A Secure Bandwidth-Efficient Treatment for Dropout-Resistant Time-Series Data Aggregation

Reyhaneh Rabaninejad
Tampere University
Tampere, Finland
reyhaneh.rabbaninejad@tuni.fi

Alexandros Bakas
Tampere University, and
Nokia-Bell Labs, Finland
alexandros.bakas@tuni.fi

Eugene Frimpong
Tampere University
Tampere, Finland
eugene.frimpong@tuni.fi

Antonis Michalas
Tampere University, Finland and
RISE Research Institutes of Sweden
antonios.michalas@tuni.fi

***Abstract*—Aggregate statistics derived from time-series data collected by individual users are extremely beneficial in diverse fields, such as e-health applications, IoT-based smart metering networks, and federated learning systems. Since user data are privacy-sensitive in many cases, the untrusted aggregator may only infer the aggregation without breaching individual privacy. To this aim, secure aggregation techniques have been extensively researched over the past years. However, most existing schemes suffer either from high communication overhead when users join and leave, or cannot tolerate node dropouts. In this paper, we propose a dropout-resistant bandwidth-efficient time-series data aggregation. The proposed scheme does not incur any interaction among users, involving a solo round of user→aggregator communication exclusively. Additionally, it does not trigger a re-generation of private keys when users join and leave. Moreover, the aggregator is able to output the aggregate value by employing the re-encrypt capability acquired during a one-time setup phase, notwithstanding the number of nodes in the ecosystem that partake in the data collection of a certain epoch. Dropout-resistancy, trust-less key management, low-bandwidth and non-interactive nature of our construction make it ideal for many rapid-changing distributed real-world networks. Other than bandwidth efficiency, our scheme has also demonstrated efficiency in terms of computation overhead.**

***Index Terms*—privacy-preserving aggregation, time-series data, proxy re-encryption, dropout-tolerant, dynamic groups, bandwidth-efficient**

I. Introduction

Data and analytics constantly evolve shaped by the ever-improving complexity of technology. The industry is experiencing continuous digital transition, resulting in a constantly increasing volume of generated data. Every organization today relies on small and big data obtained through networks and distributed applications, as well as useful information inferred from them. Data analytics are extremely beneficial in comprehending diseases and the effect of medicines in medical IoT-based applications or target audiences and their preferences in recommendation systems or even smart energy consumption meters in smart cities. This rapid growth in the use of data analytics inevitably raises privacy concerns about how individual user data are used [1], [2]. For example, according to a study [3], 19 out of 24 general-purpose mobile health applications shared user data with more than 50 different organizations, the majority of which were data analytics firms. This, along with other previously known privacy breaches are extremely alarming, given the significance of statistical data analytics. Secure aggregation techniques fall under privacy-preserving data analytics category and, hence, have been extensively researched over the last decade (e.g., [4], [5], [6]). The fundamental setup for such protocols consists of multiple independent parties collaborating with an untrusted aggregator, whose purpose is to compute the sum of different party inputs without exposing any information about the private inputs of individual parties with the exception of the aggregated value itself. However, only a few of the proposed aggregation solutions are resilient to user failure and compromise, and can efficiently support dynamic group joins and leaves. In these *static* aggregation schemes, even when a single user fails to submit within a specific aggregation epoch, the protocol aborts the input and the aggregator is unable to deduce the sum value. This can prove to be a serious obstacle in realistic scenarios, where user failures may be inevitable. E.g., in a distributed IoT-based environment, node failures are quite likely to happen either due to malfunctioning or Denial-of-Service attack placed by malicious nodes. Failure resilience and dynamic user management are considered as important open-research challenges in secure aggregation.

Contributions. Having in mind that security and failure resilience are two equilateral problems in aggregation systems, this paper proposes a secure dynamic construction for time-series data aggregation based on the promising concept of *Proxy re-Encryption*. Our main goals are to design a mechanism for aggregating data in a privacy-preserving way, supporting an efficient dynamic user management, also to provide a *new* approach towards addressing the data aggregation problem based on proxy re-encryption techniques for the first time. Our construction utilizes additively homomorphic proxy re-encryption and proxy re-signature to *translate* all collected ciphertexts/tags on different secret keys to second-level ciphertexts/tags on a same secret key. This enables the protocol to tackle dynamic changes in the population of users (i.e., joins, leaves or user failures) in each round without triggering a key update. Our contributions are summarized as follows:

C1. Dynamic construction: The core contribution of this work is the design of a secure time-series data aggregation based on proxy re-encryption, which simultaneously supports efficient dynamic joins/leaves and user dropouts.

The scheme also enables us to verify the authenticity of the final aggregation result with the aid of tags generated based on proxy re-signature. The scheme is proven secure under the Aggregator Obliviousness security definition.

C2. Bandwidth-efficiency: The proposed dropout-resistant data aggregation construction has *a fixed communication overhead*. Specifically, although dropout-resistant, our scheme does not require any interaction among users and it only involves a solo round of user→aggregator communication. That is, despite unknown user dropouts, the aggregator is still able to output the aggregate value by employing the re-encrypt capability acquired during a *one-time* setup phase. Moreover, in case of user join/ leave, only the keys related to the moving node need to be updated, while no key update is triggered for the rest of users. This result is quite remarkable considering that bandwidth is typically more expensive than computational power in most applications including IoT and mobile devices, making our scheme easy to scale. Table I provides a communication comparison between this work and existing dynamic data aggregation frameworks.

C3. Simple trust-less key management: In our construction, users generate their keys locally and independently. As a result, no trusted key manager is required. Besides, users do not need separate keys for encryption and tag generation, resulting in a simplified key management.

C4. Efficiency: From the performance standpoint, the experimental results demonstrate that the proposed construction is very efficient with results comparable to those published in existing schemes. We provide experimental results with a testbed consisting of a standard desktop machine and a resource-constrained device to demonstrate how our solution may be applicable to the real-world.

TABLE I: Comparison between dynamic constructions. n, $U \rightarrow A$, and $U \Leftrightarrow A$ respectively denote the number of users, user-to-aggregator uni-directional communication, and round-trip user-aggregator communication.

Scheme	Avg comm. per user	Comm. model
[7]	$o(\log n)$	$U \rightarrow A$
[8]	$o(d)^{\dagger}$	$U \Leftrightarrow A^{\ddagger}$
[9]	$o(n)$	$U \Leftrightarrow A$
[10]	$o(n)$	$U \Leftrightarrow A$
this paper	$o(1)$	$U \rightarrow A$

† d is a system parameter defined in [8].
‡ To tackle user dropouts, [8] needs bidirectional communication.

II. Related Work

Secure Data Aggregation. In [11], authors propose and design a protocol including private and unforgeable data aggregation (PUDA) for multiple independent users. Unlike existing data aggregation works that primarily focused on data confidentiality between end-users and the aggregator, PUDA ensures that the aggregated result is also unforgeable. To this end, the authors define new security requirements revolving around an untrusted aggregator. In an effort to increase the security requirements of [11], authors in [6] recently propose a data aggregation protocol that considers both end-users and aggregators as potentially malicious parties. The protocol can be considered as an extension of [11] plus the design of an Oblivious Programmable Pseudo-Random Function (OPPRF). Notwithstanding the progress, [6] suffers from additional communication overhead between users and the aggregator. The scheme also introduces a preliminary stage consisting of the OPPRF protocol, which in itself is a *very expensive* function. Besides, this solution builds upon the *static* scheme of [11], where user dropouts induce protocol aborts and impede the aggregator from deducing the aggregate value. Actually, contrary to what the authors have claimed, the scheme in [6] does not tackle malicious users, as they can potentially place *Denial-of-Service* attacks by refusing to respond within a certain epoch. This is an inherent obstacle in all static aggregation systems, which we call it *all-or-nothing* property: the protocol will output the aggregate value within an epoch, if and only if *all* users submit data. In addition, despite the arguments expressed in the threat model regarding malicious parties that may send bogus inputs, this scheme only includes threshold checking and does not handle data pollution attacks where users submit flawed or nonsense input values to the aggregator.

Dropout-Robust Data Aggregation. Chan et al. [7] introduced an aggregation scheme resilient to user failure and compromise by building a binary interval tree over n users, enabling the aggregator to approximate the sum of contiguous intervals of users based on tree interval nodes. The scheme incurs a logarithmic communication cost in the number of users for achieving fault tolerance. Additionally, it suffers from a high-aggregation error. The authors in [10] build an aggregation scheme for privacy-preserving machine learning. To deal with user dropouts, the scheme requires an all-to-all communication, with each individual securely sharing their seed with others in a way that the shares of dropped users can be reconstructed at the aggregator using alive users' inputs. This all-to-all secret sharing induces quadratic $O(n^2)$ communication costs, making it impractical for many applications. The authors in [8] presented a design by leveraging a ring-based interleaved grouping technique dividing users into disjoint subsets. This result ensures efficiency in dynamic joins and leaves, since by joining or leaving a node, only the nodes in the same cluster with the moving node need to update their keys. However, the scheme does not tackle node dropouts and relies on a fully trusted key dealer, aware of the secret keys of all users. In an effort to eliminate reliance in a trusted key dealer and tolerate node dropouts, Leontiadis et al. [9] proposed a secure data aggregation protocol improving [5]. According to this solution, with each interval, users await to receive an obfuscated value $pk_{A,t} = H(t)^{sk_A}$ published by the aggregator, where sk_A is the aggregator's secret key. Each user then computes auxiliary information $aux_{i,t}$ based on $pk_{A,t}$ and sends it to a semi-trusted collector, while submitting

the ciphertext $c_{i,t}$ to the aggregator. The aggregated auxiliary value aux_t generated by the collector enables the aggregator to unmask the aggregated ciphertext and retrieve the sum value for any number of users who have participated in that certain time interval, thus making the scheme dropout-tolerant. However, a malicious user u_i can perform Denial-of-Service attack in the protocol by sending $aux_{i,t}$ to the collector and *not* submitting ciphertext $c_{i,t}$ to the aggregator. This disables the aggregator to extract correct aggregations. Furthermore, at each epoch, the server has to communicate the obfuscated value to all users. This yields significant communication latency to the protocol and makes the overall communication cost linear regarding the number of users. In addition, the scheme does not provide aggregate unforgeability as introduced in [11]. Considering that bandwidth is typically much more expensive than computational power in most applications including IoT and mobile devices, designing communication-efficient constructions is believed to be a major challenge for dropout-resistant data aggregation schemes to scale out.

III. Preliminaries

Bilinear Maps: Let $\mathbb{G}$ and $\mathbb{G}_{\mathbb{T}}$ be two multiplicative cyclic groups of prime order q, g be a generator of $\mathbb{G}$ and e be a bilinear map where $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_{\mathbb{T}}$. The bilinear map e is a function with the following properties: (1) *Bilinearity*: $\forall u, v \in \mathbb{G}$ and $a, b \in Z_q$, $e(u^a, v^b) = e(u, v)^{ab}$. (2) *Non-degeneracy*: $e(g, g) \neq 1$, where 1 denotes the identity element of $\mathbb{G}_{\mathbb{T}}$. (3) *Computability*: An efficient algorithm computes $e(u, v)$ for $u, v \in \mathbb{G}$.

Proxy re-Encryption and Proxy re-Signature: Blaze et al. [12] developed the concept of "atomic proxy cryptography". Here, a semi-trusted proxy $\mathcal{P}$ acts as a translator between Alice (delegator) and Bob (delegatee) and computes a function that *translates* ciphertext/signature for Alice into ciphertext/signature for Bob on the same message. The proxy, however, does not acquire information neither about the underlying plaintext nor any encrypt/sign key and cannot sign or encrypt arbitrary messages on behalf of either Alice or Bob. These primitives were later followed up by Ateniese et al. [13]. A proxy re-encryption scheme is a tuple of algorithms $(\mathsf{KeyGen}, \mathsf{reKey}, \mathsf{Enc}, \mathsf{reEnc}, \mathsf{Dec})$, where:

- $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$: form the standard key generation, encryption and decryption algorithms in the same way as an ordinary encryption scheme.
- reKey: On input (pk_A, sk_A, pk_B, sk_B), the algorithm outputs a re-encryption key $rk_{A \rightarrow B}$ for the proxy.
- reEnc: the proxy given a re-encryption key $rk_{A \rightarrow B}$, *translates* a valid encryption from user A on message m, $\mathsf{Enc}_A(m)$, into a valid encryption from user B on the *same message*, $\mathsf{Enc}_B(m)$.

Correctness for a proxy re-encryption scheme requires that (1) $\mathsf{Dec}(sk_A, \mathsf{Enc}(pk_A, m))) = m$, and (2) $\mathsf{Dec}(sk_B, \mathsf{reEnc}(rk_{A \rightarrow B}, \mathsf{Enc}(pk_A, m))) = m$.

Similarly, a proxy re-signature scheme is a tuple of algorithms $(\mathsf{KeyGen}, \mathsf{reKey}, \mathsf{Sign}, \mathsf{reSign}, \mathsf{Verify})$, where:

- $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$: form the standard key generation, signing and verification algorithms as with an ordinary signature scheme.
- reKey: On input (pk_A, sk_A, pk_B, sk_B), the algorithm outputs a re-signature key $rk_{A \rightarrow B}$ for the proxy.
- reSign: the proxy given a re-signature key $rk_{A \rightarrow B}$, *translates* a valid signature from user A on message m, $\mathsf{Sign}_A(m)$, into a valid signature from user B on the same message, $\mathsf{Sign}_B(m)$.

Correctness for a proxy re-signature scheme requires that (1) $\mathsf{Verify}(m, pk_A, \mathsf{Sign}(sk_A, m))) = 1$, and (2) $\mathsf{Verify}(m, pk_B, \mathsf{reSign}(rk_{A \rightarrow B}, \mathsf{Sign}(sk_A, m))) = 1$. $\mathsf{Enc}_A(m)/\mathsf{Sign}_A(m)$ and $\mathsf{Enc}_B(m)/\mathsf{Sign}_B(m)$ are called *first-level* and *second-level* encryption/signature, respectively [14].

IV. System Model

In this section, we provide a brief description of the system model for the proposed scheme. Consider a case in which an untrusted aggregator wishes to compute the aggregate sum of some users' private time-series data sensed at equally spaced time intervals, without jeopardizing the privacy of individual data. Our setup consists of four main entities including: Users ($\mathcal{U}$), Proxy ($\mathcal{P}$), Aggregator ($\mathcal{A}$), and Data Analyst ($\mathcal{DA}$).

1) **Users**: Let $\mathcal{U} = \{u_1, \ldots, u_n\}$ be a set of users or edge devices deployed to collect data from the environment. Once data has been generated, each u_j computes an encryption of the data and a verification tag, and sends it to proxy $\mathcal{P}$.

2) **Proxy**: Let $\mathcal{P}$ be an entity to collect data received from multiple users in each time interval. The proxy is responsible for computing a second-level ciphertext/tag for each ciphertext/tag received from individual users. The proxy forwards set of second-level ciphertexts/tags to the aggregator.

3) **Aggregator**: Let $\mathcal{A}$ be an aggregator deployed in our environment to aggregate data received from $\mathcal{P}$. $\mathcal{A}$ is responsible for computing the sum and forwards it to the data analyst. In our proposed protocol, we assume that both $\mathcal{A}$ and $\mathcal{P}$ are more powerful and resourceful devices than edge users.

4) **Data Analyst**: We consider $\mathcal{DA}$ as the final recipient of the aggregated data and verification tag. The data analyst is able to verify the authenticity of the aggregated data value with the aid of the aggregated tag received from $\mathcal{A}$.

V. Proposed Construction for Dropout-Resistant Time-Series Data Aggregation

Now we propose our dropout-resistant data aggregation construction utilizing additive-homomorphic proxy re-encryption and re-signature techniques in [13]. In the proposed scheme, users do not need separate keys for encryption and tag generation. Hence, key management is simplified, as opposed to the existing verifiable data aggregation schemes. Besides, the requirement for a trusted key manager is eliminated. Intuitively, we use proxy re-encryption and proxy re-signature to translate all collected ciphertexts/tags to second-level ciphertexts/tags with the same secret key possessed by $\mathcal{A}$. This enables efficient support for dynamic changes in the population of users (i.e. user joins, leaves, or failures).

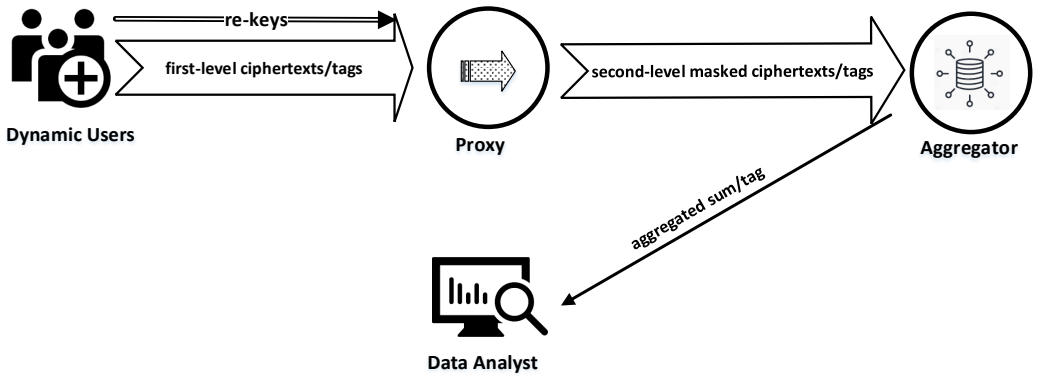

Fig. 1: Overview of our dynamic data aggregation scheme. Duplex arrow (re-key) represent secure channel.

In compliance with what is described in section IV, the scheme is executed between a set of n users sending first-level ciphertexts/tags to proxy $\mathcal{P}$. After computing reProcess on the received values, $\mathcal{P}$ forwards all generated second-level ciphertexts/tags to aggregator $\mathcal{A}$. The final aggregated result is sent to a data analyst $\mathcal{DA}$. The overview of our construction is presented in Figure 1. The proposed scheme consists of seven algorithms described in Algorithm 1: Setup, KeyGen, reKey, Process, reProcess, Extract, and Verify.

The setup algorithm takes security parameter λ as input and defines the bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_{\mathbb{T}}$ with $\mathbb{G}$ and $\mathbb{G}_{\mathbb{T}}$ as groups of prime order $q = \Theta(2^{\lambda})$, g as generator of $\mathbb{G}$, and h is a member of $\mathbb{G}_{\mathbb{T}}$. The assumption here is that $\mathbb{G}_{\mathbb{T}}$ is a group with an easy Discrete Logarithm (DL) subgroup as defined in [15], where Discrete Logarithm Problem (DLP) is easy only in base h. In KeyGen, without coordination by a trusted key manager, each user in the group $\mathcal{U}$ and aggregator $\mathcal{A}$ generate their private keys and following $\mathcal{A}$ publishes its public key. In reKey algorithm, each user $u_i \in \mathcal{U}$ inputs $\mathcal{A}$'s public key and non-interactively generates re-key $rk_{i \rightarrow A}$ on base g, and sends it to $\mathcal{P}$ via a secure (private and authenticated) channel.

At time slot t, each user u_i generates sensitive data $x_{i,t}$ which is an elements from $\mathbb{Z}_q$ and executes Process algorithm to compute ciphertext and tag on $x_{i,t}$. Following it forwards them to $\mathcal{P}$. At time slot t, in reProcess algorithm, $\mathcal{P}$ waits until a specified timeout and considers all users that did not respond before the timeout to have dropped out of the protocol in that specific epoch. Next, $\mathcal{P}$ translates the received first-level ciphertexts/tags to second-level ones using the private re-keys. It then forwards translated values to $\mathcal{A}$. Since $\mathcal{A}$ holds a delegatee role in this scheme, it can retrieve individual data from second-level ciphertexts with the aid of its private key. However, this is tackled due to the *random masking* applied by $\mathcal{P}$ such that it preserves individual users' private data, but neutralized when all ciphertexts are aggregated. Besides, since all translated ciphertexts/tags are under the same secret key, $\mathcal{A}$ is able to output the correct aggregation value, notwithstanding the number of nodes in the ecosystem that partake in the data collection of a certain epoch.

The aggregation of data is performed in Extract algorithm, where $\mathcal{A}$ aggregates second-level ciphertexts/tags received from $\mathcal{P}$. It then inputs aggregated ciphertext c_t to the decryption algorithm of proxy re-encryption to compute v_t, extracts the sum value by feeding v_t to Solve algorithm that solves DLP in base h and sends the sum and aggregated tag to $\mathcal{DA}$ for correctness validation of the result in Verify algorithm.

VI. Security Analysis

A. Threat Model

– Users: We assume a dynamic group of users in the sense that not all users are obliged to send their inputs in each epoch t. Users are assumed to be honest-but-curious in the sense that they submit correct non-polluted inputs to the aggregation protocol. We finally assume that in each epoch t, at least three users provide their inputs and at least two are fully honest. This is a valid assumption, since if only one user u_n is honest, then an adversary could trivially retrieve their input by solving $x_{n,t} = \sum_1^n x_i - \sum_1^{n-1} x_i$.

– Proxy Server: In compliance with [12], we assume the existence of a semi-trusted proxy server that receives first-level ciphertexts from the users and re-encrypts to generate second-level ciphertexts. Finally, the proxy forwards second-level ciphertexts to the aggregator.

– Aggregator: The aggregator is an un-trusted entity that receives second-level ciphertexts from the proxy, aggregates them and retrieves the sum of all inputs by users. Besides, we presume that while users in $\mathcal{U}$ may collude with either aggregator or proxy by exposing their private keys to them, the aggregator and proxy do not collude with each other.

B. Proofs of Security

Similarly to [4], we start by examining the property of *Aggregator Obliviousness* (AO). AO ensures that adversary $\mathcal{ADV}$ learns **nothing more** than the sum of the user data during protocol execution.

Proposition 1 (AO)**.** *Assuming that the aggregator is observed by a probabilistic polynomial time adversary ($\mathcal{ADV}$), then at*

Algorithm 1 The proposed dynamic data aggregation scheme

Setup

Consider the map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_{\mathbb{T}}$ with $\mathbb{G}$ and $\mathbb{G}_{\mathbb{T}}$ as groups of prime order $q = \Theta(2^{\lambda})$, and g as generator of $\mathbb{G}$, and h is a member of $\mathbb{G}_{\mathbb{T}}$. Also we define $Z = e(g, g) \in \mathbb{G}_{\mathbb{T}}$. The public parameters are $\mathsf{pp} = (e, q, \mathbb{G}, \mathbb{G}_{\mathbb{T}}, g, h, Z, H)$, where $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$ is a secure hash function. Also, the group of n users is denoted by $\mathcal{U} = \{u_1, \ldots, u_n\}$.

KeyGen

Without any coordination by a trusted key manager, each user u_i in group $\mathcal{U} = [1, n]$ samples random $k_i \xleftarrow{\$} \mathbb{Z}_q$ and sets its private key as $sk_i = k_i$.

Also, $\mathcal{A}$ samples random $k \xleftarrow{\$} \mathbb{Z}_q$ as its private key and publishes $pk_A = g^k$ publicly.

reKey

each user u_i in group $\mathcal{U} = [1, n]$ takes $\mathcal{A}$'s public key and generates its re-key $rk_{i \rightarrow A} = (g^k)^{1/k_i} = g^{k/k_i}$.

u_i sends its re-key $rk_{i \rightarrow A}$ to proxy $\mathcal{P}$ via *a secure channel*.

Process

At each time interval t, every user u_i generates sensitive data $x_{i,t}$ and computes first-level ciphertext $c_{i,t}$ and first-level tag $\sigma_{i,t} = (s_{i,t}, r_{i,t})$ as follows:

samples $e_{i,t} \xleftarrow{\$} \mathbb{Z}_q$ and sets $r_{i,t} = g^{e_{i,t}}$.

computes $c_{i,t} = (g^{k_i \cdot H(t \| e_{i,t})}, h^{x_{i,t}} \cdot Z^{H(t \| e_{i,t})})$.

computes $s_{i,t} = k_i(x_{i,t} + e_{i,t} + H(t \parallel r_{i,t}))$.

sends first-level ciphertext $c_{i,t} = (c_{i,t}^{(1)}, c_{i,t}^{(2)})$ and first-level tag $\sigma_{i,t} = (s_{i,t}, r_{i,t})$ to $\mathcal{P}$.

reProcess

At each time interval t, $\mathcal{P}$ collects data received from subgroup of users $\mathcal{U}_t \subseteq \mathcal{U}$ who have participated in that round, such that $|\mathcal{U}_t| > 2$. Next, it generates a set of random values $r_i \xleftarrow{\$} \mathbb{Z}_q$ with the condition that $\sum_{i \in \mathcal{U}_t} r_i = 0$.

for all $i \in \mathcal{U}_t$ at each time interval t, $\mathcal{P}$ **do**

inputs re-key $rk_{i \rightarrow A}$, first-level ciphertext $c_{i,t}$, and first-level tag $\sigma_{i,t}$.

computes second-level ciphertext $c'_{i,t} = (c_{i,t}^{'(1)}, c_{i,t}^{'(2)})$:

$c_{i,t}^{'(1)} = e(c_{i,t}^{(1)}, rk_{i \rightarrow A}) = e(g^{k_i \cdot H(t \| e_{i,t})}, g^{k/k_i}) = Z^{k \cdot H(t \| e_{i,t})}$

$c_{i,t}^{'(2)} = c_{i,t}^{(2)} \cdot h^{r_i} = h^{x_{i,t} + r_i} \cdot Z^{H(t \| e_{i,t})}$ ▷ randomly mask individual data

computes second-level tag $\sigma'_{i,t} = (s'_{i,t}, r_{i,t})$:

$s'_{i,t} = (rk_{i \rightarrow A})^{s_{i,t}} = (g^{k/k_i})^{k_i(x_{i,t} + e_{i,t} + H(t \| r_{i,t}))} = g^{k(x_{i,t} + e_{i,t} + H(t \| r_{i,t}))}$.

end for

$\mathcal{P}$ sends second-level ciphertexts/tags $\{c'_{i,t}, \sigma'_{i,t}\}_{i \in U_t}$ to $\mathcal{A}$.

Extract

Set $c_t^{(1)} \leftarrow 1$, $c_t^{(2)} \leftarrow 1$, and $\sigma_t \leftarrow 1$

for all $i \in \mathcal{U}_t$ at each time interval t, $\mathcal{A}$ **do**

inputs second-level ciphertext $c'_{i,t} = (c_{i,t}^{'(1)}, c_{i,t}^{'(2)})$ and second-level tag $\sigma'_{i,t} = (s'_{i,t}, r_{i,t})$.

aggregates the ciphertexts as $c_t^{(l)} \leftarrow c_t^{(l)} \times c_{i,t}^{'(l)}$, $l = 1, 2$

aggregates the tags as $\sigma_t \leftarrow \sigma_t \times s'_{i,t}$.

end for

computes $v_t = \frac{c_t^{(2)}}{\left(c_t^{(1)}\right)^{1/k}}$ ▷ decryption algorithm of proxy re-encryption

Finally, $\mathcal{A}$ computes the sum value $sum_t \leftarrow \mathsf{Solve}(\mathsf{pp}, v_t)$ and sends $(sum_t, \sigma_t, \{r_{i,t}\}_{i \in \mathcal{U}_t})$ to $\mathcal{DA}$.

Verify

$\mathcal{DA}$ verifies if $sum_t \stackrel{?}{=} \sum_{i \in \mathcal{U}_t} x_{i,t}$ by checking the following equation:

$e(g, \sigma_t) \stackrel{?}{=} e(pk_A, g^{sum_t} \prod_{i \in \mathcal{U}_t} (r_{i,t} g^{H(t \| r_{i,t})}))$

each time interval t, $\mathcal{ADV}$ only learns the sum of user inputs $x_{i,t}$ and nothing else. Also, if aggregator colludes with an arbitrary set of users $\mathcal{U}_c$, it will be able to learn nothing but the sum of honest users inputs (i.e., $\sum_{i \in \mathcal{U} - \mathcal{U}_c} x_{i,t}$).

Proof. For a complete proof of proposition 1, we consider two distinct cases; First we examine what can $\mathcal{ADV}$ learn from looking at the re-processed ciphertexts received from the proxy. Then, we rely on a hybrid argument to prove the security of the computations deployed on the aggregator's side.

C1: Received Ciphertexts. Each ciphertext received from the proxy at each time interval t, is of the form $c'_{i,t} = (c_{i,t}^{'(1)}, c_{i,t}^{'(2)})$, where $c_{i,t}^{'(1)} = e(g, g)^{kH(t \| e_{i,t})} = Z^{kH(t \| e_{i,t})}$ and for the second component we get $c_{i,t}^{'(2)} = h^{x_{i,t} + r_i} \cdot Z^{H(t \| e_{i,t})}$. We will examine each component separately:

- Since we have assumed that the DLP is hard in Z, $\mathcal{ADV}$ can only break $c_{i,t}^{'(1)}$ with negligible probability.
- Our construction is feasible based on the assumption that the DLP is easy in base h. Hence, when $\mathcal{ADV}$ sees $c_{i,t}^{'(2)}$, they can trivially recover the exponent $x_{i,t} + r_i$. However, since the aggeregator cannot collude with the proxy (by assumption), it is impossible to know the value of r_i and hence, has zero advantage in recovering $x_{i,t}$.

C2: Secure Computation on second-level ciphertexts. After the aggregator has received all second-level ciphertexts, they aggregate them by simply multiplying them. It suffices to show that for each pair of received ciphertexts, the multiplication does not leak any information that is not already known to the aggregator - and hence to $\mathcal{ADV}$. W.l.o.g. we assume that at some point in the time interval t, the aggregator receives second-level ciphertexts of the users u_i and u_j. That is $c'_{i,t} = (c_{i,t}^{'(1)}, c_{i,t}^{'(2)})$ and $c'_{j,t} = (c_{j,t}^{'(1)}, c_{j,t}^{'(2)})$. For each received pair, the aggregator performs two multiplications. We examine each multiplication separately:

- Multiplying the first components (i.e. $\left(c_{i,t}^{'(1)} \cdot c_{j,t}^{'(1)}\right)$) yields $\left[Z^{k\left[\sum_{\ell=1}^{\ell=2} H(t \| e_{\ell,t})\right]}\right]$ and, as already stated, under the assumption that the DLP is hard in Z, no information can be leaked with all but negligible probability.
- The multiplication of the second components yields $h^{x_{i,t} + r_i} Z^{H(t \| e_{i,t})} \cdot h^{x_{j,t} + r_j} Z^{H(t \| e_{j,t})} = h^{x_{i,t} + x_{j,t} + r_i + r_j} Z^{H(t \| e_{i,t}) + H(t \| e_{j,t})}$. However, similarly to **C1**, breaking it would yield $\sum x_i + \sum r_i$. Since, by assumption, the proxy cannot collude with the

aggregator, the r_i's remain private and, therefore, no information can be leaked from multiplying the second components either.

□

Proposition 2, proves *Proxy Obliviousness* (PO) to guarantee privacy against proxy server.

Proposition 2 (PO)**.** *Let $\mathcal{ADV}$ be a PPT adversary that observes the semi-trusted Proxy Server. Then, $\mathcal{ADV}$ cannot infer any information about the data of individual users either from the first-level ciphertexts it receives directly from the users or by computations on the proxy side. Also, if the proxy colludes with an arbitrary set of users $\mathcal{U}_c$, it will not be able to learn additional information about the private data of honest users in set $\mathcal{U} - \mathcal{U}_c$.*

Proof. Similarly to Proposition 1, we examine two distinct cases. First, we will examine what information can $\mathcal{ADV}$ infer by looking at first-level ciphertexts. Then, we will prove the security of the computations from the aspect of the proxy.

C1: Received Ciphertexts. At each epoch time t, the proxy receives a number of first-level ciphertexts of the form $c_{i,t} = (g^{k_i \cdot H(t \| e_{i,t})}, h^{x_{i,t}} \cdot Z^{H(t \| e_{i,t})})$. As per our assumption, the DLP is hard in base g and, hence, no information can be leaked from the first component of the message $g^{k_i \cdot H(t \| e_{i,t})}$. Concerning the second component $h^{x_{i,t}} \cdot Z^{H(t \| e_{i,t})})$, while solving the DLP in base h is easy, the values ei, t are private since they are generated locally on the users' side. Hence, no information can be gained from the second component as well. As a result, the observation of first-level ciphertexts does not leak anything to $\mathcal{ADV}$.

C2: Secure Computation on first-level ciphertexts. To prove the security of the computations on the proxy side, we will compare the distribution of the real outputs to an ideal model, in which users provide input to a fully trusted entity (a simulator $\mathcal{S}$) that computes second-level ciphertexts and outputs the result. Our goal is to prove that the two distributions are indistinguishable. To do so, we will prove that the following hybrids are indistinguishable:

Hybrid 0:	This is the real world.
Hybrid 1:	Like Hybrid 0 but $\mathcal{S}$ now randomly generates the random values r_i such that $r_n = -\sum_{i=1}^{n-1} r_i$, where n is the index of the last sampled value.

However, it is trivial to prove that Hybrid 0 and Hybrid 1 are indistinguishable, as the original set or the r_i values used in the real world are also randomly sampled. Hence, we conclude that no novel information can be leaked from re-processing first-level ciphertexts. □

VII. Experiments

In this section, we provide a comprehensive performance evaluation of our proposed scheme. For these experiments, our testbed consisted of the following commercially available devices: (1) **Users (Device A)**: An nRF25840-dk[1] board built on a 64 MHz Cortex-M4 SoC with 1 MB Flash and 256 KB RAM. To implement the proposed protocols, we installed Zephyr RTOS[2] on the board and utilized the built-in c25519 cryptographic library to design the cryptographic components. (2) **Aggregator/Proxy (Device B)**: An Intel Core i7 Desktop with 16GB RAM and an 8-core 3.60GHz CPU. We installed Ubuntu 20.04 on this machine and utilized the PBC cryptographic library[3] to implement the proposed scheme.

Evaluation at User Side: In an effort to show the real-world applicability of the proposed scheme, we evaluated the performance of user-specific functions on a resource-constrained device (Device A). More precisely, we measured the execution time of the reKey and Process functions. Prior works and experiments have shown that the performance of any scheme on a resource-constrained device correlates directly to the cost of exponentiation and Elliptic Curve (EC) point multiplications. To this end, we equate the computational cost of one exponentiation to the cost of executing the reKey function. On average, it took 0.742 seconds to execute reKey function, while the time taken to execute the Process function was 3.513 seconds on Device A.

Evaluation on the Aggregator/Proxy Side: We now evaluate the performance of the Process, reProcess, and Extract functions by measuring their execution times on the Proxy and Aggregator. Aggregator and Proxy entities are run on the same device for these evaluations (Device B). We iterated the execution of each function 50 times with a varying number of users ranging from 10 to 1500. For 10 users, it took the device 0.034 seconds to execute the Process function, 0.018 seconds to execute the reProcess function and 0.0001 seconds to execute the Extract function. For 1500 users, it took 5.06 seconds to execute the Process function, 2.68 seconds to execute the reProcess function, and 0.02 seconds for the Extract function (Figure 2).

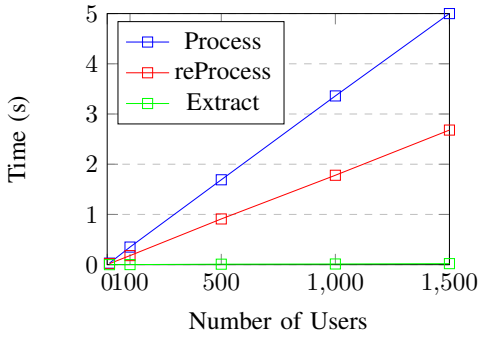

Fig. 2: Function Execution Times

Open Science and Reproducible Research: To support open science and reproducible research, our source codes have been anonymized and made publicly available[4].

[1] www.nordicsemi.com/Products/Development-hardware/nrf52840-dk

[2] https://www.zephyrproject.org/

[3] https://github.com/blynn/pbc

[4] https://anonymous.4open.science/r/DynamicDAS-F1EB

VIII. Conclusion

We presented scalable constructions for dropout-resistant time-series data aggregation by leveraging the concept of proxy re-encryption. The protocol can tolerate an arbitrary and unknown number of user dropouts and provides efficient support for join and leave operations. These properties together with the trust-less, low-bandwidth and non-interactive nature of our construction make it ideal for many rapid-changing distributed real-world networks.

Acknowledgment

This work was funded by Technology Innovation Institute (TII), UAE, for the project ARROWSMITH: Living (Securely) on the edge.

References

[1] A. Bakas, A. Michalas, and T. Dimitriou, "Private lives matter: A differential private functional encryption scheme," in *Proceedings of the Twelfth ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 300–311.

[2] A. Bakas, A. Michalas, E. Frimpong, and R. Rabaninejad, "Feel the quantum functioning: Instantiating generic multi-input functional encryption from learning with errors," in *Data and Applications Security and Privacy XXXVI: 36th Annual IFIP WG 11.3 Conference, DBSec 2022, Newark, NJ, USA, July 18–20, 2022, Proceedings*. Springer, 2022, pp. 279–299.

[3] Q. Grundy, K. Chiu, F. Held, A. Continella, L. Bero, and R. Holz, "Data sharing practices of medicines related apps and the mobile ecosystem: traffic, content, and network analysis," *BMJ*, vol. 364, 2019.

[4] E. Shi, T. H. Chan, E. Rieffel, R. Chow, and D. Song, "Privacy-preserving aggregation of time-series data," in *Proc. NDSS*, vol. 2. Citeseer, 2011, pp. 1–17.

[5] M. Joye and B. Libert, "A scalable scheme for privacy-preserving aggregation of time-series data," in *Financial Cryptography and Data Security*. Springer, 2013, pp. 111–125.

[6] F. Karakoç, M. Önen, and Z. Bilgin, "Secure aggregation against malicious users," in *ACM Symposium on Access Control Models and Technologies*, 2021, pp. 115–124.

[7] T.-H. H. Chan, E. Shi, and D. Song, "Privacy-preserving stream aggregation with fault tolerance," in *Financial Cryptography and Data Security*. Springer, 2012, pp. 200–214.

[8] Q. Li and G. Cao, "Efficient privacy-preserving stream aggregation in mobile sensing with low aggregation error," in *Privacy Enhancing Technologies Symposium*. Springer, 2013, pp. 60–81.

[9] I. Leontiadis, K. Elkhiyaoui, and R. Molva, "Private and dynamic time-series data aggregation with trust relaxation," in *Cryptology and Network Security*. Springer, 2014, pp. 305–320.

[10] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.

[11] I. Leontiadis, K. Elkhiyaoui, M. Önen, and R. Molva, "Puda – privacy and unforgeability for data aggregation," *Cryptology and Network Security*, p. 3–18, 2015.

[12] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *Advances in Cryptology– EUROCRYPT'98*. Springer, 1998, pp. 127–144.

[13] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," *ACM Transactions on Information and System Security (TISSEC)*, 2006.

[14] R. Rabaninejad, M. A. Attari, M. R. Asaar, and M. R. Aref, "A lightweight auditing service for shared data with secure user revocation in cloud storage," *IEEE Transactions on Services Computing*, vol. 15, no. 1, pp. 1–15, 2022.

[15] G. Castagnos and F. Laguillaumie, "Linearly homomorphic encryption from DDH," in *Cryptographers' Track at the RSA Conference*. Springer, 2015, pp. 487–505.