that groups signatures do not offer confidentiality over the entire group messages and moreover they do not support homomoprphic operations on the signatures.

Proxy signatures In proxy signatures the signer delegates its signing rights to an authorized proxy. The proxy can sign on behalf of the designator and the receiver of the signatures can verify the authenticity of the signature as originated from the designator. In practice the secret key of the original signer is split between the receiver and the proxy. Variations of proxy signatures as warrant-signatures [14, 19] restrict the proxy to sign only specific parts of the messages without being able to learn the space of the allowed messages that it can sign. *Convertible* tags enable a multi-user setting, in which multiple tags from different users are converted in a single tag with common randomness.

Proxy re-signatures The primitive of proxy re-signatures allows a designator to delegate a transformation operation on its signature with the aid of proxy in order the latter to transform the original signature signed with the signature key of a different user. The proxy re-signatures primitive bears similarities with the convertible tags primitive since in both there is a transformation mechanism by a third party, who converts the authentication tags. However convertible tags operate in a different model: multiple users tag their data such that the third party cannot learn the authenticated data. As such, confidentiality is being preserved in contrast with proxy-re signatures in which there is only authenticity guarantee. Another major issue with proxy re-signatures is that they are not homomorphic, while convertible are constructed not for a per message verification but for computation verification.

Conceptually, *convertible tags* can be viewed as a combination of blind signatures, group signatures, and proxy (re-) signatures. They employ the privacy guarantee of confidentiality of blind signatures, the communication model of groups signatures and the transformation property of a signature from one user to another as with proxy (re-) signatures. However a simple assembly of the aforementioned primitives for the construction of a *convertible tag* is not a trivial plug in of all those primitives, simply because in case of collusions the security guarantees of each are not preserved. Notice that unforgeable signatures on the tag solves the problem, but that would incur extra computational complexity to the Aggregator for verifying each signature, and the different public keys for all users burden its storage complexity.

4 Idea and Model

4.1 Idea

The core idea of our solution for collusion resistant aggregation is a symmetric authentication mechanism at the target group of bilinear pairings. Each user chooses uniformly at random tag keys for the authentication tag, which at a first level, is named metatag. Users send their metatags to a semi-honest party, the *Converter* $\mathcal{C}$ and their ciphertexts to the malicious Aggregator $\mathcal{A}$. The protocol at this point assures *unforgeability* and *obliviousness* against the *Converter* $\mathcal{C}$. Along with the metatags each user transmits to $\mathcal{C}$ some auxiliary information coupled with a blinded version of their secret tag key. $\mathcal{C}$ then couples all this information and ends up with the final tag of each user at the second level. The coupling annihilates the randomness per user and transforms the metatags to the final *convertible tag*, that is forwarded to the malicious Aggregator $\mathcal{A}$. Users upon receiving their tags from $\mathcal{C}$ validate its correctness. This is happening in order to ensure that in case of a collusion between a colluding user and the *Converter* $\mathcal{C}$, the latter cannot forward a forged tag, with the key that is used by $\mathcal{C}$ to couple the metatag and the auxiliary information. That is, a malicious user cannot extract the randomness used for the final computation of the authentication tag in case of collusion with the malicious $\mathcal{A}$, in order to forge an authentication tag for another user. Aggregator receives all tags and ciphertexts. $\mathcal{A}$ then decrypts and learns the result $\mathsf{sum}_\mathsf{t} = f = \sum_{i=1}^{n} x_{i,t}$ and computes a proof of correct computations σ_t based on the convertible tags.

Finally, $\mathcal{A}$ forwards to the data analyzer $\mathcal{DA}$ the result sum_t and the proof σ_t. $\mathcal{DA}$ verifies the correctness of computations as a honest verifier in constant time. The *convertible tags* assure *Aggregator*

obliviousness and *aggregate unforgeability*. In a nutshell with *Aggregator obliviousness* $\mathcal{A}$ cannot learn anything more than the aggregate result $\sum_{i=1}^{n} x_{i,t}$. *Aggregate unforgeability* guarantees the correct computation of $\mathsf{sum}_t = f = \sum_{i=1}^{n} x_{i,t}$. Both security guarantees are enriched, in contrast with previous work [26], with collusions between malicious users, Aggregator $\mathcal{A}$ and *Converter* $\mathcal{C}$. Thus, our solution assures :

1. Collusion resistance between a malicious user and a malicious Aggregator $\mathcal{A}$, thanks to the individual keys chosen by each user. Despite the convertible tags that in the end cancel out all the individual keys and use a common randomness in order $\mathcal{A}$ to compute a proof of correctness based on the sum computation, individual randomness chosen by each user permits collusions between a user and an Aggregator without the latter being able to forge a tag of a legitimate user.
2. Collusion resistance between a malicious user and a semi-honest *Converter* $\mathcal{C}$, thanks to the *convertible tag* that is verified by each user after receiving their tags by the *Converter*. *Convertible tags* allow each user to verify whether or not $\mathcal{C}$ tried to forge a convertible tag after colluding with another user.
3. Collusion resistance between a malicious Aggregator $\mathcal{A}$ and a semi-honest *Converter* $\mathcal{C}$. In the case of users who do not act adversarially, meaning they have not been captured by an external adversary, who shares secret information with a malicious Aggregator or *Converter*, our protocol is resilient to collusions between a malicious Aggregator $\mathcal{A}$ and a semi-honest *Converter* $\mathcal{C}$.

As we extend the model for privacy preserving and unforgeable aggregation as presented in [26] and in section 2.1, with malicious users and extra parties (*Converter*) in the protocol, we also change the model of the scheme syntactically and we describe it as follows.

4.2 Collusion Resistant Aggregation Model

- **Setup**(1^λ) : This is a probabilistic algorithm that on input the security parameter λ it outputs the public parameters pp and the secret key sk_A of the Aggregator.
- **UKeygen**$(1^\lambda)\langle\mathcal{KD}, \mathbb{U}\rangle$: The key dealer $\mathcal{KD}$ runs this algorithm in order to distribute secret keys to each user for encryption. Moreover users choose uniformly at random their tag keys.
- **CKeygen**$(1^\lambda)\langle\mathcal{KD}, \mathbb{U}, \mathcal{C}, \mathcal{DA}\rangle$: This key distribution algorithm runs between the users who blind their randomness from the **UKeygen**$(1^\lambda)\langle\mathcal{KD}, \mathbb{U}\rangle$ algorithm, send that to the Converter $\mathcal{C}$, and the latter distributes the secret authentication tag key to the data analyzer $\mathcal{DA}$.
- **EncTag**$(\mathsf{pp}, \mathsf{sk}_i, x_{i,t})$: Each user using its secret encryption key encrypts its individual data and sends the ciphertext $c_{i,t}$ to $\mathcal{A}$. Moreover using its secret tag key computes a metatag $\mathsf{mtag}_{i,t}$ and sends that to the *Converter* $\mathcal{C}$.
- **Convert**$(\mathsf{pp}, r, \mathsf{mtag}_{i,t})$: $\mathcal{C}$ with the key r, and the metatag $\mathsf{mtag}_{i,t}$ computes the final tag $\sigma_{i,t}$ for user $\mathcal{U}_i$.
- **VTag**$(\mathsf{pp}, \mathsf{sk}_i, \sigma_{i,t}, x_{i,t})$: Each user verifies the correctness of the final tag $\sigma_{i,t}$. *Convertible tags* prevent $\mathcal{C}$ to forge a user's tag using secret information from a colluding user.
- **Aggregate**$(\mathsf{sk}_A, \{c_{i,t}\}, \{\sigma_{i,t}\})$: Aggregator $\mathcal{A}$ upon collecting the ciphertexts $\{c_{i,t}\}$ and the tags $\{\sigma_{i,t}\}$ decrypts with the secret key sk_A and learns the result $\mathsf{sum}_t = \sum_{i=1}^{n} x_{i,t}$. Moreover, it computes a proof of correct computation σ_t and finally and forwards to the data analyzer $\mathcal{DA}$ $\mathsf{sum}_{t}t, \sigma_t$.
- **Verify**$(\mathsf{pp}, \mathsf{vk}, \mathsf{sum}_t, \sigma_t)$: The data analyzer $\mathcal{DA}$ verifies the correctness of computation for the $\mathsf{sum}_{t}t$, using the proof σ_t and the secret verification key vk and the public parameters pp.

4.3 Security and Privacy Model

In this section we analyze the collusions resiliency property for aggregation protocols. We further formally define the security and the privacy properties.

Collusions and Trust model In contrast with previous model and solution [26], our scheme fulfills its security guarantees under weakened assumptions. More specifically, collusions in between users and malicious parties are supported without jeopardizing security definitions for unforgeability. Users $\mathbb{U} = \{\mathcal{U}\}_{i=1}^{n}$ in the scheme are unauthenticated and can act maliciously. Collusions can happen between a malicious user $\mathcal{U}_m$ and a colluding Aggregator $\mathcal{A}$ or a malicious *Converter* $\mathcal{C}$. Users share any secret

User	$\mathsf{Coll}_{\mathcal{A},\mathcal{C}}$	$\mathsf{Coll}_{\mathcal{A},\mathcal{U}_m}$	$\mathsf{Coll}_{\mathcal{C},\mathcal{U}_m}$
Trustworthy	✓	✓	✓
Malicious	✗	✓	✓

Table 1: Collusion model for Aggregate unforgeability.

information they know with the colluding members with the goal to forge other users' tag. Users are only trusted to submit correct data be it malicious or untrustworthy. Collusions between $\mathcal{C}$ and $\mathcal{A}$ are also possible in case of trustworthy users (cf. table 1). We also assume the data analyzer $\mathcal{DA}$ to be a trustworthy party, which does communicates with the users. We thus, omit it from the security model. We first describe the oracles an adversary $\mathcal{A}$ has access to when collusions between $\mathcal{U}, \mathcal{C}$ and $\mathcal{A}$ are possible:

- $\mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{U}_m}}(\mathsf{uid} = i \in \mathbb{U})$: On input a user identifier uid this oracle transmits to an adversary who impersonates a malicious Aggregator $\mathcal{A}$ the user's secret information $(\mathsf{ek}_{\mathsf{uid}}, \mathsf{tk}_{\mathsf{uid}}, \mathsf{r}_{\mathsf{uid}}, w)$ after running the $\mathbf{UKeygen}(1^\lambda)$ and $\mathbf{CKeygen}(1^\lambda)$ algorithms.
- $\mathcal{O}^{\mathsf{Coll}_{\mathcal{C},\mathcal{U}_m}}(\mathsf{uid} = i \in \mathbb{U})$ On input user identifier uid this oracle runs the $\mathbf{UKeygen}(1^\lambda)$ and $\mathbf{CKeygen}(1^\lambda)$ algorithms and forwards to a malicious *Converter* $\mathcal{C}$ the user's secret information $(\mathsf{ek}_{\mathsf{uid}}, \mathsf{tk}_{\mathsf{uid}}, \mathsf{r}_{\mathsf{uid}}, w)$.
- $\mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{C}}}(\mathsf{uid} = i \in \mathbb{U})$ In case of of trustworthy users this oracle returns the secret key of $\mathcal{C}$ to an adversary $\mathcal{A}$.

Collusion Resistant Aggregate Unforgeability The security of the scheme is modeled under the *collusion resistant aggregate unforgeability* ($\mathsf{CR}-\mathsf{AU}$) security definition. An adversary $\mathcal{A}$ is able to obtain valid authentication tags for values of its choice by corrupting users. $\mathcal{A}$ also learns valid encryptions of its choice, and learns the final result over plaintext values $\sum_{i=1}^{n} x_{i,t}$. In the end we claim that an aggregation scheme is secure if a malicious Aggregator $\mathcal{A}$ cannot forge an aggregate tag for a time interval t^* such that for the underlying plaintexts it holds that $\sum_{i=1}^{n} x_{i,t^*} \neq \sum_{i=1}^{n} x_{i,t}$ for a set of users $\mathcal{U}_i \in \mathbb{S}$ that did not collude with the Aggregator or the *Converter*. We follow the security syntax as in [26] and we differentiate between:

- **Type-I** forgeries, in which $\mathcal{A}$ tries to forge for a time interval t^* in which she has not seen any tags from the users.
- **Type-II** forgeries for a time interval t, in which $\mathcal{A}$ has received valid tags for the users but $\mathsf{sum}_{\mathsf{t}^*} \neq \sum x_{i,t}$.

However, in our model we allow a malicious Aggregator or Converter to collude with a user, in pursuance of forging another user's tag and convince the honest data analyzer $\mathcal{DA}$ for the correctness of computations given erroneous data inputs. An adversary during the $\mathsf{CR}-\mathsf{AU}$ game has access to the following oracles:

- $\mathcal{O}^{\mathsf{Setup}}()$: This oracle when queried responds with the public parameters of the scheme pp and the secret key of the Aggregator sk_A.
- $\mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{U}_m}}(\mathsf{uid} = i \in \mathbb{U})$: On input a user identifier uid, this oracle when is queried by a malicious Aggregator $\mathcal{A}$ replies with the secret key of a user $\mathsf{sk}_{\mathsf{uid}}$.
- $\mathcal{O}^{\mathsf{Coll}_{\mathcal{C},\mathcal{U}_m}}(\mathsf{uid} = i \in \mathbb{U})$: Upon receiving a user identifier uid the $\mathcal{O}^{\mathsf{Coll}_{\mathcal{C},\mathcal{U}_m}}$ oracle responds to a malicious *Converter* $\mathcal{C}$ with the secret key of a user $\mathsf{sk}_{\mathsf{uid}}$.
- $\mathcal{O}^{\mathsf{Corr}_{\mathcal{A}}}()$: This oracles responds with the secret decryption key sk_A of the Aggregator.
- $\mathcal{O}^{\mathsf{Corr}_{\mathcal{C}}}()$: This oracles responds with the secret key of the *Converter* $\mathcal{C}$.
- $\mathcal{O}^{\mathsf{Corr}_{\mathcal{DA}}}()$: This oracles responds with the secret verification key vk of the data analyzer $\mathcal{DA}$.
- $\mathcal{O}_{\mathcal{A}}^{\mathsf{EncTag}}(t, \mathsf{uid}, x_{i,t})$: This is an oracle that replies with the encryption of the value $x_{i,t}$ using the secret key of the user i after calling the $\mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{U}_m}}(t, \mathsf{uid} = i \in \mathbb{U})$ or $\mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{C}}}(t, \mathsf{uid} = i \in \mathbb{U})$ oracle.
- $\mathcal{O}_{\mathcal{A}}^{\mathsf{Mtag}}(\mathsf{mtag}_{i,t})$: The $\mathcal{O}^{\mathsf{Mtag}}$ oracle on input a metatag $\mathsf{mtag}_{i,t}$) it converts it to the tag $\sigma_{i,t}$ after corrupting *Converter's* secret key with the $\mathcal{O}^{\mathsf{Corr}_{\mathcal{C}}}$ oracle.
- $\mathcal{O}_{\mathcal{A}}^{\mathsf{Aggregate}}(\{c_{i,t}\}_{i=1}^{n})$: This oracle simulates the behavior of the Aggregator $\mathcal{A}$ and when invoked with inputs the ciphertexts $\{c_{i,t}\}_{i=1}^{n}$, it gives as a response the sum $\sum_{i=1}^{n} x_{i,t}$, after calling the $\mathcal{O}^{\mathsf{Corr}_{\mathcal{A}}}$ oracle, in order to obtain the secret decryption key of the Aggregator sk_A.

– $\mathcal{O}_{\mathcal{A}}^{\mathsf{Verify}}(t, \sigma_{\mathsf{t}}, \mathsf{sum}_{\mathsf{t}})$: Upon receiving a tuple, containing a time interval t, a proof σ_{t} and and a result $\mathsf{sum}_{\mathsf{t}}$, the $\mathcal{O}_{\mathcal{A}}^{\mathsf{Verify}}$ oracle invokes the $\mathcal{O}^{\mathsf{Corr}_{\mathcal{DA}}}$ oracle and replies with $1 \iff \mathsf{sum}_{\mathsf{t}} = \sum_{i=1}^{n} x_{i,t}$, or $\perp$ otherwise.

We model the security definition of $\mathsf{CR} - \mathsf{AU}$, with two games: $\textbf{Game}^{\mathsf{CR-AU}-I}$ and $\textbf{Game}^{\mathsf{CR-AU}-II}$ respectively.

In $\textbf{Game}^{\mathsf{CR-AU}-I}$ users act maliciously and collusions between a user and a malicious Aggregator $\mathcal{A}$ or a $\mathcal{C}$ are allowed. During the learning phase of the game (cf. algorithm 1), $\mathcal{A}$ interacts with $\mathcal{O}^{\mathsf{Setup}}(), \mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{U}_m}}(\mathsf{uid} = i \in \mathbb{U}), \mathcal{O}^{\mathsf{Coll}_{\mathcal{C},\mathcal{U}_m}}(\mathsf{uid} = i \in \mathbb{U}), \mathcal{O}_{\mathcal{A}}^{\mathsf{EncTag}}(t, \mathsf{uid}, x_{i,t}), \mathcal{O}_{\mathcal{A}}^{\mathsf{Mtag}}(\mathsf{mtag}_{i,t}), \mathcal{O}_{\mathcal{A}}^{\mathsf{Verify}}(t, \sigma_{\mathsf{t}}, \mathsf{sum}_{\mathsf{t}})$ oracles, in order to get the public parameters pp, the secret tag key of the user, allow the *Converter* to collude with a malicious user, the ciphertexts, the tags and the metatags of a user, respectively. Finally through $\mathcal{O}_{\mathcal{A}}^{\mathsf{Verify}}(t, \sigma_{\mathsf{t}}, \mathsf{sum}_{\mathsf{t}})$ $\mathcal{A}$ has access to the verification oracle. Note, that this oracle during the game makes sense since, our scheme operates in a symmetric setting, thus $\mathcal{A}$ cannot publicly verify. Finally $\mathcal{A}$ outputs a forgery for a time interval t^*. The forgery is successful if $\textbf{Verify}(\mathsf{pp}, \mathsf{vk}, \mathsf{sum}_t^*, \sigma_t^*) = 1$ for a time interval t^* in which $\mathcal{A}$ did not query the $\mathcal{O}_{\mathsf{EncTag}}$ (**Type-I** forgery), or for t^* in which $\mathcal{A}$ called $\mathcal{O}_{\mathsf{EncTag}}$ (**Type-II** forgery) and none of users $\mathcal{U}_i \in \mathbb{S}$ collude with the Aggregator or the *Converter*.

Algorithm 1: Learning phase of the $\mathsf{CR} - \mathsf{AU} - I$ game

$(\mathsf{pp}, \mathsf{sk}_A) \leftarrow \mathcal{O}_{\mathsf{Setup}}(1^\lambda)$;
// $\mathcal{A}$ executes the following a polynomial number of times
$\mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{U}_m}}(\mathsf{uid} = i \in \mathbb{U})$;
$\mathcal{O}^{\mathsf{Coll}_{\mathcal{C},\mathcal{U}_m}}(\mathsf{uid} = i \in \mathbb{U})$;
// $\mathcal{A}$ is allowed to call $\mathcal{O}_{\mathsf{EncTag}}$ for all users $\mathcal{U}_i$
$(c_{i,t}, \sigma_{i,t}) \leftarrow \mathcal{O}_{\mathsf{EncTag}}(t, \mathsf{uid}_i, x_{i,t})$;
$\mathcal{O}_{\mathcal{A}}^{\mathsf{Mtag}}(\mathsf{mtag}_{i,t})$;
$\mathcal{O}_{\mathcal{A}}^{\mathsf{Verify}}(t, \sigma_{\mathsf{t}}, \mathsf{sum}_{\mathsf{t}})$;

Algorithm 2: Challenge phase of the $\mathsf{CR} - \mathsf{AU} - I$ game

$(t^*, \mathsf{sum}_{t^*}, \sigma_{t^*}) \leftarrow \mathcal{A}$

Definition 1. *($\mathsf{CR} - \mathsf{AU} - I$) An aggregetion scheme is $\mathsf{CR} - \mathsf{AU} - I$ secure if any probabilistic polynomial time adversary $\mathcal{A}$ has negligible probability $\epsilon(\lambda)$ on the winning probabilities $\Pr[\mathcal{A}^{\mathsf{CR-AU}-I}(\lambda)])$ of the game as describe in algorithms 1, 2:* $\Pr[\mathcal{A}^{\mathsf{CR-AU}-I}(\lambda)] \leq \epsilon(\lambda)$.

In $\textbf{Game}^{\mathsf{CR-AU}-II}$ users are assumed as trustworthy and collusions between $\mathcal{C}$ and $\mathcal{A}$ can occur. During the security game though, in the learning phase (cf. algorithm 3) $\mathcal{A}$ does not have access to the $\mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{U}_m}}(\mathsf{uid} = i \in \mathbb{U})$ and $\mathcal{O}^{\mathsf{Coll}_{\mathcal{C},\mathcal{U}_m}}(\mathsf{uid} = i \in \mathbb{U})$ oracles during which users share their secret keys with $\mathcal{A}$ and $\mathcal{C}$. However, $\mathcal{A}$ has access to $\mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{C}}}(\mathsf{uid} = i \in \mathbb{U})$ oracle since Aggregator and *Converter* can collude. Similarly with $\textbf{Game}^{\mathsf{CR-AU}-I}$ $\mathcal{A}$ succeeds if it outputs during the challenge phase (cf. algorithm 4) either a **Type-I** or **Type-II** forgery.

Correspondingly for a scheme with trustworthy users we define:

Definition 2. *($\mathsf{CR} - \mathsf{AU} - II$) An aggregetion scheme is $\mathsf{CR} - \mathsf{AU} - II$ secure if any probabilistic polynomial time adversary $\mathcal{A}$ has negligible probability $\epsilon(\lambda)$ on the winning probabilities $\Pr[\mathcal{A}^{\mathsf{CR-AU}-II}(\lambda)])$ of the game as describe in algorithms 3, 4:* $\Pr[\mathcal{A}^{\mathsf{CR-AU}-II}(\lambda)] \leq \epsilon(\lambda)$.

Aggregator Obliviousness The privacy guarantees of the scheme assure Aggregator obliviousness (AO) as has been first modeled by Shi *et al.* [31] and followed in subsequent work [22, 25, 26]. In a nutshell,

Algorithm 3: Learning phase of the $\mathsf{CR}-\mathsf{AU}-II$ game

$(\mathsf{pp}, \mathsf{sk_A}) \leftarrow \mathcal{O}_{\mathsf{Setup}}(1^\lambda)$;
// $\mathcal{A}$ executes the following a polynomial number of times
// $\mathcal{A}$ is allowed to call $\mathcal{O}_{\mathsf{EncTag}}$ for all users $\mathcal{U}_\mathsf{i}$
$(c_{i,t}, \sigma_{i,t}) \leftarrow \mathcal{O}_{\mathsf{EncTag}}(t, \mathsf{uid}_i, x_{i,t})$;
$\mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{C}}}(\mathsf{uid} = i \in \mathbb{U})$;
$\mathcal{O}^{\mathsf{Mtag}}_{\mathcal{A}}(\mathsf{mtag}_{i,t})$;
$\mathcal{O}^{\mathsf{Verify}}_{\mathcal{A}}(t, \sigma_\mathsf{t}, \mathsf{sum_t})$;

Algorithm 4: Challenge phase of the $\mathsf{CR}-\mathsf{AU}-II$ game

$(t^*, \mathsf{sum}_{t^*}, \sigma_{t^*}) \leftarrow \mathcal{A}$

a malicious Aggregator $\mathcal{A}$ or *Converter* $\mathcal{C}$ cannot compromise individual privacy by learning in cleartext the sum over users' data inputs. The privacy definition has been augmented in order to capture the extra functionality of unforgeability. As such, an adversary $\mathcal{A}$ is able to observe apart from ciphertexts, metatags and the final convertible tag.

We are focused on AO since the Aggregator learns most of the information during the protocol execution. It is the party, which in contrast with the security definition of $\mathsf{CR}-\mathsf{AU}-I$ and $\mathsf{CR}-\mathsf{AU}-II$, has access to all collusion oracles $\mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{U}_m}}(\mathsf{uid} = i \in \mathbb{U}), \mathcal{O}^{\mathsf{Coll}_{\mathcal{C},\mathcal{U}_m}}(\mathsf{uid} = i \in \mathbb{U}), \mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{C}}}(\mathsf{uid} = i \in \mathbb{U})$ during the learning phase of the game in algorithm 5. At the challenge phase (cf. algorithm 6), $\mathcal{A}$ chooses a subset of users $\mathbb{S}^*$ that have not been corrupted and issues two time series data $\mathcal{X}^0_{t^*} = (\mathcal{U}_i, t^*, x^0_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$ and $\mathcal{X}^1_{t^*} = (\mathcal{U}_i, t^*, x^1_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$, such that $\sum x^0_{i,t^*} = \sum_{\mathcal{U}_i \in \mathbb{S}^*} x^1_{i,t^*}$ for a time interval t^* and sends them to the $\mathcal{O}_{\mathsf{AO}}(\mathcal{X}^0_{t^*}, \mathcal{X}^1_{t^*})$ oracle.

$\mathcal{O}_{\mathsf{AO}}(\mathcal{X}^0_{t^*}, \mathcal{X}^1_{t^*})$ upon receiving the time series data flips a random coin $b \xleftarrow{\$} \{0, 1\}$ and replies to $\mathcal{A}$ with the ciphertexts $\{c^b_{i,t}\}_{\mathcal{U}_i \in \mathbb{S}^*}$ the metatags $\{\mathsf{mtag}^b_{i,t}\}_{\mathcal{U}_i \in \mathbb{S}^*}$ and the tags $\{\sigma^b_{i,t}\}_{\mathcal{U}_i \in \mathbb{S}^*}$. $\mathcal{A}$ can adaptively call the $\mathcal{O}^{\mathsf{Verify}}_{\mathcal{A}}(t, \sigma_\mathsf{t}, \mathsf{sum_t})$ oracle after the challenge.

At the end of the game $\mathcal{A}$ outputs its guess b^*, and $\mathcal{A}$ wins the game $\iff b^* = b$.

Definition 3 (Aggregator Obliviousness). *Let* $\Pr[\mathcal{A}^{\mathbf{AO}}]$ *denote the probability that Aggregator* $\mathcal{A}$ *outputs* $b^* = b$*. Then an aggregation protocol is said to ensure Aggregator obliviousness if for any polynomially bounded Aggregator* $\mathcal{A}$ *the probability* $\Pr[\mathcal{A}^{\mathbf{AO}}] \leqslant \frac{1}{2} + \epsilon(\lambda)$*, where* ϵ *is a negligible function and* λ *is the security parameter.*

5 Preliminaries

In this section we explain the basic building blocks and computation assumptions that are used in our proofs.

5.1 Bilinear maps

Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be cyclic groups of large prime order p and g_1, g_2 generators of $\mathbb{G}_1, \mathbb{G}_2$ accordingly. We say that e is a bilinear map, if the following properties are satisfied:

1. *bilinearity*: $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$, where $g_1, g_2 \in \mathbb{G}_1 \times \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$.
2. *Computability*: there exists an efficient algorithm that computes $e(g_1^a, g_2^b)$ where $g_1, g_2 \in \mathbb{G}_1 \times \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$.
3. *Non-degeneracy*: $e(g_1, g_2) \neq 1$.

5.2 Computational Assumptions

Definition 4. *(Bilinear Computational Diffie-Hellman (*BCDH*) Assumption)*

Let $e(\mathbb{G}_1 \times \mathbb{G}_2) \to \mathbb{G}_T$ *be a bilinear pairing,* g *a generator of* $\mathbb{G}_1$ *and* g_2 *a generator of* $\mathbb{G}_2$ *and* p *the order of* $\mathbb{G}_1, \mathbb{G}_2$ *and* $\mathbb{G}_T$*. Given* $U = (g, g^a, g^b, g^c) \in \mathbb{G}_1$ *and* $V = (g_2, g_2^a, g_2^b) \in \mathbb{G}_2$ *for random* $a, b, c \in \mathbb{Z}_p$

Algorithm 5: Learning phase of the Aggregator obliviousness game

$(\mathsf{pp}, \mathsf{sk_A}, \mathsf{vk}) \leftarrow \mathcal{O}_{\mathsf{Setup}}(1^\lambda)$;
$\mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{U}_m}}(\mathsf{uid} = i \in \mathbb{U})$;
$\mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{C}}}(\mathsf{uid} = i \in \mathbb{U})$;
// $\mathcal{A}$ executes the following a polynomial number of times
// $\mathcal{A}$ is allowed to call $\mathcal{O}_{\mathsf{EncTag}}$ for all users $\mathcal{U}_\mathsf{i}$
$(c_{i,t}, \sigma_{i,t}) \leftarrow \mathcal{O}_{EncTag}(t, \mathsf{uid}_i, x_{i,t})$;
$\mathcal{O}_{\mathcal{A}}^{\mathsf{Mtag}}(\mathsf{mtag}_{i,t})$;
$\mathcal{O}_{\mathcal{A}}^{\mathsf{Verify}}(t, \sigma_\mathsf{t}, \mathsf{sum_t})$;

Algorithm 6: Challenge phase of the Aggregator obliviousness game

$\mathcal{A} \rightarrow t^*, \mathbb{S}^*$;
$\mathcal{A} \rightarrow \mathcal{X}_{t^*}^0, \mathcal{X}_{t^*}^1$;
$(c_{i,t^*}^b, \sigma_{i,t^*}^b)_{\mathcal{U}_i \in \mathbb{S}^*} \leftarrow \mathcal{O}_{\mathsf{AO}}(\mathcal{X}_{t^*}^0, \mathcal{X}_{t^*}^1)$;
$\mathcal{A} \rightarrow b^*$;

we say that BCDH *holds if the probabilities of a probabilistic polynomial time adversary $\mathcal{A}$ to compute $W = e(g_1, g_2)^{abc}$ are negligible on input the security parameter λ:* $\Pr[W \leftarrow \mathcal{A}(U, V)]$.

Definition 5. *(eXternal Diffie-Hellman (*XDH*) Assumption)*
Let $e(\mathbb{G}_1 \times \mathbb{G}_2) \rightarrow \mathbb{G}_T$ be a bilinear pairing, g a generator of $\mathbb{G}_1$ and g_2 a generator of $\mathbb{G}_2$ and p the order of $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$. We say that XDH *holds if the probabilities of a probabilistic polynomial time adversary $\mathcal{A}$ solve* DDH *and* DL *in $\mathbb{G}_1$ are negligible on input the security parameter λ.*

Definition 6. *(Fixed Argument Pairing Inversion I (*FAPI − I*) Assumption) [18]*
Let $e(\mathbb{G}_1 \times \mathbb{G}_2) \rightarrow \mathbb{G}_T$ be a bilinear pairing, $d_1 \in \mathbb{G}_1, d_2 \in \mathbb{G}_2$ and $e(d_1, d_2) = z \in \mathbb{G}_T$ We say that FAPI − I *holds if the probabilities of a probabilistic polynomial time adversary $\mathcal{A}$ $\Pr[d_2 \leftarrow \mathcal{A}(d_1, z)]$ are negligible on input the security parameter λ.*

6 Protocol

In order to guarantee AO our protocol employs Shi *et al.* scheme [31]. For completeness we briefly describe their encryption scheme.

6.1 Shi-Chan-Rieffel-Chow-Song Scheme

- **Setup**(1^λ): On input the security parameter λ this probabilistic algorithm outputs a cryptographic secure hash function $H : \{0,1\}^* \rightarrow \mathbb{G}_1$, for a group $\mathbb{G}_1$ of large prime order p. Through a secure channel the trusted key dealer $\mathcal{KD}$ distributes to each user a secret encryption key $\mathsf{ek}_i \in \mathbb{Z}_p$, which is chosen uniformly at random. $\mathcal{KD}$ also forwards to the $\mathcal{A}$ the secret decryption key $\mathsf{sk_A} = \sum_{i=1}^n \mathsf{ek}_i$.
- **Encrypt**$(\mathsf{ek}_i, x_{i,t})$: To encrypt data value $x_{i,t}$ at time interval t with secret key ek_i, user $\mathcal{U}_i$ computes the ciphertext $c_{i,t} = H(t)^{\mathsf{ek}_i} g_1^{x_{i,t}} \in \mathbb{G}_1$.
- **Aggregate**$(\{c_{i,t}\}_{\mathcal{U}_i \in \mathbb{U}}, \{\sigma_{i,t}\}_{\mathcal{U}_i \in \mathbb{U}}, \mathsf{sk_A})$: Upon receiving all the ciphertexts $\{c_{i,t}\}_{i=1}^n$, the Aggregator computes: $V_t = (\prod_{i=1}^n c_{i,t}) H(t)^{-\mathsf{sk}_\mathcal{A}} = H(t)^{\sum_{i=1}^n \mathsf{ek}_i} g_1^{\sum_{i=1}^n x_{i,t}} H(t)^{-\sum_{i=1}^n \mathsf{ek}_i} = g_1^{\sum_{i=1}^n x_{i,t}} \in \mathbb{G}_1$. $\mathcal{A}$ then learns the sum $\mathsf{sum_t} = \sum_{i=1}^n x_{i,t} \in \mathbb{Z}_p$ by computing the discrete logarithm of V_t on the base g_1. The sum computation is correct as long as $\sum_{i=1}^n x_{i,t} < p$.

6.2 Collusion resistant aggregation I (CRA-I)

In order to communicate the ideas of the protocol in a clear way we first define a protocol that is collusion resistant between colluding users and a malicious Aggregator $\mathcal{A}$.

- **Setup**$_I(1^\lambda)$: On input the security parameter λ this probabilistic algorithm defines a cryptographic secure hash function $H : \{0,1\}^* \to \mathbb{G}_1$, a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ of prime order p with generator g. Finally it outputs the public parameters $\mathsf{pp} = (H, e, g, g_2)$. It also calls the **Setup**(1^λ) algorithm of the Shi *et al.* scheme and outputs the secret key of the Aggregator sk_A
- **UKeygen**$_I(1^\lambda)\langle \mathcal{KD}, \mathbb{U}\rangle$: Each user independently chooses uniformly random tag keys tk_i and r_i. Through a secure channel each $\mathcal{U}_i$ forwards r_i to the key dealer $\mathcal{KD}$, who computes $\sum_{i=1}^n \mathsf{r}_i$.
- **CKeygen**$_I(1^\lambda)\langle \mathcal{KD}, \mathbb{U}\rangle$: The key dealer chooses uniformly at random a key $r \in \mathbb{Z}_p$ and a random generator $w \in \mathbb{G}_2$. It distributes through a secure channel r to the *Converter* $\mathcal{C}$. It also sends to the $\mathcal{DA}$ the secret verification key $\mathsf{vk} = (w, r, \sum_{i=1}^n \mathsf{r}_i)$. Moreover it forwards w to each user. Then the key dealer $\mathcal{KD}$ goes off-line.
- **EncTag**$_I(\mathsf{pp}, \mathsf{sk_i}, \mathsf{x_{i,t}})$: This deterministic algorithm takes as input the secret key of each user $\mathsf{sk_i} = (\mathsf{r_i}, \mathsf{w}, \mathsf{tk_i}, \mathsf{ek_i})$ and the private values $x_{i,t}$ and outputs the metatag:

$$\mathsf{mtag}_{i,t} = (\mathsf{mtag}^1_{i,t}, \mathsf{mtag}^2_{i,t}) = ([H(t)^{\mathsf{r_i}} g^{x_{i,t}}]^{\mathsf{tk_i}}, w^{\frac{1}{\mathsf{tk_i}}})$$

  Moreover, users encrypt their data with the encryption key ek_i, with the encryption scheme of Shi *et al.* [31] as already presented in 6.1. Finally, $\mathcal{U}_i$ forwards $c_{i,t}$ to the Aggregator $\mathcal{A}$ and the metatag $\mathsf{mtag}_{i,t}$ to the Converter.
- **Convert**$_I(\mathsf{pp}, r, \mathsf{mtag}_{i,t})$: The *Converter* runs this algorithm in order to "unify" all the tags under the same key. It allows the homomorphic operations on the tags. The algorithm takes as input the public parameters pp, the key r, and metatag $\mathsf{mtag}_{i,t}$ and outputs the tag $\sigma_{i,t}$ as follows:

$$\sigma_{i,t} = e(\mathsf{mtag}^1_{i,t}, \mathsf{mtag}^2_{i,t})^r = e([H(t)^{\mathsf{r_i}} g^{x_{i,t}}]^{\mathsf{tk_i}}, w^{\frac{1}{\mathsf{tk_i}}})^r =$$
$$e(H(t)^{\mathsf{r_i}\ \mathsf{tk_i}}, w^{\frac{1}{\mathsf{tk}\ i}})^r e(g^{x_{i,t}\mathsf{tk}_i}, w^{\frac{1}{\mathsf{tk_i}}})^r = e(H(t)^{\mathsf{r_i}}, w)^r e(g^{x_{i,t}}, w)^r$$

- **Aggregate**$_I(\mathsf{sk}_A, \{c_{i,t}\}, \{\sigma_{i,t}\})$: The Aggregator $\mathcal{A}$ after collecting all the ciphertexts $c_{i,t}$ for the users $\mathbb{U}$ decrypts with the secret key sk_A and learns the sum $\sum_{i=}^n x_{i,t}$. For the decryption algorithm $\mathcal{A}$ uses the decryption algorithm as in Shi *et al.* scheme [31]. Moreover, $\mathcal{A}$ computes a proof of correct computation by aggregating the tags $\sigma_{i,t}$ as follows:

$$\sigma_{\mathsf{t}} = \prod_{i=1}^n \sigma_{i,t} = \prod_{i=1}^n e(\mathsf{mtag}^1_{i,t}, \mathsf{mtag}^2_{i,t})^r = \prod_{i=1}^n e(H(t)^{\mathsf{r_i}}, w)^r e(g^{x_{i,t}}, w)^r =$$
$$\prod_{i=1}^n e(H(t)^{\mathsf{r_i}}, w)^r \prod_{i=1}^n e(g^{x_{i,t}}, w)^r = e(H(t), w)^{r \sum_{i=1}^n \mathsf{r_i}} e(g, w)^{r \sum_{i=1}^n x_{i,t}}$$

  Finally $\mathcal{A}$ returns to the honest verifier the result $\mathsf{sum_t} = \sum_{i=}^n x_{i,t}$ and the proof $\sigma_{\mathsf{t}} = e(H(t), w)^{r \sum_{i=1}^n \mathsf{r_i}} e(g, w)^{r \sum_{i=1}^n x_{i,t}}$
- **Verify**$_I(\mathsf{pp}, \mathsf{vk}, \mathsf{sum_t}, \sigma_{\mathsf{t}})$: The data analyzer $\mathcal{DA}$, who acts as honest verifier verifies the correctness of the sum computation by employing its verification key $\mathsf{vk} = (\mathsf{vk}_1 = w, \mathsf{vk}_2 = r, \mathsf{vk}_3 = \sum_{i=1}^n \mathsf{r}_i)$. $\mathcal{DA}$ verifies by checking if the following equation holds:

$$e(H(t), \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{vk}_3} e(g, \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{sum_t}} \stackrel{?}{=} \sigma_{\mathsf{t}}$$

  Thanks to the bilinearity of the pairings the correctness of the verification procedure is assured. Indeed:

$$e(H(t), \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{vk}_3} e(g, \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{sum_t}} =$$
$$e(H(t), w)^{r \sum_{i=1}^n \mathsf{r_i}} e(g, w)^{r \sum_{i=1}^n x_{i,t}} = \sigma_{\mathsf{t}}$$

6.3 Collusion resistant aggregation II (CRA-II)

We now present an extension of the previous scheme in order mitigate collusions between users and a malicious $\mathcal{A}$ and between users and malicious $\mathcal{C}$, meaning that a user can collude at the same time with $\mathcal{A}$ and $\mathcal{C}$. First we define a simple attack on the previous scheme:

Attack on CRA-I scheme A colluding user $\mathcal{U}_c$ shares with the *Converter* his secret tag key tk_i and the shared common key between all users w. $\mathcal{C}$ can forge a valid tag $\mathsf{tag}_{i,t}$ for a trustworthy user $\mathcal{U}_i$ as follows: $\mathsf{tag}'_{i,t} = \mathsf{tag}_{i,t} \cdot e(g^{x'_{i,t}}, w) = e(H(t)^{\mathsf{r}_i}, w)^r e(g^{x_{i,t}+x'_{i,t}}, w)^r$, which is a valid forge for the value $x_{i,t} + x'_{i,t}$.

The core idea to mitigate these type of attacks is to enforce the *Converter* $\mathcal{C}$ to re-randomize the metatag $\mathsf{mtag}_{i,t} = [H(t)^{\mathsf{r}_i} g^{x_{i,t}}]^{\mathsf{tk}_i}$ with the randomness r, such that $\mathcal{C}$ replies to $\mathcal{U}_i$ with the final tag $\sigma_{i,t} = e(H(t)^{\mathsf{r}_i}, w)^r e(g^{x_{i,t}}, w)^r$ along with the randomized metatag $\mathsf{mtag}_{i,t} = [H(t)^{\mathsf{r}_i} g^{x_{i,t}}]^{r\mathsf{tk}_i}$. Finally the user recomputes the final tag from the randomized metatag and validates whether the final tag has been forged. As such, in case of collusions between a malicious user and a malicious $\mathcal{C}$, the latter can forge the final tag, but the user can detect it, thanks to the unforgeability of the metatag $\mathsf{mtag}_{i,t}$. We describe the entire protocol for collusion resistant aggregation against $\mathcal{C}$ and $\mathcal{A}$:

- $\mathbf{Setup}_{II}(1^\lambda)$: This algorithm calls the $\mathbf{Setup}_I(1^\lambda)$ algorithm and outputs the public parameters $\mathsf{pp} = (H, e, g, g_2)$ and the secret key of the Aggregator sk_A
- $\mathbf{UKeygen}_{II}(1^\lambda)\langle \mathcal{KD}, \mathbb{U}\rangle$: $\mathbf{UKeygen}_{II}(1^\lambda)$ invokes the $\mathbf{UKeygen}_I(1^\lambda)$ algorithm during which each user independently chooses uniformly random tag keys tk_i and r_i. Moreover users transmit r_i, through a secure channel to the key dealer who computes $\sum_{i=1}^n \mathsf{r}_i$.
- $\mathbf{CKeygen}_{II}(1^\lambda)\langle \mathcal{KD}, \mathbb{U}, \mathcal{C}, \mathcal{DA}\rangle$: This algorithm calls the $\mathbf{CKeygen}_I(1^\lambda)\langle \mathcal{KD}, \mathbb{U}, \mathcal{C}, \mathcal{DA}\rangle$, in which the key dealer outputs the secret verification key $\mathsf{vk} = (w, r, \sum_{i=1}^n \mathsf{r}_i)$, chooses uniformly at random a key $r \in \mathbb{Z}_p$ and a random generator $w \in \mathbb{G}_2$. It distributes through a secure channel r to the *Converter* $\mathcal{C}$. It also sends w to each user, and forwards to the $\mathcal{DA}$ the secret verification key $\mathsf{vk} = (w, r, \sum_{i=1}^n \mathsf{r}_i)$. Finally the key dealer $\mathcal{KD}$ goes off-line.
- $\mathbf{EncTag}_{II}(\mathsf{pp}, \mathsf{sk}_i, x_{i,t})$: $\mathbf{EncTag}_{II}(\mathsf{pp}, \mathsf{sk}_i, x_{i,t})$ calls $\mathbf{EncTag}_I(\mathsf{pp}, \mathsf{sk}_i, x_{i,t})$ and operates similarly. It outputs for each user $\mathcal{U}_i$ the ciphertext $c_{i,t}$ and the metatag :

$$\mathsf{mtag}_{i,t} = (\mathsf{mtag}^1_{i,t}, \mathsf{mtag}^2_{i,t}) = ([H(t)^{\mathsf{r}_i} g^{x_{i,t}}]^{\mathsf{tk}_i}, w^{\frac{1}{\mathsf{tk}_i}})$$

  which is forwarded to the *Converter* $\mathcal{C}$.
- $\mathbf{Convert}_{II}(\mathsf{pp}, r, \mathsf{mtag}_{i,t})$: Upon receiving the metatag $\mathsf{mtag}_{i,t} = (\mathsf{mtag}^1_{i,t}, \mathsf{mtag}^2_{i,t}) = ([H(t)^{\mathsf{r}_i} g^{x_{i,t}}]^{\mathsf{tk}_i}, w^{\frac{1}{\mathsf{tk}_i}})$, $\mathcal{C}$ uses its secret key r to compute the final tag as follows:

$$\sigma^1_{i,t} = e(\mathsf{mtag}^1_{i,t}, \mathsf{mtag}^2_{i,t})^r = e([H(t)^{\mathsf{r}_i} g^{x_{i,t}}]^{\mathsf{tk}_i}, w^{\frac{1}{\mathsf{tk}_i}})^r =$$
$$e(H(t)^{\mathsf{r}_i \cdot \mathsf{tk}_i}, w^{\frac{1}{\mathsf{tk}_i}})^r e(g^{x \cdot \mathsf{tk}_i}, w^{\frac{1}{\mathsf{tk}_i}})^r = e(H(t)^{\mathsf{r}_i}, w)^r e(g^{x_{i,t}}, w)^r$$

$$\sigma^2_{i,t} = (\mathsf{mtag}^1_{i,t})^r = [H(t)^{\mathsf{r}_i} g^{x_{i,t}}]^{r\mathsf{tk}_i}$$ [1]

  Finally $\mathcal{C}$ sends to $\mathcal{U}_i$ the final tag $\sigma_{i,t} = (\sigma^1_{i,t}, \sigma^2_{i,t})$.
- $\mathbf{VTag}_{II}(\mathsf{pp}, \mathsf{sk}_i, \sigma_{i,t}, x_{i,t})$: Each user verifies the correctness of the final tag as follows:

$$e(\sigma^2_{i,t}, w)^{\frac{1}{\mathsf{tk}_i}} \stackrel{?}{=} \sigma^1_{i,t}$$

  The correctness of the equation holds since:

$$e(\sigma^2_{i,t}, w)^{\frac{1}{\mathsf{tk}_i}} = e([H(t)^{\mathsf{r}_i} g^{x_{i,t}}]^{r\mathsf{tk}_i}, w)^{\frac{1}{\mathsf{tk}_i}} =$$
$$e(H(t)^{\mathsf{r}_i} g^{x_{i,t}}, w)^{\frac{r\mathsf{tk}_i}{\mathsf{tk}_i}} = e(H(t)^{\mathsf{r}_i} g^{x_{i,t}}, w)^r = \sigma^1_{i,t}$$

  At this point if the equation is not true the user $\mathcal{U}_i$ halts the execution of the protocol and it infers that $\mathcal{C}$ forged the tag $\sigma_{i,t}$. Otherwise it continues by sending the final tag $\sigma_{i,t} = \sigma^1_{i,t}$ to the Aggregator $\mathcal{A}$.
- $\mathbf{Aggregate}_{II}(\mathsf{sk}_A, \{c_{i,t}\}, \{\sigma_{i,t}\})$: This algorithm calls $\mathbf{Aggregate}_I(\mathsf{sk}_A, \{c_{i,t}\}, \{\sigma_{i,t}\})$, which consecutively decrypts with the secret key sk_A and $\mathcal{A}$ learns $\mathsf{sum}_\mathsf{t} = \sum_{i=1}^n x_{i,t}$. Moreover, it computes a proof of correct computation σ_t and finally and forwards the result $\mathsf{sum}_\mathsf{t} = \sum_{i=} x_{i,t}$ and the proof $\sigma_\mathsf{t} = \prod_{i=1}^n \sigma_{i,t} = e(H(t), w)^{r \sum_{i=1}^n \mathsf{r}_i} e(g, w)^{r \sum_{i=1}^n x_{i,t}}$ to the data analyzer $\mathcal{DA}$.
- $\mathbf{Verify}_{II}(\mathsf{pp}, \mathsf{vk}, \mathsf{sum}_\mathsf{t}, \sigma_\mathsf{t})$: The $\mathbf{Verify}_{II}(\mathsf{pp}, \mathsf{vk}, \mathsf{sum}_\mathsf{t}, \sigma_\mathsf{t})$ algorithm invokes $\mathbf{Verify}_I(\mathsf{pp}, \mathsf{vk}, \mathsf{sum}_\mathsf{t}, \sigma_\mathsf{t})$ and verifies the correctness of the sum computation by checking :

$$e(H(t), \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{vk}_3} e(g, \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{sum}_\mathsf{t}} \stackrel{?}{=} \sigma_\mathsf{t}$$

[1] gray background denotes the different crypto machinery needed to prevent collusions

7 Cost Analysis

Participant	Computation	Communication
User	$\mathbf{2}$ EXP $\in \mathbb{G}_1 + \mathbf{1}$ EXP $\in \mathbb{G}_2 + \mathbf{1}$ INV $\in \mathbb{Z}_p + \mathbf{1}$ EXP $\in \mathbb{G}_T + \mathbf{1}$ PAIR	$\mathbf{2l} + \mathbf{l_T}$
Converter	$\mathbf{1}$ PAIR $+\mathbf{1}$ EXP $\in \mathbb{G}_T$	$\mathbf{l} + \mathbf{l_T}$
Aggregator	$(\mathbf{n-1})$ MUL $\in \mathbb{G}_T$	$\mathbf{l} + \mathbf{l_T}$
Data Analyzer	$\mathbf{1}$ HASH $\in \mathbb{G}_1 + \mathbf{2}$ EXP $\in \mathbb{G}_T + \mathbf{2}$ MUL $\in \mathbb{Z}_p + \mathbf{2}$ PAIR	-

Table 2: Performance of tag computation and metatag computation, proof construction and verification operations. l denotes the bit-size of the prime number p and l_T the bit-size of elements in $\mathbb{G}_T$.

We perform a theoretical evaluation of the scheme with respect to the cardinality of operations that have to be performed by each party during the protocol execution for collusion resistant unforgeability. The results are depicted in table 2. Notice that we omit from the analysis the computational costs for encryption per user and decryption time for the Aggregator, since our goal is to show the extra cost for collusion resistant unforgeability. That is the costs to compute tags and metatags, convert them, per user verify them, aggregate them at the Aggregator's side and compute/verify the proof of correct computation.

At each time interval for the computation of the metatag $\mathsf{mtag}_{i,t} = [H(t)^{\mathsf{r}_i} g^{x_{i,t}}]^r, w^{\frac{1}{\mathsf{tk}_i}}, \mathcal{U}_i$ is committed to two exponentiations in $\mathbb{G}_1$ and one exponentiation in $\mathbb{G}_2$. Afterwards, in order to validate the final tag, users check if the following equation holds: $e(\sigma_{i,t}^2, w)^{\frac{1}{\mathsf{tk}_i}} \stackrel{?}{=} \sigma_{i,t}^2$, by performing one exponentiation in $\mathbb{G}_T$ and one bilinear pairing operation. The *Converter*, in order to convert the metatag $\mathsf{mtag}_{i,t}$ to the final tag $= \sigma_{i,t} = e(H(t)^{\mathsf{r}_i}, w)^r e(g^{x_{i,t}}, w)^r$, is committed to one bilinear pairing computation and one exponentiation in $\mathbb{G}_T$. The Aggregator computes the proof with $n-1$ multiplications in $\mathbb{G}_T$: $\sigma_{\mathsf{t}} = \prod_{i=1}^{n} \sigma_{i,t} = e(H(t), w)^{r \sum_{i=1}^{n} \mathsf{r}_i} e(g, w)^{r \sum_{i=1}^{n} x_{i,t}}$ and the data analyzer verifies with two multiplications in $\mathbb{G}_T$, two exponentiations in $\mathbb{G}_T$ and two bilinear pairing evaluations: $e(H(t), \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{vk}_3} e(g, \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{sum}_{\mathsf{t}}} \stackrel{?}{=} \sigma_{\mathsf{t}}$. Notice, that the protocol achieves constant verification time, which does not dependent on the number of users that participate in the protocol with their values.

We also performed a real world prototype implementation for the verification process on a machine running Ubuntu 14.04 with kernel version 3.19.0-29. The machine has 8MB RAM memory and is equipped with an INTEL Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz processor with 4 cores. For our prototype implementation we used python version 3 and charm cryptographic framework[1]. Charm supports an abstract layer for basic cryptographic primitives and its core system for the mathematical operations is implemented in ANSI C, yielding substantial efficiency results. Since out protocol is based on bilinear pairings we run our benchmarks with different implemented elliptic curves (cf. table 3). We measured the exact time of computing the metatags, the final tag and the verification of the final tag from the user side. Secondly, we measured the computational overhead at the converter side for transforming the metatags of each user. Finally, the time for verification is computed, for different curve types. The implementation results are obtained on average after running them for 1000 times. We observed that for the computation and verification of each tag, user computations yield a cost of microseconds ($\approx \mathbf{3}$ ms on average for all different type of curves) . The cost for the Converter to convert each tag remains small compared with the computation of each tag, thanks to the simplicity of the Convert operations. The exact computational cost for verifying is also significantly low and it is constant, independent on the number of the users.

Table 3: Benchmark results

Operation	Curve Type		
	MNT159	MNT201	MNT224
User : Tag	**6** ms	**10** ms	**12** ms
Converter : Convert	**3** ms	**5** ms	**6** ms
DA : Verify	**8** ms	**11** ms	**13** ms

8 Security Analysis

In this section we give evidence for the security of the scheme, following the security definitions in section 4.3. We start our analysis with privacy and we prove the *Aggregator unforgeability* privacy property. Notice that be it **CRA-I** or **CRA-II** the privacy guarantee is not affected as with the encryption scheme of Shi *et al.* [31] in case of corrupted users, thanks to the trusted key dealer that distributes individual secret keys to each user. As such, we assume a trusted key distribution phase before the key dealer $\mathcal{KD}$ goes off-line.

8.1 Aggregator Obliviousness

Theorem 1. *The **CRA-I** and **CRA-II** schemes provide Aggregator Obliviousness under the DDH assumption in $\mathbb{G}_1$ in the random oracle mode.*

Proof. We assume an adversary $\mathcal{A}$ who breaks with non-negligible probability the AO privacy definition for *Aggregator obliviousness*. We will show in our proof how a probabilistic polynomial time adversary $\mathcal{B}$ invokes $\mathcal{A}$ as a subroutine in order to break the *Aggregator obliviousness* definition as defined in the scheme of Shi *et al.* [31]. We will refer to this scheme as private streaming aggregation (PSA). Adversary $\mathcal{B}$ has access to $\mathcal{O}^{\mathsf{PSA}}_{\mathsf{Setup}}$, $\mathcal{O}^{\mathsf{PSA}}_{\mathsf{Corrupt}}$, $\mathcal{O}^{\mathsf{PSA}}_{\mathsf{Encrypt}}$, and $\mathcal{O}^{\mathsf{PSA}}_{\mathsf{AO}}$ oracles with the challenger, when she tries to break AO in PSA. The $\mathcal{O}^{\mathsf{PSA}}_{\mathsf{Setup}}$ oracle gives the public parameters and the secret keys to the users and the Aggregator. The $\mathcal{O}^{\mathsf{PSA}}_{\mathsf{Corrupt}}$ oracle on input a user id uid returns the secret encryption key sk_i of a corrupted user. The $\mathcal{O}^{\mathsf{PSA}}_{\mathsf{Encrypt}}$ oracle on input a data input $x_{i,t}$ returns the encryption $c_{i,t}$ under the encryption algorithm of [31]. The $\mathcal{O}^{\mathsf{PSA}}_{\mathsf{AO}}$ oracle during the challenge phase with $\mathcal{B}$ flips a random coin $b \xleftarrow{\$} \{0,1\}$ and responds with the encryption of the time series $\mathcal{X}^b_t = \{x_{i,t}\}$.

Algorithm $\mathcal{B}$ simulates as a challenger the oracles $\mathcal{A}$ has access to during the **Learning phase** as follows:

- $\mathcal{O}_{\mathsf{Setup}}(1^\lambda)$: Whenever $\mathcal{A}$ calls the $\mathcal{O}_{\mathsf{Setup}}(1^\lambda)$ oracle, $\mathcal{B}$ calls the $\mathcal{O}^{\mathsf{PSA}}_{\mathsf{Setup}}$ oracle, which responds to $\mathcal{B}$ with a hash function $H : \{0,1\}^* \to \mathbb{G}_1$, a generator g of the group $\mathbb{G}_1$ of safe prime order p, and the Aggregator's secret key $\mathsf{sk}_A = \sum_{i=1}^{n} \mathsf{ek_i}$. Moreover, $\mathcal{B}$ chooses the parameters of a bilinear pairing $bp = (e, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$. Uniformly at random it selects secret keys $r, \{\mathsf{r}_i\}_{i=1}^n \in \mathbb{Z}_p, w \in \mathbb{G}_2$. Finally $\mathcal{B}$ replies to $\mathcal{A}$ with H, g, bp, sk_A.
- $\mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{U}_m}}(\mathsf{uid} = i \in \mathbb{U})$: When $\mathcal{A}$ invokes this oracle then $\mathcal{B}$ calls the $\mathcal{O}^{\mathsf{PSA}}_{\mathsf{Corrupt}}$ oracle and transmits to $\mathcal{A}$ the secret encryption key ek_c of a corrupted user $\mathcal{U}_c \in \mathbb{S}$ and its secret tag key r_i, w.
- $\mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{C}}}(\mathsf{uid} = i \in \mathbb{U})$: The collusion between the *Converter* and $\mathcal{A}$ are simulated by the $\mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{C}}}(\mathsf{uid} = i \in \mathbb{U})$ oracle. $\mathcal{B}$ replies on these calls with the secret key r.
- $\mathcal{O}_{\mathsf{EncTag}(t,\mathsf{uid},x_{i,t})}$: Upon call on the $\mathcal{O}_{\mathsf{EncTag}(t,\mathsf{uid},x_{i,t})}$ oracle, $\mathcal{B}$ invokes the $\mathcal{O}^{\mathsf{PSA}}_{\mathsf{Encrypt}}$ with input $(t, \mathsf{uid}, x_{i,t})$, which in turns reply to $\mathcal{B}$ with the encryption $c_{i,t} = H(t)^{\mathsf{ek}_i} g^{x}_{i,t}$ of $x_{i,t}$. $\mathcal{B}$ also computes $\mathsf{mtag}_{i,t} = c^{\mathsf{r}_i}_{i,t}, w^{\frac{1}{\mathsf{r}_i}} = (H(t)^{\mathsf{ek}_i} g^{x}_{i,t})^{\mathsf{r}_i}, w^{\frac{1}{\mathsf{r}_i}}$. Notice that $\mathsf{mtag}_{i,t}$ is indistinguishable from the real one if we interchange the randomness and set $\mathsf{r}_i = \mathsf{ek}_i$ and $\mathsf{tk}_i = \mathsf{r}_i$, for uniformly random keys $\mathsf{ek}_i, \mathsf{r}_i$. $\mathcal{B}$. Finally $\mathcal{B}$ replies to $\mathcal{A}$ with $(c_{i,t}, \mathsf{mtag}_i, t)$.
- $\mathcal{O}^{\mathsf{Mtag}}_{\mathcal{A}}(\mathsf{mtag}_{i,t})$: $\mathcal{A}$ calls this oracle in order to learn the final tag of each user $\sigma_{i,t}$. $\mathcal{B}$ computes the final tag as $\sigma_{i,t} = e(\mathsf{mtag}^1_{i,t}, \mathsf{mtag}^2_{i,t}) = e((H(t)^{\mathsf{ek}_i} g^{x}_{i,t})^{\mathsf{r}_i}, w^{\frac{1}{\mathsf{r}_i}})^r = e(H(t)^{\mathsf{ek}_i}, w)^r e(g^{x_{i,t}}, w)^r$. Under the verification key $\mathsf{vk} = (w, r, \mathsf{sk}_A)$ the aggregation of the tags $\prod \sigma_{i,t}$ can be correctly verified, upon calling the $\mathcal{O}^{\mathsf{Verify}}_{\mathcal{A}}(t, \sigma_{\mathsf{t}}, \mathsf{sum_t})$ oracle.
- $\mathcal{O}^{\mathsf{Verify}}_{\mathcal{A}}(t, \sigma_{\mathsf{t}}, \mathsf{sum_t})$: $\mathcal{A}$ can query this oracle to learn the result of verification. We assume a honest verifier and this oracle makes sense, since we are in a symmetric verifications setting with a secret key. $\mathcal{B}$ returns the result of the verification since it knows the secret verification key $\mathsf{vk} = (\mathsf{vk}_1, \mathsf{vk}_2, \mathsf{vk}_3) = (w, r, \mathsf{sk}_A,)$:

$$e(H(t), \mathsf{vk}_1)^{\mathsf{vk}_2, \mathsf{vk}_3} e(g, \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{sum_t}} \stackrel{?}{=} \sigma_{\mathsf{t}}$$

When the learning phase is over, then $\mathcal{A}$ during the **Challenge phase**, chooses a set of users $\in \mathbb{S}^*$, that have not been corrupted during the **Learning phase** and chooses two time series $\mathcal{X}^*_0 = (\mathcal{U}_i, t^*, x^0_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$ and $\mathcal{X}^*_1 = (\mathcal{U}_i, t^*, x^1_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$ such that $\sum x^0_{i,t^*} = \sum x^1_{i,t^*}$ for a time interval t^* in which $\mathcal{A}$ did not query neither the $\mathcal{O}_{\mathsf{EncTag}}$ nor the $\mathcal{O}^{\mathsf{Mtag}}$ oracle and sends them to $\mathcal{O}_{\mathsf{AO}}(\mathcal{X}^0_{t^*}, \mathcal{X}^1_{t^*})$ oracle.

To simulate $\mathcal{O}_{\mathsf{AO}}(\mathcal{X}_{t^*}^0, \mathcal{X}_{t^*}^1)$ $\mathcal{B}$ queries the $\mathcal{O}_{\mathsf{AO}}^{\mathsf{PSA}}$ oracle with input $\mathcal{X}_{t^*}^0, \mathcal{X}_{t^*}^1$, which in turns flips a random coin $b \xleftarrow{\$} \{0,1\}$ and responds to $\mathcal{B}$ with the ciphertexts $\{c_{i,t^*}^b\}_{\mathcal{U}_i \in \mathbb{S}^*}$. $\mathcal{B}$ also computes the final tags:

$$\sigma_{i,t^*}^b = e(c_{i,t^*}^{\mathsf{r}_i} = (H(t^*)^{\mathsf{ek}_i} g^{x_{i,t^*}^b})^{\mathsf{r}_i}, w^{\frac{1}{\mathsf{r}_i}})^r \tag{1}$$

$$= e(H(t^*)^{\mathsf{ek}_i}, w)^r e(g^{x_{i,t^*}^b}, w^r) \tag{2}$$

Finally $\mathcal{B}$ forwards $\{c_{i,t^*}^b, \sigma_{i,t^*}^b\}$ to $\mathcal{A}$. The tag σ_{i,t^*}^b simulates perfectly the final tag of a user and the aggregation of the tags for the computation of the final proof σ_{t} correctly verifies the sum under the secret verification key $\mathsf{vk} = (\mathsf{vk}_1, \mathsf{vk}_2, \mathsf{vk}_3) = (w, r, \mathsf{sk}_A)$:

$$\prod_{i=1}^{n} \sigma_{i,t^*}^b = \prod_{i=1}^{n} e(H(t^*)^{\mathsf{ek}_i}, w)^r e(g^{x_{i,t^*}^b}, w^r) \tag{3}$$

$$= \sigma_{\mathsf{t}} = e(H(t), \mathsf{vk}_1)^{\mathsf{vk}_2, \mathsf{vk}_3} e(g, \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{sum}_{\mathsf{t}}} \tag{4}$$

If $\mathcal{A}$ has non-negligible advantage ϵ to correctly guess the bit b^* for the bit b, then $\mathcal{B}$ will break the AO game in the PSA scheme with non-negligible advantage ϵ. This contradicts the DDH assumption since the security of PSA is reduced to the DDH assumption. As such our scheme assures AO in the random oracle model under the XDH assumption, which assumes the intractability of DDH in $\mathbb{G}_1$.

8.2 Aggregate unforgeability

Theorem 2. *An adversary A who colludes with a user $\mathcal{U}_c$ in the **CRA-I** scheme has negligible probability on forging a **Type-I** $\mathsf{CR} - \mathsf{AU} - I$ forgery, under the* BCDH *assumption in the random oracle mode.*

We prove theorem 2 in three steps. First we prove the security of a base scheme (BaseLine) without any collusions in between a user and any other party. To model this scheme, an adversary $\mathcal{A}$ plays the game as described in algorithms 1 and 2 without access to the corruption oracles $\mathcal{O}^{\mathsf{Corr}_{\mathcal{A}}}, \mathcal{O}^{\mathsf{Corr}_{\mathcal{C}}}$ and $\mathcal{O}^{\mathsf{Corr}_{\mathcal{D}\mathcal{A}}}$. For the sake of clarity we call the security definition of *aggregate unforgeability* in the BaseLine scheme as BAU and the corresponding game **Game**$^{\mathsf{BAU}}$. Then we show that a **Type-I** forgery in the **CRA-I** can be transformed to a **Type-I** forgery in the BaseLine scheme and finally that a **Type-I** forgery in the **CRA-II** scheme can be transformed to a **Type-I** forgery in the BaseLine scheme, as well.

Lemma 1. *The baseline scheme guarantees* aggregate unforgeability *for **Type-I** forgeries under the* BCDH *assumption in the random oracle model.*

Proof. We will show how an adversary $\mathcal{B}$ injects the challenge of the BCDH assumption into the game that adversary $\mathcal{A}$ plays. During the setup phase $\mathcal{B}$ receives the challenge $(g, g_2, g^a, g^b, g^c, g_2^a, g_2^b)$ from $\mathcal{O}_{\mathsf{Setup}}^{\mathsf{BCDH}}$ oracle and is asked to output $e(g, g_2)^{abc}$. $\mathcal{B}$ simulates the Challenger when $\mathcal{A}$ plays the **Game**$^{\mathsf{BAU}}$ game as follows:

$\mathcal{B}$ first chooses uniformly at random secret keys $\mathsf{w}, \mathsf{r}, \{\mathsf{r}_i, \mathsf{ek}_i, \mathsf{tk}_i\}_{i=1}^n$

Learning phase

- $\mathcal{O}_{\mathsf{Setup}}$: Whenever $\mathcal{A}$ calls this oracle, $\mathcal{B}$ returns the public parameters $\mathsf{pp} = (H, e, g, g_2)$ for a hash function $H : \{0,1\}^* \to \mathbb{G}_1$, bilinear pairing e, generators g, g_2 for $\mathbb{G}_1, \mathbb{G}_2$ and the secret key of the Aggregator $\mathsf{sk}_A = \sum_{i=1}^n \mathsf{ek}_i$. $\mathcal{B}$ also sets as the secret verification key $\mathsf{vk} = (g_2^a, r, g^{b \sum_{i=1}^n \mathsf{r}_i})$ and does not share this information.
- $\mathcal{A}$ can query the random oracle H for a time interval t. In order to respond to the queries $\mathcal{B}$ constructs a list $\mathtt{H_L}\langle t : v_t, \mathsf{coin}(t), H(t)\rangle$ and responds to $\mathcal{A}$ as follows:
  - If H has been queried before at the time interval t, $\mathcal{B}$ fetches the tuple $\mathtt{H_L}(t)$ and replies to $\mathcal{A}$ with $H(t)$.
  - If t is fresh then $\mathcal{B}$ selects uniformly at random $\phi_t \in \mathbb{Z}_p$ and flips a random $\mathsf{coin}(t)$. With probability p $\mathsf{coin}(t) = 0$ and $\mathcal{B}$ appends to $\mathtt{H_L}(t) = g^{\phi_t}$. Otherwise with probability $1-p$ when $\mathsf{coin}(t) = 1$ then $\mathcal{B}$ sets $\mathtt{H_L}(t) = g^{c\phi_t}$. Finally $\mathcal{B}$ sends $\mathtt{H_L}(t)$ to $\mathcal{A}$.
  - Whenever $\mathcal{A}$ calls the $\mathcal{O}_{\mathsf{EncTag}(t,\mathsf{uid},x_{i,t})}$ oracle, $\mathcal{B}$ constructs a tuple $\mathtt{ET}\langle t, \mathsf{uid}_i, x_{i,t}, \sigma_{i,t}\rangle$. We differentiate three cases:

1. If at time interval t, $\mathcal{O}_{\mathsf{EncTag}(t,\mathsf{uid},x_{i,t})}$ has not been queried before, then $\mathcal{B}$ calls the simulated random oracle for time interval t and gets the response $H(t)$. If $\mathtt{coin}(t) = 1$ then $\mathcal{B}$ halts the simulation. Otherwise it computes the ciphertext with the secret encryption key ek_i as $c_{i,t} = H(t)^{\mathsf{ek}_i} g^{x_{i,t}} = g^{\phi_t \mathsf{ek}_i} g^{x_{i,t}}$. Finally $\mathcal{B}$ computes the metatag $\mathsf{mtag}_{i,t} = [H(t)^{\mathsf{r}_i} g^{x_{i,t}}]^{\mathsf{tk}_i}, w^{\frac{1}{\mathsf{tk}_i}}$ and forwards $c_{i,t}, \mathsf{mtag}_{i,t}$ to $\mathcal{A}$. It also updates $\mathtt{ET}$ list with the tuple: $\langle t, \mathsf{uid}_i, x_{i,t}, \sigma_{i,t} \rangle$ and sets $\Sigma_t = \Sigma_t + x_{i,t}$.
2. If there exists uid in the list $\mathtt{ET}$ for time interval t, then $\mathcal{B}$ fetches this tuple and forwards $c_{i,t}, \sigma_{i,t}$ to $\mathcal{A}$.
3. Else $\mathcal{B}$ fetches the corresponding tuple from the $\mathtt{H_L}$ list. If $\mathtt{coin}(t) = 1$ then $\mathcal{B}$ halts the simulation. Otherwise it computes the ciphertext with the secret encryption key ek_i as $c_{i,t} = H(t)^{\mathsf{ek}_i} g^{x_{i,t}} = g^{\phi_t \mathsf{ek}_i} g^{x_{i,t}}$. Finally $\mathcal{B}$ computes the metatag $\mathsf{mtag}_{i,t} = [H(t)^{\mathsf{r}_i} g^{x_{i,t}}]^{\mathsf{tk}_i}, w^{\frac{1}{\mathsf{tk}_i}}$ and forwards $c_{i,t}, \mathsf{mtag}_{i,t}$ to $\mathcal{A}$. It also updates $\mathtt{ET}$ list with the tuple: $\langle t, \mathsf{uid}_i, x_{i,t}, \sigma_{i,t} \rangle$ and sets $\Sigma_t = \Sigma_t + x_{i,t}$.

– When $\mathcal{A}$ calls the $\mathcal{O}_{\mathcal{A}}^{\mathsf{Mtag}}(\mathsf{mtag}_{i,t})$ oracle, $\mathcal{B}$ calls the simulated random oracle to get $H(t)$. If $\mathtt{coin}(t) = 0$ then $\mathcal{B}$ halts, otherwise it forwards to $\mathcal{A}$ $\sigma_{i,t} = e(H(t)^{\mathsf{r}_i}, w)^r e(g^{x_{i,t}}, w)^r$.

Challenge phase At the challenge phase $\mathcal{A}$ outputs a forgery $\mathsf{sum_t}^*, \sigma_\mathsf{t}^*$ for a time interval t^*. $\mathcal{B}$ fetches the tuple $\mathtt{H_L}(t^*)$ and:

– If $\mathtt{coin}(t^*) = 0$, then it aborts.
– Otherwise it solves the BCDH assumption by computing:

$$I = \frac{(\sigma_\mathsf{t}^*)}{e(g, \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{sum}_\mathsf{t}^*}} = \frac{e(H(t^*), \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{vk}_3} e(g, \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{sum}_\mathsf{t}^*}}{e(g, \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{sum}_\mathsf{t}^*}}$$
$$= e(H(t^*), \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{vk}_3} = e(g^{c\phi_{t^*}}, g_2^a)^{rb \sum_{i=1}^n \mathsf{r}_i}$$

Finally it outputs $I^{\frac{1}{\phi_{t^*} r \sum_{i=1}^n \mathsf{r}_i}} = e(g, g_2)^{abc}$, which is the solution to the BCDH problem.

The probabilities of $\mathcal{B}$ to not abort are $p^2(1-p)^{q_h}$ for q_h queries to the random oracle. So assuming $\mathcal{A}$ forges a **Type-I** forgery with some non-negligible probability $\epsilon'(\lambda)$, then $\Pr[\mathcal{B}^{\mathsf{BCDH}}] = p^2(1-p)^{q_h}\epsilon'(\lambda)$. As such we ended up in a contradiction assuming the hardness of the BCDH assumption and $\Pr[\mathcal{A}^{\mathsf{BAU}}] = \epsilon(\lambda)$ for some negligible function ϵ on input of the security parameter λ.

Lemma 2. *Let $\mathcal{A}$ be a probabilistic polynomial time adversary who colludes with a user $\mathcal{U}_c$ in the **CRA-I** scheme and outputs a **Type-I** forgery with non-negligible probability. Then, there is an adversary $\mathcal{B}$ that outputs a **Type-I** forgery for the* `BaseLine` *scheme with non-negligible probability.*

Proof. $\mathcal{B}$ calls the $\mathcal{O}_{CRA-I}^{\mathsf{Setup}}$ oracle which returns the public parameters $\mathsf{pp} = (H, e, g, g_2)$ and the secret key of the Aggregator sk_A. $\mathcal{B}$ relays this information to $\mathcal{A}$. Whenever $\mathcal{A}$ calls the $\mathcal{O}_{\mathcal{A}}^{\mathsf{EncTag}}(t, \mathsf{uid}, x_{i,t})$ oracle, then $\mathcal{B}$ in turn forwards the query to the $\mathcal{O}_{\mathcal{A}}^{\mathsf{EncTag}}(t, \mathsf{uid}, x_{i,t})$ oracle of the **CRA-I** game, which replies with $c_{i,t} = H(t)^{\mathsf{ek}_i} g^{x_{i,t}}, \mathsf{mtag}_{i,t} = [H(t)^{\mathsf{r}_i} g^{x_{i,t}}]^{\mathsf{tk}_i}, w^{\frac{1}{\mathsf{tk}_i}}$. Similarly $\mathcal{B}$ relays the queries to the $\mathcal{O}_{\mathcal{A}}^{\mathsf{Mtag}}(\mathsf{mtag}_{i,t})$ and forwards the response $= \sigma_{i,t} = e(H(t)^{\mathsf{r}_i}, w)^r e(g^{x_{i,t}}, w)^r$ back to $\mathcal{A}$. $\mathcal{B}$ responds to the queries for the $\mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{U}_m}}(t, \mathsf{uid} = i \in \mathbb{U})$ oracle, with $(\mathsf{r}_i, \mathsf{ek}_i, \mathsf{tk}_i, w)$. Note that for trustworthy users, $\mathcal{A}$ only learns $H(t)^{\mathsf{ek}_i} g^{x_{i,t}}, [H(t)^{\mathsf{r}_i} g^{x_{i,t}}]^{\mathsf{tk}_i}, w^{\frac{1}{\mathsf{tk}_i}}, e(H(t)^{\mathsf{r}_i}, w)^r e(g^{x_{i,t}}, w)^r$ by knowing w. Thus the secret value $x_{i,t}$ and the secret keys of the user are computationally hidden. At this point the view of $\mathcal{A}$ is consistent with the real protocol and thus does not abort the game. Eventually $\mathcal{A}$ outputs a forgery σ_t^*. $\mathcal{B}$ also outputs σ_t^* as a valid forgery.

Lemma 3. *Let $\mathcal{C}$ be a probabilistic polynomial time adversary who colludes with a user $\mathcal{U}_c$ in the **CRA-II** scheme and outputs a **Type-I** forgery with non-negligible probability. Then, there is an adversary $\mathcal{B}$ that outputs a **Type-I** forgery for the* `BaseLine` *scheme with non-negligible probability.*

Proof. The proof proceeds accordingly with the previous proof for lemma 2. $\mathcal{B}$ relays queries to $\mathcal{O}_{CRA-II}^*$ oracles, coming from $\mathcal{C}$. When $\mathcal{C}$ corrupts a user $\mathcal{U}_i \in \mathbb{S}$ then $\mathcal{B}$ forwards to $\mathcal{C}$ the secret keys $(\mathsf{r}_i, \mathsf{ek}_i, \mathsf{tk}_i, w)$. Finally the view of adversary $\mathcal{C}$ is identical with the real game without being able to distinguish since $H(t)^{\mathsf{ek}_i} g^{x_{i,t}}, [H(t)^{\mathsf{r}_i} g^{x_{i,t}}]^{\mathsf{tk}_i}, w^{\frac{1}{\mathsf{tk}_i}}, e(H(t)^{\mathsf{r}_i}, w)^r e(g^{x_{i,t}}, w)^r$ computationally hide the secret value $x_{i,t}$ and $(\mathsf{r}_i, \mathsf{ek}_i, \mathsf{tk}_i,)$ keys from uncorrupted users by an adversary $\mathcal{C}$ knowing the secret secret key r and secret keys of corrupted users.

With lemmas 1, 2, 3 we conclude the proof of theorem 2.

Theorem 3. *An adversary A has negligible probability on forging a* ***Type-I*** $\mathsf{CR}-\mathsf{AU}-II$ *forgery, under the* BCDH *assumption in the random oracle mode.*

Notice the a $\mathsf{CR}-\mathsf{AU}-II$ forgery entails collusions between a *Converter* and an Aggregator, by revealing r to the latter. Thus the proofs proceeds as with lemma 1, with the difference that $\mathcal{A}$ during the learning phase calls the $\mathcal{O}^{\mathsf{Corr}_C}$ oracle and $\mathcal{B}$ forwards to $\mathcal{A}$ the secret key r.

Proof. $\mathcal{B}$ first chooses uniformly at random secret keys $\mathsf{w}, \mathsf{r}, \{\mathsf{r}_i, \mathsf{ek}_i, \mathsf{tk}_i\}_{i=1}^n$
Learning phase

- $\mathcal{O}_{\mathsf{Setup}}$: Whenever $\mathcal{A}$ calls this oracle, $\mathcal{B}$ returns the public parameters $\mathsf{pp} = (H, e, g, g_2)$ for a hash function $H : \{0,1\}^* \to \mathbb{G}_1$, bilinear pairing e, generators g, g_2 for $\mathbb{G}_1, \mathbb{G}_2$ and the secret key of the Aggregator $\mathsf{sk}_A = \sum_{i=1}^n \mathsf{ek}_i$. $\mathcal{B}$ also sets as the secret verification key $\mathsf{vk} = (g_2^a, r, g^{b\sum_{i=1}^n \mathsf{r}_i})$ and does not share this information.
- $\mathcal{A}$ can query the random oracle H for a time interval t. In order to respond to the queries $\mathcal{B}$ constructs a list $\mathtt{H_L}\langle t : v_t, \mathsf{coin}(t), H(t)\rangle$ and responds to $\mathcal{A}$ as follows:
  - If H has been queried before at the time interval t, $\mathcal{B}$ fetches the tuple $\mathtt{H_L}(t)$ and replies to $\mathcal{A}$ with $H(t)$.
  - If t is fresh then $\mathcal{B}$ selects uniformly at random $\phi_t \in \mathbb{Z}_p$ and flips a random $\mathsf{coin}(t)$. With probability p $\mathsf{coin}(t) = 0$ and $\mathcal{B}$ appends to $\mathtt{H_L}(t) = g^{\phi_t}$. Otherwise with probability $1-p$ when $\mathsf{coin}(t) = 1$ then $\mathcal{B}$ sets $\mathtt{H_L}(t) = g^{c\phi_t}$. Finally $\mathcal{B}$ sends $\mathtt{H_L}(t)$ to $\mathcal{A}$.
  - Whenever $\mathcal{A}$ calls the $\mathcal{O}_{\mathsf{EncTag}(t,\mathsf{uid},x_{i,t})}$ oracle, $\mathcal{B}$ constructs a tuple $\mathtt{ET}\langle t, \mathsf{uid}_i, x_{i,t}, \sigma_{i,t}\rangle$. We differentiate three cases:
    1. If at time interval t, $\mathcal{O}_{\mathsf{EncTag}(t,\mathsf{uid},x_{i,t})}$ has not been queried before, then $\mathcal{B}$ calls the simulated random oracle for time interval t and gets the response $H(t)$. If $\mathtt{coin}(t) = 1$ then $\mathcal{B}$ halts the simulation. Otherwise it computes the ciphertext with the secret encryption key ek_i as $c_{i,t} = H(t)^{\mathsf{ek}_i} g^{x_{i,t}} = g^{\phi_t \mathsf{ek}_i} g^{x_{i,t}}$. Finally $\mathcal{B}$ computes the metatag $\mathsf{mtag}_{i,t} = [H(t)^{\mathsf{r}_i} g^{x_{i,t}}]^{\mathsf{tk}_i}, w^{\frac{1}{\mathsf{tk}_i}}$ and forwards $c_{i,t}, \mathsf{mtag}_{i,t}$ to $\mathcal{A}$. It also updates $\mathtt{ET}$ list with the tuple: $\langle t, \mathsf{uid}_i, x_{i,t}, \sigma_{i,t}\rangle$ and sets $\Sigma_t = \Sigma_t + x_{i,t}$.
    2. If there exists uid in the list $\mathtt{ET}$ for time interval t, then $\mathcal{B}$ fetches this tuple and forwards $c_{i,t}, \sigma_{i,t}$ to $\mathcal{A}$.
    3. Else $\mathcal{B}$ fetches the corresponding tuple from the $\mathtt{H_L}$ list. If $\mathtt{coin}(t) = 1$ then $\mathcal{B}$ halts the simulation. Otherwise it computes the ciphertext with the secret encryption key ek_i as $c_{i,t} = H(t)^{\mathsf{ek}_i} g^{x_{i,t}} = g^{\phi_t \mathsf{ek}_i} g^{x_{i,t}}$. Finally $\mathcal{B}$ computes the metatag $\mathsf{mtag}_{i,t} = [H(t)^{\mathsf{r}_i} g^{x_{i,t}}]^{\mathsf{tk}_i}, w^{\frac{1}{\mathsf{tk}_i}}$ and forwards $c_{i,t}, \mathsf{mtag}_{i,t}$ to $\mathcal{A}$. It also updates $\mathtt{ET}$ list with the tuple: $\langle t, \mathsf{uid}_i, x_{i,t}, \sigma_{i,t}\rangle$ and sets $\Sigma_t = \Sigma_t + x_{i,t}$.
- $\mathcal{B}$ forwards to $\mathcal{A}$ the secret key r while invoking the $\mathcal{O}^{\mathsf{Corr}_C}$ oracle.
- When $\mathcal{A}$ calls the $\mathcal{O}_{\mathcal{A}}^{\mathsf{Mtag}}(\mathsf{mtag}_{i,t})$ oracle, $\mathcal{B}$ calls the simulated random oracle to get $H(t)$. If $\mathtt{coin}(t) = 0$ then $\mathcal{B}$ halts, otherwise it forwards to $\mathcal{A}$ $\sigma_{i,t} = e(H(t)^{\mathsf{r}_i}, w)^r e(g^{x_{i,t}}, w)^r$.

Challenge phase At the challenge phase $\mathcal{A}$ outputs a forgery $\mathsf{sum_t}^*, \sigma_{\mathsf{t}}{}^*$ for a time interval t^*. $\mathcal{B}$ fetches the tuple $\mathtt{H_L}(t^*)$ and:

- If $\mathtt{coin}(t^*) = 0$, then it aborts.
- Otherwise it solves the BCDH assumption by computing:

$$I = \frac{(\sigma_{\mathsf{t}}{}^*)}{e(g, \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{sum_t}^*}} = \frac{e(H(t^*), \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{vk}_3} e(g, \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{sum_t}^*}}{e(g, \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{sum_t}^*}}$$
$$= e(H(t^*), \mathsf{vk}_1)^{\mathsf{vk}_2 \mathsf{vk}_3} = e(g^{c\phi_{t^*}}, g_2^a)^{rb\sum_{i=1}^n \mathsf{r}_i}$$

Finally it outputs $I^{\frac{1}{\phi_{t^*} r \sum_{i=1}^n \mathsf{r}_i}} = e(g, g_2)^{abc}$, which is the solution to the BCDH problem.

Similarly with Lemma 1 the probabilities of $\mathcal{B}$ to not abort are $p^2(1-p)^{q_h}$ for q_h queries to the random oracle. $\mathcal{A}$ outputs a **Type-I** $\mathsf{CR}-\mathsf{AU}-II$ forgery with some non-negligible probability $\epsilon'(\lambda)$, then $\Pr[\mathcal{B}^{\mathsf{BCDH}}] = p^2(1-p)^{q_h}\epsilon'(\lambda)$, which is a contradiction assuming the hardness of BCDH assumption and concludes the proof.

Theorem 4. *An adversary $\mathcal{A}$ who colludes with a user $\mathcal{U}_c$ in the **CRA-II** scheme has negligible probability on forging a **Type-II** $\mathsf{CR}-\mathsf{AU}-I,II$ forgery, under the* $\mathsf{DFAPI}-\mathsf{I}$ *assumption in the standard model.*

For the proof of the theorem 4 we first introduce the following assumption:

Definition 7. *(Dual Fixed Argument Pairing Inversion I (*$\mathsf{DFAPI}-\mathsf{I}$*) Assumption)*
Let $e(\mathbb{G}_1 \times \mathbb{G}_2) \rightarrow \mathbb{G}_T$ be a bilinear pairing, $c_1, d_1 \in \mathbb{G}_1, c_2, d_2 \in \mathbb{G}_2$ and $e(c_1, c_2) = z_1 \in \mathbb{G}_T$, $e(d_1, d_2) = z_2 \in \mathbb{G}_T$. We say that $\mathsf{FAPI}-\mathsf{I}$ *holds if the probabilities of a probabilistic polynomial time adversary $\mathcal{A}$ $\Pr[d_2 \leftarrow \mathcal{A}(d_1, z_1 \cdot z_2)]$ are negligible on input the security parameter λ.*

For the proof of the aforementioned assumption we will show how a probabilistic polynomial time adversary $\mathcal{A}$ who has non-negligible probabilities on the $\mathsf{DFAPI}-\mathsf{I}$ assumption, can be used by a probabilistic polynomial time adversary $\mathcal{B}$ to break the $\mathsf{FAPI}-\mathsf{I}$ assumption with non negligible probabilities. We denote by $\mathcal{O}^{\mathsf{FAPI}-\mathsf{I}}$ the oracle of the $\mathsf{FAPI}-\mathsf{I}$ assumption, which outputs the challenge to an adversary and by $\mathcal{O}^{\mathsf{DFAPI}-\mathsf{I}}$ the oracle of the $\mathsf{DFAPI}-\mathsf{I}$ assumption.

Proof. $\mathcal{B}$ queries the $\mathcal{O}^{\mathsf{FAPI}-\mathsf{I}}$ oracle and gets back $(e, d_1, z = e(d_1, d_2))$. When $\mathcal{A}$ asks the $\mathcal{O}^{\mathsf{DFAPI}-\mathsf{I}}$ for the public parameters of the scheme then $\mathcal{B}$ computes $y = e(r_1, r_2)e(d_1, d_2)$ by choosing $r_1 \in \mathbb{G}_1, r_2 \in \mathbb{G}_2$ and forwards it to $\mathcal{A}$. Assuming $\mathcal{A}$ breaks the $\mathsf{DFAPI}-\mathsf{I}$ assumption with non-negligible probability ϵ by outputting d_2, then $\mathcal{B}$ outputs d_2 as a solution to $\mathsf{FAPI}-\mathsf{I}$ with equivalent non-negligible probability ϵ.

Proof. We show the interaction of $\mathcal{A}$ with the oracles, who eventually in order to provide a valid forgery has to solve the $\mathsf{DFAPI}-\mathsf{I}$ assumption with non-negligible probability.

- $\mathtt{flag} = 0$.
- $\mathcal{O}_{\mathsf{Setup}}$: Whenever $\mathcal{A}$ calls this oracle, it receives the public parameters $\mathsf{pp} = (H, e, g, g_2)$ for a hash function $H : \{0,1\}^* \rightarrow \mathbb{G}_1$, bilinear pairing e, generators g, g_2 for $\mathbb{G}_1, \mathbb{G}_2$ and the secret key of the Aggregator $\mathsf{sk}_A = \sum_{i=1}^{n} \mathsf{ek}_i$. This oracle also sets as the secret verification key $\mathsf{vk} = (g_2^a, r, g^{b\sum_{i=1}^{n} \mathsf{r}_i})$ and does not share this information.
- $\mathcal{O}^{\mathsf{Corr}_C}$: This oracle forwards to $\mathcal{A}$ the secret key r while invoking the $\mathcal{O}^{\mathsf{Corr}_C}$ oracle and sets $\mathtt{flag} = 1$, to indicate a $\mathsf{CR}-\mathsf{AU}-II$ forgery.
- $\mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{U}_m}}(\mathsf{uid} = i \in \mathbb{U})$: If $\mathtt{flag} == 0$ then $\mathcal{O}^{\mathsf{Coll}_{\mathcal{A},\mathcal{U}_m}}(\mathsf{uid} = i \in \mathbb{U})$transmits the secret keys $\mathsf{ek}_i, \mathsf{tk}_i, r_i, w$. Otherwise it returns $\mathtt{null}$ to $\mathcal{A}$.
- $\mathcal{O}_{\mathcal{A}}^{\mathsf{EncTag}}(t, \mathsf{uid}, x_{i,t})$: $\mathcal{A}$ receives the ciphertext with the secret encryption key ek_i as $c_{i,t} = H(t)^{\mathsf{ek}_i} g^{x_{i,t}}$ and the metatag $\mathsf{mtag}_{i,t} = [H(t)^{\mathsf{r}_i} g^{x_{i,t}}]^{\mathsf{tk}_i}, w^{\frac{1}{\mathsf{tk}_i}}$.
- $\mathcal{O}_{\mathcal{A}}^{\mathsf{Mtag}}(\mathsf{mtag}_{i,t})$: When $\mathcal{A}$ calls the $\mathcal{O}_{\mathcal{A}}^{\mathsf{Mtag}}(\mathsf{mtag}_{i,t})$ oracle, it gets $\sigma_{i,t} = e(H(t)^{\mathsf{r}_i}, w)^r e(g^{x_{i,t}}, w)^r$.

Eventually $\mathcal{A}$ outputs a forgery for a time interval t, $\sigma_{i,t}$ for a $\mathsf{sum}_{\mathsf{t}} = s$. In case of a $\mathsf{CR}-\mathsf{AU}-I$ forgery then $\mathtt{flag} = 0$ and $\mathcal{A}$ learns $\mathsf{ek}_i, \mathsf{tk}_i, r_i, w$ for some users. In order the forgery to be valid and be accepted by the $\mathcal{O}^{\mathsf{Verify}}(t, \sigma_{\mathsf{t}}, \mathsf{sum}_{\mathsf{t}})$ oracle it ought to have the following form: $\sigma_{i,t} = e(H(t)^{\sum \mathsf{r}_i}, w)^r e(g^s, w)^r = e(H(t)^{\sum \mathsf{r}_i}, w)^r e(g^s, w^r)$ for $s = \mathsf{sum}_{\mathsf{t}}$. As such, in order $\mathcal{A}$ to compute $\sigma_{i,t}$ needs to extract r which is coupled with each tag $\prod \sigma_{i,t} = e(H(t)^{\sum \mathsf{r}_i}, w)^r e(g^{x_{i,t}}, w)^r$, which is an instance of a $\mathsf{FAPI}-\mathsf{I}$ assumption with $c_1 = H(t)^{\sum \mathsf{r}_i}, c_2 = w^r, d_1 = g^s, d_2 = w^r$ and its hardness is proved in theorem 7.

Similarly, in case the forgery is of type $\mathsf{CR}-\mathsf{AU}-II$ then $\mathtt{flag} = 1$ and $\mathcal{A}$ knows r, s and $H(t)$ from $\sigma_{i,t}$. In order the forgery to be valid $\mathcal{A}$ needs to extract w from $\sigma_{i,t} = e(H(t)^{\sum \mathsf{r}_i}, w)^r e(g^s, w)^r$. From the bilinearity the equation can be expanded as $\sigma_{i,t} = e(H(t)^{\sum \mathsf{r}_i}, w)^r e(g^{rs}, w)$. The latter is an instance of a $\mathsf{FAPI}-\mathsf{I}$ assumption with $c_1 = H(t)^{\sum \mathsf{r}_i}, c_2 = w^r, d_1 = g^{rs}, d_2 = w$.

8.3 Comparison

We present a detailed comparison with respect to the security model and the collusion resistant property of existing protocols in table 4. Protocols which assure *Aggregator obliviousness* (AO) protect individual privacy from semi-honest Aggregators. Interestingly, a recent published paper [23], necessitates the appropriate and rigorous security analysis that should be conducted for secure aggregation protocols. As already mentioned in [13] there are two flaws in [23]. By exploiting the underlying mathematical structure of the encryption algorithms a passive adversary can fully recover the plaintext values from the ciphertext of a user. Moreover collusions, which are allowed as stated in the trust model of the paper, permit users to annihilate the randomness used to evaluate multiplications over plaintexts. Apart

Protocol	Obliviousness	Verifiability	Collusions
Shi *et al.* [31]	✓	✗	✓
Joye *et al.* [22]	✓	✗	✓
Erkin *et al.* [15]	✓	✗	✓
Li *et al.* [27]	✓	✗	✓
Jawurek *et al.* [21]	✓	✓	✗
Kursawe *et al.* [24]	✓	✗	✓
Barthe *et al.* [3]	✓	✓	-
Leontiadis *et al.* [25]	✓	✗	✓
Leontiadis *et al.* [26]	✓	✓	✗
Jung *et al.* [23]	✗	✗	✗
Melis *et al.* [30]	✓	✗	✗
This work	✓	✓	✓

Table 4: Security comparison of existing protocols.

from this flawed protocol, to the best of our knowledge all the existing protocols guarantee AO in case of collusions, simply because each user does not share the encryption key with any other party in the protocol, thus the Aggregator cannot distinguish individual ciphertexts. Verifiability allows a party in the protocol to verify the correctness of the results performed by a malicious Aggregator. The protocol in [26] achieves public verifiability with the assumption of trustworthy users. As we showed in section 2.2, this protocol is insecure with respect to unforgeability as long as a malicious user colludes with a malicious Aggregator. Verifiability is also achieved in Barthe *et al.* [3] but in a different context. The authors presented a tool-assisted verifiable computations framework for program code verification for differential private computations. Thus, the notion of collusions cannot be used in program verification code for comparison with our work.

9 Conclusion

We addressed the problem of collusion resistant aggregation. Under this scenario users can collude with a malicious Aggregator, without the latter being able to forge other users' data. For our solution we initiate the study of *convertible tag*. Users first compute an authentication tag over their personal data and they forward this information along with some auxiliary data, which comprises a blinded version of their key, to an untrusted *Converter*. Finally the *Converter* transforms the tags, in order to allow an Aggregator to compute a proof of correct computations over user's data. We augment the current privacy definitions of Aggregate unforgeability with collusions between a user, the Aggregator and the *Converter*. Our protocol is provably secure and achieves constant time verification in the symmetric setting.

Bibliography

[1] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128, 2013.

[2] G. Ateniese and S. Hohenberger. Proxy re-signatures: New definitions, algorithms, and applications. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, pages 310–319, New York, NY, USA, 2005. ACM.

[3] G. Barthe, G. Danezis, B. Grégoire, C. Kunz, and S. Z. Béguelin. Verified computational differential privacy with applications to smart metering. In *2013 IEEE 26th Computer Security Foundations Symposium, New Orleans, LA, USA, June 26-28, 2013*, pages 287–301, 2013.

[4] M. Blaze, G. Bleumer, and M. Strauss. Divertible protocols and atomic proxy cryptography. In K. Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 127–144. Springer Berlin Heidelberg, 1998.

[5] A. Boldyreva, A. Palacio, and B. Warinschi. Secure proxy signature schemes for delegation of signing rights. *J. Cryptology*, 25(1):57–115, 2012.

[6] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In M. Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer Berlin Heidelberg, 2004.

[7] S. A. Brands. An efficient off-line electronic cash system based on the representation problem. Technical report, Amsterdam, The Netherlands, The Netherlands, 1993.
[8] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology – EUROCRYPT 2001*, pages 93–118. Springer Berlin Heidelberg, 2001.
[9] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In M. Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer Berlin Heidelberg, 2004.
[10] T.-H. H. Chan, E. Shi, and D. Song. Privacy-preserving stream aggregation with fault tolerance. In *Financial Cryptography*, pages 200–214, 2012.
[11] D. Chaum. Blind signatures for untraceable payments. In D. Chaum, R. Rivest, and A. Sherman, editors, *Advances in Cryptology*, pages 199–203. Springer US, 1983.
[12] D. Chaum and E. van Heyst. Group signatures. In D. Davies, editor, *Advances in Cryptology – EUROCRYPT 1991*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer Berlin Heidelberg, 1991.
[13] A. Datta and M. Joye. Cryptanalysis of a privacy-preserving aggregation protocol, 2015. http://joye.site88.net/papers/DJ__cryptanalysis.pdf.
[14] D. Derler, C. Hanser, and D. Slamanig. Privacy-enhancing proxy signatures from non-interactive anonymous credentials. In V. Atluri and G. Pernul, editors, *Data and Applications Security and Privacy XXVIII*, volume 8566 of *Lecture Notes in Computer Science*, pages 49–65. Springer Berlin Heidelberg, 2014.
[15] Z. Erkin and G. Tsudik. Private computation of spatial and temporal power consumption with smart meters. In *ACNS*, pages 561–577, 2012.
[16] J. Fan, Q. Li, and G. Cao. Privacy-aware trustworthy data aggregation in mobile sensing. In *IEEE Conference on Communications and Network Security*, 2015.
[17] G. Fuchsbauer and D. Pointcheval. Anonymous proxy signatures. In R. Ostrovsky, R. De Prisco, and I. Visconti, editors, *Security and Cryptography for Networks*, volume 5229 of *Lecture Notes in Computer Science*, pages 201–217. Springer Berlin Heidelberg, 2008.
[18] S. D. Galbraith, F. Hess, and F. Vercauteren. Aspects of pairing inversion. *IEEE Trans. of Information Theory.*, 54:5719–5728, 2008.
[19] C. Hanser and D. Slamanig. Blank digital signatures. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, pages 95–106, New York, NY, USA, 2013. ACM.
[20] C. Hanser and D. Slamanig. Warrant-hiding delegation-by-certificate proxy signature schemes. In *Progress in Cryptology - INDOCRYPT 2013 - 14th International Conference on Cryptology in India, Mumbai, India, December 7-10, 2013. Proceedings*, pages 60–77, 2013.
[21] M. Jawurek and F. Kerschbaum. Fault-tolerant privacy-preserving statistics. In *Privacy Enhancing Technologies*, pages 221–238, 2012.
[22] M. Joye and B. Libert. A scalable scheme for privacy-preserving aggregation of time-series data. In *Financial Cryptography*, 2013.
[23] T. Jung, X. Li, and M. Wan. Collusion-tolerable privacy-preserving sum and product calculation without secure channel. *IEEE Trans. Dependable Sec. Comput.*, 12(1):45–57, 2015.
[24] K. Kursawe, G. Danezis, and M. Kohlweiss. Privacy-friendly aggregation for the smart-grid. In *PETS*, pages 175–191, 2011.
[25] I. Leontiadis, K. Elkhiyaoui, and R. Molva. Private and dynamic time-series data aggregation with trust relaxation. In *Cryptology and Network Security - 13th International Conference, CANS 2014, Heraklion, Crete, Greece, October 22-24, 2014. Proceedings*, pages 305–320, 2014.
[26] I. Leontiadis, K. Elkhiyaoui, M. Önen, and R. Molva. PUDA - privacy and unforgeability for data aggregation. In *Cryptology and Network Security - 14th International Conference, CANS 2015, Marrakesh, Morocco, December 10-12, 2015, Proceedings*, pages 3–18, 2015.
[27] Q. Li and G. Cao. Efficient privacy-preserving stream aggregation in mobile sensing with low aggregation error. In *PETS*, pages 60–81, 2013.
[28] B. Libert and D. Vergnaud. Multi-use unidirectional proxy re-signatures. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, pages 511–520, New York, NY, USA, 2008. ACM.

[29] M. Mambo, K. Usuda, and E. Okamoto. Proxy signatures for delegating signing operation. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, CCS '96, pages 48–57, New York, NY, USA, 1996. ACM.

[30] L. Melis, G. Danezis, and E. D. Cristofaro. Efficient private statistics with succinct sketches. *CoRR*, abs/1508.06110, 2015.

[31] E. Shi, T.-H. H. Chan, E. G. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *NDSS*, 2011.