Beyond Security: Achieving Fairness in Mailmen-Assisted Timed Data Delivery

Shiyu Li
Shai_Li@yeah.net
University of Electronic Science and Technology of China

Yuan Zhang*
ZY_LoYe@126.com
University of Electronic Science and Technology of China

Yaqing Song
YaqingS@163.com
University of Electronic Science and Technology of China

Hongbo Liu
hongbo.liu@uestc.edu.cn
University of Electronic Science and Technology of China

Nan Cheng
dr.nan.cheng@ieee.org
Xidian University

Hongwei Li
hongweili@uestc.edu.cn
University of Electronic Science and Technology of China

Dahai Tao
atfwuswy@163.com
University of Electronic Science and Technology of China

Kan Yang
Kan.Yang@memphis.edu
University of Memphis

ABSTRACT

Timed data delivery is a critical service for time-sensitive applications that allows a sender to deliver data to a recipient, but only be accessible at a specific future time. This service is typically accomplished by employing a set of mailmen to complete the delivery mission. While this approach is commonly used, it is vulnerable to attacks from realistic adversaries, such as a greedy sender (who accesses the delivery service without paying the service charge) and malicious mailmen (who release the data prematurely without being detected). Although some research works have been done to address these adversaries, most of them fail to achieve fairness.

In this paper, we formally define the fairness requirement for mailmen-assisted timed data delivery and propose a practical scheme, dubbed DataUber, to achieve fairness. DataUber ensures that honest mailmen receive the service charge, lazy mailmen do not receive the service charge, and malicious mailmen are punished. Specifically, DataUber consists of two key techniques: 1) a new cryptographic primitive, i.e., Oblivious and Verifiable Threshold Secret Sharing (OVTSS), enabling a dealer to distribute a secret among multiple participants in a threshold and verifiable way without knowing any one of the shares, and 2) a smart-contract-based complaint mechanism, allowing anyone to become a reporter to complain about a mailman's misbehavior to a smart contract and receive a reward. Furthermore, we formally prove the security of DataUber and demonstrate its practicality through a prototype implementation.

CCS CONCEPTS

• **Security and privacy → Security protocols**; *Public key encryption.*

KEYWORDS

Timed data delivery, smart contract, fairness, complaint-supported

1 INTRODUCTION

Time-sensitive applications (e.g., sealed-bid auctions, electronic voting, and digital time capsules) have become increasingly prevalent in digital worlds [3, 5, 6]. In these applications, a sender needs to deliver some data in a timed-release manner such that the recipient can access the data content only after a duration specified by the sender. For instance, in a sealed-bid auction, bidders deliver their bids to an auctioneer, and to ensure fairness, the bids can be "opened" by the auctioneer only after the bidding closes [23, 31].

Existing works for timed data delivery can be broadly categorized into two types: the time-locked puzzles (TLPs)-based ones and mailman-assisted ones. TLP-based schemes adopt an encrypt-then-solve paradigm: the sender generates a cryptographic puzzle, in which a decryption key is embedded as the solution; the sender encrypts the data using the key and sends the ciphertext along with the puzzle to the recipient; the recipient recovers the key by solving the puzzle and further decrypts the ciphertext [10, 26, 43]. TLP-based schemes do not introduce any additional entities and are very efficient on the sender side (since creating a cryptographic puzzle is easy). However, these schemes are computationally expensive for recipients due to the continuous computations required to search for the puzzle's solution.

Another line of research employs a set of mailmen to assist the sender in delivering the data in a threshold manner [9, 24, 47]. These schemes essentially share the same encrypt-then-release paradigm: the sender encrypts the data and sends the ciphertext to the target recipient, employs a set of mailmen by compensating them with a service charge and distributes the decryption key among the mailmen in a threshold way; at a prescribed (future) time, the mailmen jointly release the decryption key, allowing the recipient to recover the data content [49]. However, these schemes are vulnerable to misbehaved mailmen. Specifically, a lazy mailman may be absent in the delivery mission, while a malicious mailman might collude with the recipient to prematurely leak his share for personal gain. Worse still, both the lazy and malicious mailman

*Corresponding author

would always receive the service charge, regardless of whether the data is delivered on time[1].

In order to resist misbehaved mailmen, a promising solution is to leverage a smart-contract-based complaint mechanism, wherein a smart contract is utilized to verify the trustworthiness of the mailmen [40, 47]. To identify malicious mailmen, before the prescribed delivery time arrives, anyone (including the sender, the receiver, the mailmen, and the general public) can act as a reporter and complain about that some mailman has prematurely leaked the share by submitting a witness to the smart contract. If the complaint is valid, the reporter can receive a reward, and the sender can get compensation (both from the misbehaved mailman). With this mechanism, the smart contract can continuously detect whether each mailman misbehaves from beginning to end in a delivery mission. Moreover, to distinguish the honest mailmen from lazy ones, the smart contract checks whether each share is correctly released. Consequently, only the mailman who honestly follows the prescribed scheme would receive the service charge.

Nevertheless, new challenges arise due to the adoption of the complaint mechanism. Notably, a greedy sender may maliciously complain against innocent mailmen for profits. Additionally, when a reporter submits a complaint transaction to the smart contract regarding a premature leakage, there is a potential vulnerability to blockchain adversaries who "copy" the transaction and execute the "front-running" attack to steal the rewards [15, 22, 27].

Putting things together. As such, in addition to achieving timed data delivery, a secure and practical scheme should provide the following guarantees, which we subsume as a fundamental property of **fairness**:

- (honest mailman fairness) an honest mailman, who faithfully adheres to the prescribed scheme, would always receive the designated service charge
- (lazy mailman fairness) a lazy mailman, who aborts the delivery mission, would not receive the service charge,
- (malicious mailman fairness) a malicious mailman, who prematurely leaks the share, would be penalized,
- (reporter fairness) a reporter, who submits the first valid complaint of a leaked share, would receive the reward.

Despite the significance of fairness, it is not well investigated in existing schemes. In particular, some existing schemes assume that the mailmen will always honestly complete the delivery mission, making fairness trivial to achieve [9, 24, 30, 44]; others only provide weak fairness and are vulnerable to sophisticated attacks [40, 47] (which will be detailed in Section 2 and Section 4).

In this paper, we propose a complaint-supported timed data delivery scheme dubbed DataUber to achieve fairness. Particularly, we design a smart-contract-based complaint mechanism to verify the trustworthiness of the mailmen. With this mechanism, if a mailman prematurely leaks a share, anyone can complain to a smart contract, and the misbehaved mailman would be penalized; if a mailman honestly completes the data delivery mission, it would always receive the service charge.

[1]This issue cannot be trivially addressed by utilizing the "pay-on-delivery" paradigm for transferring the service charge, since everyone can maliciously send data to the recipient, and the latter has to pay for each delivery mission.

To prevent the greedy sender from breaking honest mailman fairness, we propose a new cryptographic primitive dubbed oblivious and verifiable threshold secret sharing (OVTSS). With OVTSS, the sender can distribute a secret among the mailmen in a threshold way without gaining any knowledge about the shares, and each mailman can verify the validity of the corresponding share. To prevent the blockchain adversary from breaking reporter fairness, we utilize a prover-designated zero-knowledge proof (ZKP) for the complaint mechanism. This prover-designated ZKP ensures that when a reporter complains about a premature share leakage, adversaries cannot generate a valid complaint transaction to steal the reporter's rewards.

The contributions of this paper are summarized as follows.

Fairness formalization. We investigate the practical requirements of mailmen-assisted timed data delivery and formalize a new notion, i.e. fairness, within a decentralized environment. Through an analysis of existing schemes, we conclude that none of them (or their slight extensions) achieve fairness in the context of mailmen-assisted timed data delivery.

OVTSS. We propose a powerful cryptographic primitive dubbed oblivious and verifiable threshold secret sharing (OVTSS), which we believe is of independent interest. OVTSS enables a dealer to share a secret among a set of participants in a threshold manner, while the dealer will not gain any information about each share, and each participant can verify the validity of the designated share.

DataUber construction. We develop DataUber, a mailmen-assisted timed data delivery scheme with fairness, upon OVTSS and the smart-contract-based complaint mechanism. We give a formal analysis showing that DataUber realizes fair timed data delivery.]

Prototype implementation. We implement a DataUber prototype based on Goerli [4] and present a comprehensive performance evaluation, which demonstrates its practicality.

Roadmap. The remainder of this paper is organized as follows. We review the related works in Section 2 and state the problem in Section 3. We overview DataUber in Section 4, define it in Section 5 and detail it in Section 6. We present the security proofs in Section 7 and present the implementation details and evaluation results in Section 8. Finally, we draw the conclusions and outlook the future work in Section 9.

2 RELATED WORK

The notion of timed data delivery was first proposed by May in a technique report [45]. May observed that numerous time-sensitive applications require the delivery of data in a timed-release manner, leading to the proposition of an escrow-based approach for implementing timed data delivery.

The first timed data delivery scheme was proposed by Rivest et al. [52], wherein a time-locked puzzle (TLP) is utilized in tandem with an encryption algorithm. Specifically, the decryption key is embedded within the solution to the puzzle, requiring the recipient to solve it in order to retrieve the data content [51]. Subsequent works following this research line focus on constructing puzzles using diverse cryptographic primitives to improve efficiency (e.g., the time spent in generating a puzzle is independent of the solving time) or enrich functionality (e.g., homomorphism) [7, 16, 38].

Scheme \ Fairness	Honest mailman would receive the service charge	Malicious mailman would not receive the service charge	Reporter's rewards would not be stolen
Key-servers-based [2, 9, 30, 44]	○	○	○
YOSO-style [14, 32]	○	●	○
SWE-based [24, 42]	○	●	○
SilentDelivery [40]	○	●	○
NDHC19 [47]	●	○[2]	○
DataUber	●	●	●

Table 1: Comparison between DataUber and related works

Nevertheless, the practicality of TLP-based schemes is debatable due to the following reasons. On the one hand, it is challenging for the sender to precisely control the delivery time. Factors beyond their control, such as the solver's computational capabilities and equipment, significantly impact the time required to solve the puzzle. This creates an opportunity for the recipient to outsource the solving process to a powerful server, allowing them to prematurely access the data content. On the other hand, solving the puzzle necessitates continuous computations, which not only introduces substantial costs but also causes a waste of resources [25].

Another research line employs a mailman to facilitate data delivery. Such a mailman-assisted paradigm originates from May's report [45], and prior schemes were proposed by Rivest et al. [52] and Bellare et al. [12]. In these schemes, a sender employs a mailman to release the decryption key at the designated time for delivering the data. Subsequently, various mailman-assisted schemes were proposed to enrich functionality [20, 49]. Compared with TLP-based schemes, mailman-assisted ones offer the advantage of achieving precise delivery without computationally expensive operations.

However, mailman-assisted schemes suffer from critical threats: once the mailman is breached, the data content would be prematurely recovered, undermining the primary goal of timed data delivery schemes. Furthermore, fairness poses an inherent challenge in mailman-assisted schemes, with limited investigation thus far. In cases where a mailman honestly fulfills its delivery duties, it is entitled to receive a service charge, ensuring fairness straightforwardly. If the mailman misbehaves, fairness can be easily compromised. For example, if the mailman colludes with the recipient to prematurely recover the data content for profits and cheats the sender, it would still receive the service charge, resulting in a breach of fairness.

To mitigate the above "single-point-of-failure" problem, subsequent works, such as YOSO-style protocols [14, 32], signature-based-witness-encryption (SWE)-based schemes [24, 42], and key-servers-based schemes [2, 9, 30, 44], essentially share a common paradigm: employing a committee comprising multiple members instead of a single one to reconstruct a secret in a threshold way. They can be utilized for timed data delivery, however, *none of them focus on fairness, nor can they be straightforwardly extended to achieve fairness*. We analyze these schemes in terms of fairness below.

YOSO-style protocols [14, 32]. In a YOSO MPC protocol, each party sends a single message and never speaks again in the protocol execution. The key idea behind the YOSO-style protocols is that when a party $\mathcal{S}$ speaks, $\mathcal{S}$ also conveys the message that it needs to speak later to another party $\mathcal{M}$ ($\mathcal{M}$ is typically a committee $\overrightarrow{\mathcal{M}}$ to avoid the single-point-of-failure issue). Then, $\overrightarrow{\mathcal{M}}$ will take the place of $\mathcal{S}$ to speak at the future time.

Regarding functionality, YOSO-style protocols imply mailmen-assisted timed data delivery, where $\mathcal{S}$ acts as a sender and delegates $\overrightarrow{\mathcal{M}}$ as a set of mailmen to deliver some message at a future time. In terms of security, YOSO-style protocols require correctness of the message sent into the future, i.e., the message delivered by $\overrightarrow{\mathcal{M}}$ is the same as that transmitted from $\mathcal{S}$. With correctness, YOSO-style protocols can be easily extended to achieve the honest mailman fairness defined in our work. However, they do not focus on the obliviousness of the transmission from $\mathcal{S}$ to $\overrightarrow{\mathcal{M}}$, i.e., $\mathcal{S}$ can know the message obtained (and thereby will be published) by each $\mathcal{M}_i$. Therefore, applying YOSO-style protocols in timed data delivery fails to achieve honest mailman fairness.

SWE-based schemes [24, 42]. In SWE-based schemes, timed data delivery is achieved by a proposed t-out-of-n signature-based witness encryption (SWE). SWE allows to encrypt some data with respect to n mailmen's public keys and some message T (which can be the prescribed delivery time), such that after t mailmen generate valid signatures on T under their private keys, the data can be decrypted using these signatures. For instance, in McFly [24], with the mailmen's public keys $pk_1, ..., pk_n$, for the data s to be delivered, a sender splits s into $ss_1, ..., ss_n$ in a (t, n)-threshold way such that $s = \sum_{j=1}^{t} \omega_{i_j} ss_{i_j}$, where ω_{i_j} is the corresponding Lagrange coefficient, and generates the ciphertext $c = \{c_0, c_1, ..., c_n\}$ as $c_0 = g_2^r, c_i = (e(H(T), pk_i)r \cdot g_T^{ss_i}\ \forall i \in [1, n]$, where r is a randomness, $e : G_1 \times G_2 \rightarrow G_T$ is a bilinear map, g_2 and g_T are generators of G_2 and G_T, respectively, and $H : \{0,1\}^* \rightarrow G_1$ is a hash function. To deliver s, each mailman generates a BLS signature σ_i using the secret key sk_i. With t valid signatures (say, $\sigma_1, ..., \sigma_t$), m can be recovered by decrypting c as $m = \prod_{i=1}^{t} c_i^{\omega_i} / e(\prod_{j=1}^{t} (\sigma_i)^{\omega_i}, c_0)$.

This paradigm can be extended to ensure malicious mailman fairness: when a mailman prematurely leaks a valid signature on T, the leakage can be detected by verifying the signature. However, it cannot be trivially tweaked to achieve honest mailman fairness. Once a sender employs some mailmen for a delivery with regard to the message T, anyone can generate a ciphertext that can be decrypted by those mailmen's signatures on T using only the public keys. A greedy sender can enjoy the delivery services provided by the mailmen employed by other senders without paying. In this case, McFly fails to achieve honest mailman fairness.

Key-servers-based schemes [2, 9, 30, 44]. In these schemes, a set of mailmen jointly share a master secret key, periodically derive a new one at each epoch, and publish their shares at the end of the

epoch. To deliver some data at a future epoch, a sender encrypts the data using the corresponding master public key, and the data will be delivered after the mailmen publish the shares.

Due to the adoption of asymmetric encryption, key-servers-based schemes cannot resist the greedy sender as SWE-based schemes do. Moreover, to achieve malicious mailman fairness, key-servers-based schemes implicitly require a verifiable distributed secret sharing to ensure that the benchmark used to assess each mailman's behavior corresponds to its share.

In the pursuit of fairness in timed data delivery, there are also some works such as NDHC19 [47] and SilentDelivery [40]. They leverage smart contracts to verify the trustworthiness of the mailmen and transfer the service charge to the honest ones. However, they are confronted with security and fairness issues. Particularly, NDHC19 is vulnerable to curious mailman: a single mailman can obtain the decryption key by launching a "zero-attack" as elaborated in Section 4. SilentDelivery fails to achieve honest mailman fairness since a sender can refuse to pay a well-behaved mailman: the mailmen's identity information recorded in the smart contract for payment is provided by the sender (please refer to [40] for more details), the sender can upload an incomplete one to avoid paying the service charge. Additionally, both of them suffer from front-running attacks [15, 22], failing to achieve reporter fairness.

A comparison between DataUber and the state-of-the-art schemes is provided in Table 1, where ◯ denotes the property is not considered/achieved and ● denotes that it is achieved.

3 PROBLEM STATEMENT

3.1 Notation and basic theories

In this paper, $r \xleftarrow{\$} S$ denotes randomly choosing an element r from a set S, $i++$ denotes $i = i + 1$, $|r|$ denotes the bit length of r, $\{x_i\}_1^n$ denotes the set $\{x_1, ..., x_n\}$, and x^i denotes x to the power of i. Given two assertions A_1 and A_2, $A_1 \wedge A_2$ is true iff they are both true.

Interactive algorithms. We use the notation below for an interaction Π between two parties $\mathcal{M}_1$ and $\mathcal{M}_2$:

$$\begin{pmatrix} out_1 \\ out_2 \end{pmatrix} \leftarrow \Pi \begin{pmatrix} \mathcal{M}_1(in_1) \\ \mathcal{M}_2(in_2) \end{pmatrix} (pp),$$

where for each $i \in \{1, 2\}$, $\mathcal{M}_i$ takes its private input in_i, has access to some public parameter pp, and outputs out_i. For ease of readability, in case that the output of $\mathcal{M}_i$ is not explicitly needed, we write $*$ instead of out_i.

Blockchain and smart contract. Blockchain is an append-only ledger. It records every change of participants' funds by a transaction [46, 56]. Every participant can generate and verify the transaction data, but no entity can fully control it. Smart contracts on the blockchain are public and transparent protocols. Once a smart contract is deployed, it will execute automatically as designed without a third party [17]. Due to the space limitation, please refer to [39, 53] for more details.

Shamir's secret sharing [54]. Shamir's secret sharing protocol enables a dealer holding a secret s to share it among a set of n participants $\{\mathcal{M}_i\}_1^n$. Any t participants can pool their shares and reconstruct s, but no coalition of fewer than t participants can get any information about s from their collective shares. It is described as follows. Let Z_q be a finite field that contains the domain of possible secrets, and with $|Z_q| > n$. Let $u_1, ..., u_n \in Z_q$ be distinct, nonzero elements. Given a secret $s \in Z_q$, the dealer chooses uniform $a_1, ..., a_n \in Z_q$ and defines the polynomial $f(x) = s + \sum_{i=1}^{t-1} a_i x^i$. The share of $\mathcal{M}_i$ is $ss_i = f(u_i) \in Z_q$. After collecting t shares (say, $ss_1, ..., ss_t$), the secret can be reconstructed as $s = \sum_{i=1}^{t-1} ss_i \cdot \prod_{j=1, j\neq i}^{t} \frac{u_j}{u_j - u_i}$.

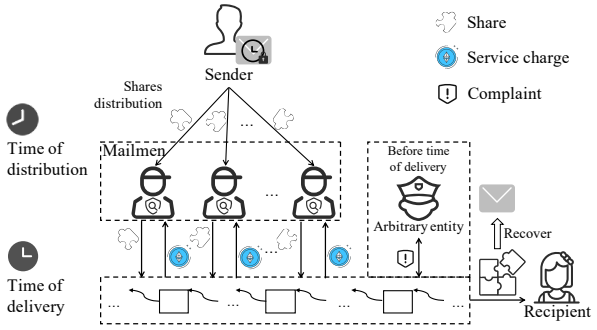

Figure 1: System model

Additively homomorphic encryption [13]. An encryption algorithm is additively homomorphic if fixing a plaintext space $\mathbb{M}$, and a ciphertext space $\mathbb{C}$, given two plaintexts $m_1, m_2 \in \mathbb{M}$ and their corresponding ciphertexts $c_1, c_2 \in \mathbb{C}$ under a public key pk, $c_1 \cdot c_2$ results in an encryption of $m_1 + m_2$.

Zero-knowledge proof (ZKP) [34]. In a ZKP for some relation R, a prover $\mathcal{P}$ holds a secret x, and a verifier $\mathcal{V}$ holds w. $\mathcal{P}$ is able to convince $\mathcal{V}$ that $R((w), (x)) = 1$ while "no knowledge" is yielded beyond the validity of the assertion.

Cryptographic commitment [19]. A party can "commit" to a message by generating a commitment, while nothing about the message is revealed. Later, the commitment can only "open" as the committed message.

Discrete logarithm (DL) assumption [21]. Given a cyclic group G and $g, h \xleftarrow{\$} G$, it is hard to compute $d = \log_g h$ in polynomial time.

3.2 System model

As depicted in Fig. 1, a typical complaint-supported mailmen-assisted timed data delivery scheme involves four entities:

- Sender. The sender publishes a timed data delivery mission via a smart contract, employs multiple mailmen for the mission, and distributes a portion of the data to each mailman.
- Mailmen. All mailmen collectively deliver the data by publishing their assigned shares through the smart contract at the designated time.
- Recipient. The recipient can recover the data content using the published shares when the delivery time has been reached.
- Smart contract. The smart contract assesses the behavior of mailmen and transfers some service charge to them based on its evaluation. The entities involved can complain about any premature share leakage to it.

We outline a general workflow of complaint-supported mailmen-assisted timed data delivery schemes as follows.

[2] In NDHC19 [47], one single mailman can obtain the decryption key of the data to be delivered without being detected by the sender.

(1) Encryption. A sender $\mathcal{S}$ encrypts the data to be delivered under a random key k using a symmetric-key encryption algorithm, sends the ciphertext to some recipient(s) and publishes a timed delivery mission via a smart contract.

(2) Distribution. $\mathcal{S}$ employs n mailmen, and each mailman contributes a deposit to the smart contract. Then, $\mathcal{S}$ splits k into multiple shares in a (t, n)-threshold way and distributes these shares to the employed mailmen.

(3) Complaint. Prior to the delivery time, any entity can complain to the smart contract about a premature share leakage by submitting the share. Then, the smart contract evaluates the complaint.

(4) Delivery. Once the designated delivery time arrives, mailmen submit their shares to the smart contract to deliver the data. The smart contract assesses mailmen's behaviors by verifying the submitted shares. After collecting at least t shares, the recipient can reconstruct k and further recover the data content.

3.3 Threat model and goals

We provide an intuitive discussion about threat model and design goals here and formalize them in Section 5.2.

Threat model. We consider all entities in DataUber to be rational and driven by the maximization of their own profits. Based on this premise, the complaint-supported mailmen-assisted timed data delivery is confronted with the following adversaries.

- Malicious recipient. A malicious recipient may compromise certain mailmen to prematurely recover the data.
- Greedy sender. A greedy sender may falsely accuse an honest mailman of prematurely revealing the share to benefit from the mailman's deposit. Additionally, the sender may attempt to avoid paying the service charge to an honest mailman.
- Lazy mailman. A lazy mailman may be absent during a delivery mission without being detected.
- Malicious mailman. A malicious mailman may intentionally leak their share ahead of schedule.
- Blockchain adversary. A blockchain adversary may monitor the transactions sent to the smart contract, copy a reporter's complaint transaction and execute a front-running attack.

What we do not focus on. Regarding the threat model of DataUber, we consider all mailmen to be rational. This assumption seems to conflict with the general threat model under the threshold paradigm, since the number of compromised mailmen may exceed the threshold in order to maximize their profits. However, this conflict can be eliminated by properly setting the amount of the deposit and the service charge. The orthogonal mechanisms, such as pricing mechanisms [18, 58], can be integrated as plug-in tools. Therefore, instead of going into the details of the case amount of the deposit and service charge, *we follow the general threshold cryptosystems and restrict our attention to the fact that the number of compromised mailmen will not exceed the threshold.*

Under this threat model, we require the following properties from a fair mailmen-assisted timed data delivery scheme.

- Time-locked confidentiality. As long as the number of misbehaved mailmen is less than the threshold, the data remains confidential to all entities (except the sender) until the designated delivery time.
- Mailman fairness.
  - Honest mailman fairness. A mailman should obtain the service charge if it faithfully adheres to the scheme, even in the presence of malicious complaints from the sender.
  - Lazy mailman fairness. A mailman absent in a delivery mission should not receive the service charge.
  - Malicious mailman fairness. If a mailman prematurely leaks its share and is complained against, it should be punished.
- Reporter fairness. If a reporter complains about the premature leakage of a share, others cannot steal its rewards.

4 TECHNICAL OVERVIEW OF DATAUBER

Plain scheme. Before presenting DataUber, we start with a plain instantiation of the workflow described in Section 3.2.

A sender $\mathcal{S}$ encrypts the data to be delivered under a random key k using a symmetric-key encryption algorithm and sends the ciphertext to the target receiver. $\mathcal{S}$ also publishes a timed-delivery mission via a timed-publishing smart contract *TimedPub*. After receiving applications, $\mathcal{S}$ selects n mailmen (denoted by $\{\mathcal{M}_i\}_1^n$) out of all applicants. Each $\mathcal{M}_i$ pays some deposit to *TimedPub*. Then, $\mathcal{S}$ generates a polynomial $f(x) = k + a_1 x + \cdots + a_{t-1}x^{t-1}$, splits k into n shares as $\{ss_i = f(i)\}_1^n$ using the Shamir's secret sharing protocol, and distributes ss_i to $\mathcal{M}_i$. Finally, $\mathcal{S}$ uploads the hash values of the shares (denoted by $h(ss_i), \forall\, i \in [1, n]$) to *TimedPub* as "benchmarks" for subsequent verifications.

Before delivery, any entity can complain about some mailman's premature leakage to *TimedPub* by submitting a leaked share (say, ss_i). *TimedPub* judges the complaint by comparing ss_i with $h(ss_i)$. If it is valid, some of the deposit paid by the misbehaved mailman is reassigned to the reporter as a reward and the rest part is transferred to $\mathcal{S}$ as compensation.

Challenge-1: resistance against greedy sender. The above plain scheme fails to achieve fairness since it is vulnerable to a greedy sender: as shown in Fig. 2-①, $\mathcal{S}$ knows all shares, he himself can complain about that $\mathcal{M}_i$ has prematurely leaked ss_i by submitting it to *TimedPub* as witness[3]. Then, $\mathcal{S}$ can obtain any mailman's deposit even if the mailman is well-behaved. It may appear that senders would not be motivated to launch such an attack, as they would want to prevent the publication of shares to maintain data confidentiality due to privacy reasons. However, a greedy sender can exploit the delivery of non-confidential data, such as public information or meaningless messages. By doing so, the sender can maliciously complain against all mailmen to get their deposits without privacy concerns.

The above attack can be resisted by making the share distribution *oblivious*: $\mathcal{M}_i$ chooses u_i, obtains the secret share $ss_i = f(u_i)$, and knows nothing else about $f(x)$, while $\mathcal{S}$ learns nothing about ss_i. This can be achieved by utilizing Paillier encryption[29]. Specifically, let pk_i denote $\mathcal{M}_i$'s public key, $\mathsf{Paillier.Enc}(pk_i, u_i)$ denote a Paillier encryption of u_i under pk_i. $\mathcal{M}_i$ computes $cu_i^{(j)} = \mathsf{Paillier.Enc}(pk_i, u_i^j), j = 1, ..., t-1$ and sends the ciphertexts to $\mathcal{S}$. Then, $\mathcal{S}$ encrypts k as $ck = \mathsf{Paillier.Enc}(pk_i, k)$. With ck and $\{cu_i^{(j)}\}_{j=1}^{t-1}$, $\mathcal{S}$ can compute $css_i = ck \cdot \prod_{j=1}^{t-1}(cu_i^{(j)})^{a_j}$ due to the additive homomorphism of Paillier encryption and send css_i to $\mathcal{M}_i$. Upon receiving css_i, $\mathcal{M}_i$ can decrypt it to get ss_i.

[3] A smarter attacker would collude with another mailman or control a blockchain account as a mailman to avoid detection.

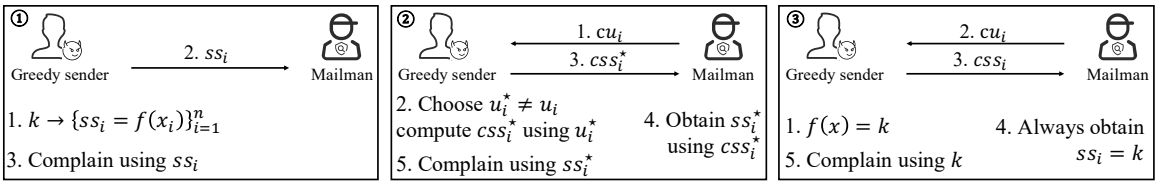

Figure 2: Vulnerabilities to greedy sender.

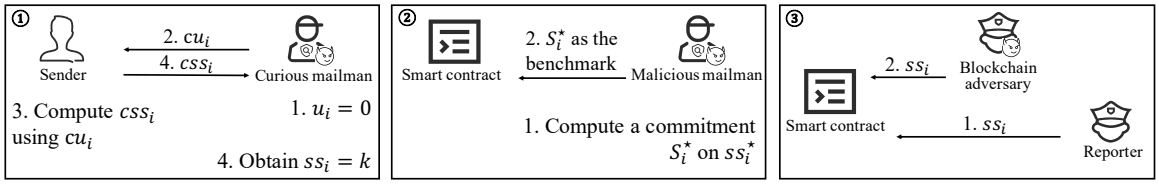

Figure 3: Vulnerabilities to adversarial mailman (① and ②) and blockchain adversary (③)

However, two sophisticated attacks from the greedy sender still work: i) as shown in Fig. 2-②, $\mathcal{S}$ chooses an arbitrary $u_i^\star$ and sends $css_i^\star = \text{Paillier.Enc}(pk_i, f(u_i^\star))$ instead of css_i to $\mathcal{M}_i$; ii) as shown in Fig. 2-③, $\mathcal{S}$ sets $f(x) = k + 0 \cdot x + \cdots + 0 \cdot x^{t-1}$ and honestly executes the prescribed scheme, and $\mathcal{M}_i$ will always obtain $ss_i = k$ with any u_i. As $f(x)$ is oblivious to $\mathcal{M}_i$, it can verify neither whether the underlying plaintext of $css_i^\star$ is indeed computed from u_i selected by herself/himself nor whether the coefficients of $f(x)$ are all 0. Subsequently, even if $\mathcal{M}_i$ follows the prescribed scheme, $\mathcal{S}$ can know the exact value of the share and submit it to *TimedPub* to complain about $\mathcal{M}_i$, and $\mathcal{M}_i$'s deposit would be transferred to $\mathcal{S}$.

To resist greedy sender, DataUber further makes the secret sharing *verifiable* to the mailmen: before interacting with $\mathcal{M}_i$, $\mathcal{S}$ generates a set of commitments enabling $\mathcal{M}_i$ to verify the correspondence between css_i and u_i. The key technique used here is Pedersen's commitment [57], which has additive homomorphism and is compatible with Paillier encryption. $\mathcal{S}$ further generates a ZKP π_{NZM} to prove that the underlying message of one commitment is not 0. By doing so, ss_i is verifiable to $\mathcal{M}_i$ while is oblivious to others. Moreover, the non-zero property of the $f(x)$'s coefficients is also verifiable to $\mathcal{M}_i$, while no additional knowledge of them is revealed. This ensures that no information helping adversaries prematurely recover the data content is exposed.

Challenge-2: resistance against curious mailmen. The above scheme fails to resist a curious mailman, who launches a so-called "zero-attack" shown in Fig. 3-①: if $\mathcal{M}_i^\star$ chooses $u_i = 0$ to execute the above scheme together with $\mathcal{S}$, he can obtain $f(u_i) = k$. Since Paillier encryption is indistinguishable against chosen-plaintext attacks (IND-CPA) [11], $\mathcal{S}$ cannot detect such an attack, and no security would be guaranteed.

To thwart the zero-attack, the key observation is that iff $u_i \neq 0$, then $\exists v_i$ s.t. $w_i = u_i \cdot v_i \neq 0$. Furthermore, when u_i and v_i are both chosen by $\mathcal{M}_i$, given w_i, $\mathcal{S}$ can confirm $u_i \neq 0$ by verifying $w_i \neq 0$ without gaining any additional knowledge about u_i. With this observation, we propose a non-zero plaintext ZKP π_{NZP} and integrate it into DataUber to resist the zero-attack. The integration of Paillier encryption, Pedersen's commitment, and π_{NZP} yields an oblivious and verifiable threshold secret sharing (OVTSS). We defer its formal definition and construction to Appendix A.

Challenge-3: judgment delegation. In the above scheme, we cannot directly extend *TimedPub* to support the judgment, since it cannot verify the validity of the submitted shares for judging complaints or paying the service charge. The fundamental reason is that before ss_i is published, it is oblivious to all entities except $\mathcal{M}_i$. We stress that enabling contract judging cannot be trivially achieved by integrating a commitment scheme, i.e. $\mathcal{M}_i$ generates a commitment S_i on ss_i and uploads S_i to *TimedPub* as the benchmark for subsequent judgments, since as shown in Fig. 3-②, a malicious $\mathcal{M}_i^\star$ could generate a commitment $S_i^\star$ on an arbitrary $ss_i^\star \neq ss_i$. By doing so, before the invalidity of $ss_i^\star$ is detected, $\mathcal{M}_i^\star$ has received the service charge, which compromises mailman fairness.

To resolve this tension, we construct a ZKP π_{SD}, which proves that both css_i and S_i are derived from the same ss_i. With π_{SD}, $\mathcal{M}_i$ can prove the validity of S_i. If S_i is valid, $\mathcal{S}$ uploads it to the smart contract for subsequent verifications.

Challenge-4: resistance against blockchain adversary. When a reporter $\mathcal{M}_j$ complains about a premature leakage of a share (say, ss_i), it uploads ss_i to *TimedPub* by sending a transaction T_j containing ss_i. If the complaint is valid, $\mathcal{M}_j$ receives a reward. However, as shown in Fig. 3-③, a blockchain adversary $\mathcal{M}_j^\star$ may steal the reward by launching a copy-then-front-running attack: once observing T_j from $\mathcal{M}_j$, $\mathcal{M}_j^\star$ conducts a complaint transaction $T_j^\star$ containing ss_i and inserts $T_j^\star$ before T_j by delaying T_i or paying higher transaction fee on $T_j^\star$. Due to this vulnerability to the front-running attack, the first reporter would always lose the reward.

A straightforward solution is to use a NIZKP for the knowledge of ss_i, which allows $\mathcal{M}_j$ to prove her/his knowledge to *TimedPub* without revealing any information about ss_i [8]. However, the copy-then-front-running attack still works since without need of ss_i, $\mathcal{M}_j^\star$ can directly copy the proof in the complaint transaction generated by $\mathcal{M}_j$. To resist this attack, we propose a prover-designated ZKP scheme allowing $\mathcal{M}_j$ to prove the knowledge of ss_i to *TimedPub* without revealing any information about ss_i, while the proof is specific to the account address of $\mathcal{M}_j$, which will be detailed in

Section 6.1. With this mechanism, $\mathcal{M}_j^\star$ cannot generate a valid complaint transaction from his account address by observing others' complaints. This yields DataUber, a complaint-supported timed data delivery scheme with fairness.

5 DEFINITION

5.1 Syntax and correctness

Definition 1. *DataUber is a tuple of seven algorithms:* (Setup, PubMission, Registration, KeyDis, Complaint, PubShare, Refund), *where* Registration *and* KeyDis *are interactively executed between a sender and a mailman.*

$(sp, \{sk_i\}_1^{\overline{n}}) \leftarrow \mathsf{Setup}(1^\ell)$:

The setup algorithm takes a security parameter ℓ as the input, outputs a set of system parameters sp and the secret keys $\{sk_i\}_1^{\overline{n}}$ of all mailmen. sp is an implicit input of the following algorithms.

Phase 1. Encryption.

$(\alpha, ID_T) \leftarrow \mathsf{PubMission}(k, payment, ts, Aux)$:

This algorithm is run by the sender $\mathcal{S}$ to publish a mission via a smart contract. $\mathcal{S}$ inputs the encryption key k of the data to be delivered, publishes a mission by transferring some service charge $payment$ to a smart contract and publishing the time slot ts in the data should be delivered and some auxiliary information Aux via the contract. After the mission is successfully published, $\mathcal{S}$ outputs some commitment(s) α (used in the following algorithms) and the identity ID_T of the mission.

Phase 2. Distribution.

$\begin{pmatrix} * \\ ind_i \end{pmatrix} \leftarrow \mathsf{Registration}\left\langle \begin{matrix} \mathcal{S} \\ \mathcal{M}_i \end{matrix} \right\rangle (ID_T, Add_{\mathcal{S}}, Add_i, deposit)$:

Registration is an interactive algorithm between a mailman $\mathcal{M}_i$ and $\mathcal{S}$, which enables $\mathcal{M}_i$ to apply for a published mission. $\mathcal{M}_i$ inputs ID_T to specify the mission and the account address $Add_{\mathcal{S}}$ of $\mathcal{S}$ to apply for the mission. After receiving the application, $\mathcal{S}$ inputs the account address Add_i of $\mathcal{M}_i$ to employ it. Then, $\mathcal{M}_i$ transfers some coins $deposit$ to the smart contract as a deposit and obtains her/his index ind_i in the mission from the contract.

$\begin{pmatrix} S_i \\ ss_i \end{pmatrix} \leftarrow \mathsf{KeyDis}\left\langle \begin{matrix} \mathcal{S}(k) \\ \mathcal{M}_i(u_i, sk_i) \end{matrix} \right\rangle (pk_i, \alpha)$:

It is an interactive algorithm in which $\mathcal{S}$ distributes a share ss_i of the decryption key k to $\mathcal{M}_i$. $\mathcal{S}$ inputs k and $\mathcal{M}_i$ inputs a random u_i. Finally, $\mathcal{M}_i$ obtains ss_i (which depends on u_i) and $\mathcal{S}$ obtains a commitment S_i on ss_i.

Phase 3. Complaint.

$(\pi^{(Add_i)}/\perp) \leftarrow \mathsf{Complaint}(ID_T, ss_j, S_j, ind_j, Add_i)$:

If a mailman with ind_j in mission ID_T prematurely leaks ss_j, any entity with an address Add_i can run this algorithm to generate a proof $\pi^{(Add_i)}$ for the premature leakage using ss_j. Output the proof if the complaint is valid and $\perp$ otherwise.

Phase 4. Delivery.

$(1/0) \leftarrow \mathsf{PubShare}(ID_T, Add_i, ss_i, ind_i)$:

This algorithm is run by a mailman $\mathcal{M}_i$ to publish ss_i for the mission ID_T. Output 1 if ss_i is valid and published at the prescribed time and 0 otherwise.

Refund (ID_T):

This algorithm is run to refund the rest of the coins locked in the smart contract to the corresponding payment accounts.

Definition 2. *(Correctness). DataUber is correct if when a sender and a mailman are honest (i.e. follow the scheme), the mailman will obtain a valid secret share. Moreover, if more than threshold number of mailmen are honest, the receiver can reconstruct the decryption key at the prescribed delivery time.*

5.2 Formalizing notions

We formalize the security notions of DataUber as follows, where the games for the notions are presented in Fig. 4.

Time-locked confidentiality. Timed-locked confidentiality asserts the property that as long as the number of mailmen compromised by an adversary is fewer than the threshold t, the adversary cannot obtain any additional information about the decryption key. We capture time-locked confidentiality using the game TL-CONF, where the adversary $\mathcal{A}$ chooses two decryption keys (denoted by k_0, k_1) and guesses which one is used. TL-CONF allows $\mathcal{A}$ to compromise up to $t-1$ mailmen and fully control them. After compromising a mailman, $\mathcal{A}$ can obtain all secret information held by the mailman and takes her/his place to interact with an honest sender.

The time-locked confidentiality is defined below.

Definition 3. *(Time-locked confidentiality). DataUber achieves time-locked confidentiality if, for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, there is a negligible function* negl *s.t.*

$$|\Pr[\text{TL-CONF}_{\mathcal{A}}(1^\ell) \Rightarrow 1] \leq \frac{1}{2} + \text{negl}(1^\ell).$$

Mailman fairness. We formalize it by honest mailman fairness, lazy mailman fairness, and malicious mailman fairness.

Honest mailman fairness requires that as long as a mailman does not prematurely leak her/his share, the mailman can get the deposit back. However, as discussed in Section 4, a greedy sender may complain that an honest mailman prematurely leaks the share to earn the mailman's deposit. Therefore, it requires that a sender cannot obtain the share requested by a mailman in the key distribution phase. We capture it by defining an HM-FAIR game. In HM-FAIR, an adversary $\mathcal{A}$ (who is actually a greedy sender) interacts with a mailman $\mathcal{M}_i$ to distribute a key. After the distribution, $\mathcal{A}$ tries to complain $\mathcal{M}_i$ by proving that he knows the share obtained by $\mathcal{M}_i$. If the complaint is valid, $\mathcal{A}$ wins. Note that if $\mathcal{A}$ compromises $\mathcal{M}_i$, he can trivially win the game. To avoid the trivial winning, in HM-FAIR, $\mathcal{M}_i$ is honest and follows the prescribed scheme.

The honest mailman fairness is defined below.

Definition 4. *(Honest mailman fairness). DataUber achieves honest mailman fairness if, for any PPT adversary $\mathcal{A}$, there is a negligible function* negl *s.t.*

$$\Pr[\text{HM-FAIR}_{\mathcal{A}} \Rightarrow 1] \leq \text{negl}(\ell).$$

Definition 5. *(Lazy mailman fairness). DataUber achieves lazy mailman fairness, if any mailman who does not publish her/his share on time would not receive the service charge.*

Malicious mailman fairness requires that if a premature share leakage is complained about, the corresponding mailman would be punished. We capture this property by defining the game MM-FAIR, where an adversary $\mathcal{A}$ interacts with a sender $\mathcal{S}$ in KeyDis and obtains a share while $\mathcal{S}$ obtains a commitment. $\mathcal{A}$ wins iff the algorithm Complaint outputs $\perp$ with the input ss_i and S_i.

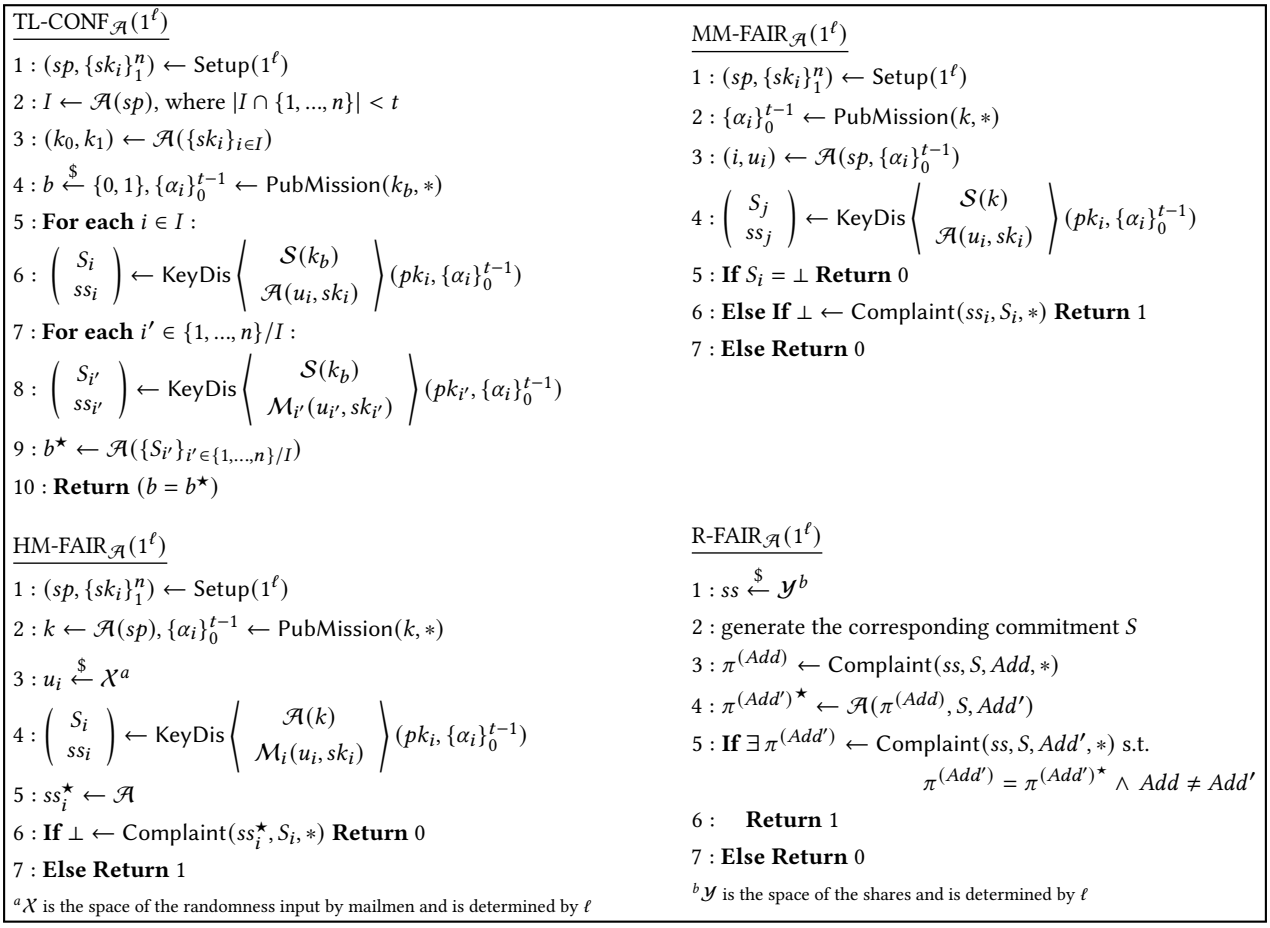

Figure 4: Games defining time-lock confidentiality (TL-CONF), honest mailman fairness (HM-FAIR), malicious mailman fairness (MM-Fair), and reporter fairness (R-FAIR), where the parameters that are not explicitly needed are presented by $*$

The malicious mailman fairness is defined below.

Definition 6. *(Malicious mailman fairness). DataUber achieves malicious mailman fairness if, for any PPT adversary $\mathcal{A}$, there is a negligible function* negl *s.t.*

$$|\Pr[\text{MM-FAIR}_{\mathcal{A}} \Rightarrow 1]| \leq \text{negl}(\ell).$$

Reporter fairness. Reporter fairness asserts the property that after a reporter complains the premature leakage of a share, an adversary without the knowledge of the leaked share cannot steal the rewards. In this work, we consider a blockchain adversary who keeps monitoring the transactions sent to *TimedPub* and can reorder the complaint transactions. In this case, after observing a complaint transaction containing a proof, the adversary may try to generate a valid complaint transaction using the proof generated by the reporter and make his transaction triggers the smart contract earlier. Therefore, reporter fairness requires that given a proof specific to an account address *Add*, without the knowledge of the share, the adversary $\mathcal{A}$ cannot forge a valid proof specific to another account address $Add_{\mathcal{A}} \neq Add$. We capture this property by defining a R-FAIR game, where $\mathcal{A}$ is given a proof $\pi^{(Add)}$ associate with a randomly chosen secret share ss_i and aims to output a valid proof $\pi^{(Add_{\mathcal{A}})}$. The reporter fairness is defined below.

Definition 7. *(Reporter fairness). DataUber achieves reporter fairness if, for any PPT adversary $\mathcal{A}$ with an address $Add_{\mathcal{A}}$, there is a negligible function* negl *s.t.* $|\Pr[\text{R-FAIR}_{\mathcal{A}} \Rightarrow 1]| \leq \text{negl}(1^\ell)$.

6 CONSTRUCTION OF DATAUBER

6.1 Building blocks

Paillier encryption [48]. An additively homomorphic encryption scheme consisting of the following three algorithms:

- $(sk, pk) \leftarrow \mathsf{Paillier.Gen}(1^\ell)$: on input a security parameter 1^ℓ, choose random primes $\overline{p}, \overline{q}$ with $|\overline{p}| = |\overline{q}|$, compute $N = \overline{p} \cdot \overline{q}, \phi(N) = (\overline{p} - 1) \cdot (\overline{q} - 1)$, output $sk = (\phi(N), N)$ as the private key and $pk = N$ as the public key.
- $c \leftarrow \mathsf{Paillier.Enc}(pk, m; r)$: on input the public key $pk = N$, a message $m \in Z_N$, and a random $r \in Z_N^*$, output the ciphertext $c = (1 + N)^m \cdot r^N \bmod N^2$.

TimedPub--Smart contract for DataUber

1: $i = 0, count[\,] = 0$;
2: function *Mission* $\left(\{\_\alpha_j\}_1^n, \_payment, \_ts, \_Aux\right)$:
  $salary[i] = \_payment/n$; $ts[i] = \_ts$;
  for $j = 1$ to n **do** $\alpha[i][j] = \_\alpha_j$;
  end for
  $i++$;
  return $ID_T = i - 1$;

3: function *Register* $(\_ID_T, \_deposit)$:
  if $count[\_ID_T] < n \wedge$ the invoker did not register **then**
    $count[\_ID_T]++$; $\mathcal{M}\left[count[\_ID_T]\right] = \_deposit$;
    return $index = count[\_ID_T]$;
  else return $\perp$;
  end if

4: function *Shares* $\left(\_ID_T, \{\_S_j\}_1^n\right)$:
  if invoker is $\mathcal{S}$ **then**
    for $j = 1$ to n **do** $Share[\_ID_T][j] = \_S_j$;
    end for
  end if

5: function *Complain* $\left(\_ID_T, \_ind_j, \_ss_j\right)$:
  if $g^{\_ss_j} = Share[\_ID_T]\left[\_ind_j\right] \wedge$
  the current time is before $ts[\_ID_T]$ **then**
    transfer $\mathcal{M}[\_ID_T]\left[\_ind_j\right]/2$ to the invoker;
    transfer $\mathcal{M}[\_ID_T]\left[\_ind_j\right]/2$ to $\mathcal{S}$;
  end if

6: function *PubShare* $(\_ID_T, \_ind_i, \_u_i, \_ss_i)$:
  if $g^{\_ss_i} = Share[\_ID_T][\_ind_i] \wedge$
  $\mathcal{M}[\_ID_T]\left[\_ind_j\right] \neq 0 \wedge$
  the current time is in $ts[\_ID_T]$ **then**
    transfer $salary[\_ID_T]$ to $\mathcal{M}_i$;
  end if

7: function *Refund* $(\_ID_T)$:
  if the current time is after $ts[\_ID_T]$ **then**
    return the rest of coins to payers;
  end if

Figure 5: Pseudocode of *TimedPub*, where variables whose names start with an underline are parameters input by invokers

- $m \leftarrow \mathsf{Paillier.Dec}(sk, c)$: on input a ciphertext c and the private key $sk = (\phi(N), N)$, output the plaintext

$$m = \frac{(c^{\phi(N)} \bmod N^2) - 1}{N} \cdot \phi(N)^{-1} \bmod N.$$

Pedersen's commitment [50]. An additively homomorphic commitment scheme consisting of three algorithms:

- $(p, q, G, g, h) \leftarrow \mathsf{Commit.Gen}(1^\ell)$: on input a security parameter 1^ℓ, output (p, q, G, g, h), where p, q are primes such that $q|p-1$; G is a group with the order q; g, h are generators of G such that $\log_g(h)$ is unknown.
- $\alpha \leftarrow \mathsf{Commit.Com}(m; r)$: with the message $m \in Z_q$ and a random $r \in Z_q$, the corresponding commitment, $\alpha = g^m h^r$, is computed and output.
- $1/0 \leftarrow \mathsf{Commit.Open}(\alpha, m, r)$: on input a commitment α, a committed message m, and the randomness r, verify the validity of α by checking $\alpha \stackrel{?}{=} g^m h^r$.

Prover-designated ZKP for discrete logarithm $\pi_{\mathsf{DL}}^{(ID)}$. A prover-designated ZKP for the prover with the identity ID to prove the knowledge of the solution of a discrete logarithm problem. Formally,

$$R_{\mathsf{DL}} = \{(G, g \in G, S \in G); (s) | S = g^s\}.$$

The proof $\pi_{\mathsf{DL}}^{(ID)}$ can be generated as follows.

- The prover randomly chooses $r \in Z_q$, computes $R = g^r$, $z = r + s \cdot H(S, R, ID)$, where $H : \{0,1\}^* \rightarrow Z_q$ is a hash function modeled as a random oracle, and sends $\pi_{\mathsf{DL}}^{(ID)} = \{R, z, ID\}$ to the verifier.
- The verifier verifies the validity of $\pi_{\mathsf{DL}}^{(ID)}$ by checking $R \cdot S^{H(S,R,ID)} \stackrel{?}{=} g^z$.

We defer the details of the following ZKP schemes to Appendix B.

ZKP for ordered-exponential computation on the same base π_{OEC} [36]. A ZKP for the relation that multiple plaintexts are a sequence of powers. Formally,

$$R_{\mathsf{OEC}} = \{((pk = N, \{cu^{(i)} \in Z_{N^2}\}_{i=1}^{t-1}, u \in Z_N) | \\ cu^{(i)} = \mathsf{Paillier.Enc}(pk, u^i; r_i)\ \forall i \in [1, t-1]\}.$$

ZKP for non-zero plaintext π_{NZP}. A ZKP for the relation that the value of the underlying plaintext of a ciphertext is not 0. Formally,

$$R_{\mathsf{NZP}} = \{(N, c_1 \in Z_{N^2}); (m_1 \in Z_N, r_1 \in Z_N^*) | \\ c_1 = \mathsf{Paillier.Enc}(pk, m_1; r_1) \wedge\ m_1 \neq 0\}.$$

ZKP for non-zero message π_{NZM}. A ZKP for the relation that given a Pedersen's commitment, the value of its underlying message is not 0. Formally,

$$R_{\mathsf{NZM}} = \{(G, p, q, g \in G, h \in G, \alpha_1 \in G); (a_1 \in Z_q, r_1 \in Z_q) | \\ \alpha_1 = \mathsf{Commit.Com}(a_1; r_1) \wedge\ a_1 \neq 0\}.$$

ZKP for same derivation π_{SD}. A ZKP for the relation that the exponent of an exponentiation in the group G is the corresponding plaintext of a Paillier ciphertext. Formally,

$$R_{\mathsf{SD}} = \{(N, cu \in Z_{N^2}, G, g \in G, U \in G); (u, R) | \\ cu = (1+N)^u \cdot R \bmod N^2 \wedge U = g^u\}.$$

6.2 Smart contract for DataUber: *TimedPub*

We then present the constructed smart contract *TimedPub*. As shown in Fig. 5, *TimedPub* consists of six functions: *Mission, Register, Shares, Complain, PubShare*, and *Refund*.

- *Mission.* It can be invoked by a sender to publish a timed data delivery mission, where some related information, such as the amount of the service charge and prescribed delivery time, is published via the blockchain. After a mission is published, it returns the identity of the mission.
- *Register.* It can be invoked by a mailman to register for a published mission, where the mailman transfers some coins as the deposit to *TimedPub*.
- *Shares.* It can be invoked by a sender to publish all commitments on the shares distributed to the mailmen. These commitments are utilized for subsequent judgments.
- *Complain.* It can be invoked by any entity to complain about the premature leakage of a share by uploading the share as the witness. Then, this function verifies i) whether the current time is before the delivery time and ii) the validity of the complaint using the commitment (uploaded in *Shares*) of the share.

- *PubShare.* It can be invoked by a mailman to publish her/his share at a prescribed delivery time by uploading the share. After a share is published, this function verifies i) whether the current time is the prescribed delivery time and ii) the validity of the share using the corresponding commitment.
- *Refund.* It can be invoked to unlock all coins in the smart contract after a mission is finished.

6.3 Algorithm details

A sender $\mathcal{S}$ with a blockchain account address $Add_{\mathcal{S}}$, a recipient $\mathcal{R}$, and a set of mailmen $\{\mathcal{M}_1, ..., \mathcal{M}_{\overline{n}}\}$ with account addresses $\{Add_{\mathcal{M}_1}, ..., Add_{\mathcal{M}_{\overline{n}}}\}$ are involved in DataUber.

We assume that there is a secure channel between $\mathcal{S}$ and $\mathcal{R}$ for communication. The messages transmitted through these channels are encrypted and authenticated.

Setup. Let ℓ be a security parameter, system parameters $sp = \{t, n, pp\}$ are published, where t is a threshold, n is the number of employed mailmen, and $pp = (G, p, q, g, h) \leftarrow \mathsf{Commit.Gen}(1^\ell, t, n)$. Each mailman $\mathcal{M}_i$ runs $(sk_i, pk_i) \leftarrow \mathsf{Paillier.Gen}(1^\ell)$ and publishes the public key pk_i. The smart contract *TimedPub* shown in Fig. 5, is deployed on the blockchain with the address Add_C.

Phase 1. Encryption. In this phase, $\mathcal{S}$ executes PubMission to publish a timed transmission mission on the blockchain.

PubMission.

- Randomly choose $k \in Z_q^*$, encrypt m using a symmetric-key encryption algorithm (e.g., CBC[AES]) and key k, and send the ciphertext c to $\mathcal{R}$ via a secure channel.
- Choose $a_1, ..., a_{t-1} \xleftarrow{\$} Z_q^*$ and generates a polynomial $f(x) = k + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}$.
- Choose $r_0, ..., r_{t-1} \in Z_q$ and compute the commitments on k and a_i as $\alpha_0 = \mathsf{Commit.Com}(k; r_0)$, $\alpha_i = \mathsf{Commit.Com}(a_i; r_i)$, $\forall i \in [1, t-1]$.
- Generate a proof π_{NZM} for

$$R_{\mathsf{NZM}} = \{(G, p, q, g, h, \alpha_{t-1}); (a_{t-1}, r_{t-1}) | \alpha_{t-1} = \mathsf{Commit.Com}(a_{t-1}; r_{t-1}) \wedge a_{t-1} \neq 0\}.$$

- Invoke *TimedPub.Mission* $(\{\alpha_i\}_0^{t-1}, payment, ts, Aux)$ to publish a timed-delivery mission, where *payment* is the service charge, *ts* is the time slot in which m should be delivered, and *Aux* is the auxiliary information such as the amount of the deposit and π_{NZM}.

After publishing the mission, $\mathcal{S}$ obtains ID_T as the identity of the current mission.

Phase 2. Distribution. In this phase, two algorithms, Registration and KeyDis, are included. In Registration, any mailman can apply for the published mission, $\mathcal{S}$ selects n mailmen (say, $\mathcal{M}_1, ..., \mathcal{M}_n$) out of all applicants, and these mailmen pay some deposit for the mission. In KeyDis, $\mathcal{S}$ distributes k among all $\mathcal{M}_i$ in a threshold, oblivious, and verifiable way.

Registration.

- A mailman $\mathcal{M}_i$ who wants to take the mission first verifies the validity of π_{NZM}. If it is valid, $\mathcal{M}_i$ generates a transaction from Add_i to $Add_{\mathcal{S}}$, where the string "Apply" is integrated.
- $\mathcal{S}$ selects n mailmen (say, $\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_n$) out of all applicants and informs them.
- Each mailman $\mathcal{M}_i$ invokes *TimedPub.Register* by generating a transaction and obtains an index ind_i.

KeyDis.

- $\mathcal{M}_i$ randomly chooses u_i s.t. $(u_i^t - 1)/(u_i - 1) \leq (pk_i/q)$[4], encrypts u_i^j as $cu_j \leftarrow \mathsf{Paillier.Enc}(pk_i, u_i^j)$ for each $j \in [1, t-1]$, generates π_{NZP} for $R_{\mathsf{NZP}} = \{((pk, cu_1), u_i) | cu_1 \leftarrow \mathsf{Paillier.Enc}(pk, u_i) \wedge u_i \neq 0\}$, and sends $(cu_1, \pi_{\mathsf{NZP}})$ to $\mathcal{S}$.
- If π_{NZP} is valid, $\mathcal{S}$ accepts that $u_i \neq 0$, and $\mathcal{M}_i$ sends $\{cu_j\}_2^{t-1}$ to $\mathcal{S}$ and generates π_{OEC} for $R_{\mathsf{OEC}} = \{(pk_i, cu_1, ..., cu_{t-1}); (u) | cu_j = \mathsf{Paillier.Enc}(pk_i, u_i^j), \forall j \in [1, t-1]\}$.
- If the π_{OEC} is valid, $\mathcal{S}$ sends $\{cv_i, css_i\}$ to $\mathcal{M}_i$, where

$$cv_i \leftarrow \mathsf{Paillier.Enc}(pk_i, r_0) \cdot \prod_{j=1}^{t-1} (cu_j)^{r_j},$$

$$css_i \leftarrow \mathsf{Paillier.Enc}(pk_i, k) \cdot \prod_{j=1}^{t-1} (cu_j)^{a_j}.$$

- After receiving cv_i and css_i, $\mathcal{M}_i$ decrypts css_i to obtain a share $ss_i \leftarrow \mathsf{Paillier.Dec}(sk_i, css_i)$. Then, $\mathcal{M}_i$ verifies the validity of ss_i by computing $v_i \leftarrow \mathsf{Paillier.Dec}(sk_i, cv_i)$ and checking $g^{ss_i} h^{v_i} \stackrel{?}{=} \prod_{j=0}^{t-1} (\alpha_j)^{u_i^j}$. If it is not valid, $\mathcal{M}_i$ aborts.
- $\mathcal{M}_i$ computes $S_i = g^{ss_i}$, $R_i = css_i \cdot (1 + pk_i)^{-ss_i}$, generates π_{SD} for $R_{\mathsf{SD}} = \{(pk_i, css_i, G, g, S_i); (ss_i, R_i) | css_i = (1 + pk_i)^{ss_i} \cdot R_i \wedge S_i = g^{ss_i}\}$, and sends (S_i, π_{SD}) to $\mathcal{S}$.
- If the above proof is valid, $\mathcal{S}$ accepts it. Otherwise, $\mathcal{S}$ rejects and asks $\mathcal{M}_i$ for a valid one. After receiving all valid S_i, $\mathcal{S}$ invokes *TimedPub.Shares* $(ID_T, \{S_i\}_1^n)$ by generating a transaction to upload them to the smart contract for subsequent verifications for ss_i.

Phase 3. Complaint. In this phase, if some share (say, $\mathcal{M}_j$'s share ss_j) is leaked before the prescribed time slot, another mailman (say, $\mathcal{M}_i$) can run Complaint to complain about it.

Complaint.

- Given ss_j, $\mathcal{M}_i$ extracts S_j from the blockchain and concludes from $g^{ss_j} = S_j$ that $\mathcal{M}_j$ leaks his share (note that $S_1, ..., S_n$ are recorded on blockchain and everyone can extract them.).
- $\mathcal{M}_i$ generates an prover-designated proof $\pi_{\mathsf{DL}}^{(Add_{\mathcal{M}_i})}$ for $R_{\mathsf{DL}} = \{(G, g, S_j); (ss_i)) | S_j = g^{ss_j}\}$, and invokes *TimedPub.Report* $(ID_T, ind_j, \pi_{\mathsf{DL}}^{(Add_{\mathcal{M}_i})})$ by generating a transaction to report to *TimedPub* that ss_j is leaked.
- *TimedPub* verifies the validity of the complaint by checking the validity of $\pi_{\mathsf{DL}}^{(Add_{\mathcal{M}_i})}$. If it is valid, a part of the deposit paid by $\mathcal{M}_j$ is transferred to Add_i as a reward, and the rest is transferred to $Add_{\mathcal{S}}$ as compensation.
- After receiving the reward, $\mathcal{M}_i$ informs $\mathcal{S}$ that ss_j is leaked by sending $\{Add_{T_{re}}, ss_j\}$ to $\mathcal{S}$, where $Add_{T_{re}}$ is the address of the transaction created by *TimedPub* for rewarding $\mathcal{M}_i$.

Phase 4. Delivery. Two algorithms, PubShare and Refund, are included. In PubShare, the mailmen publish their secret shares, and $\mathcal{R}$ recovers the data content. After the mission is finished, anyone can run Refund to unlock the coins by invoking *TimedPub.Refund*.

[4]To ensure the correctness of the decryption, the range of u_i should be limited such that $f(u_i) \leq pk_i$.

PubShare.

- Each $\mathcal{M}_i$ invokes *TimedPub.PubShare* (ID_T, ind_i, u_i, ss_i) by generating a transaction to publish the share.
- *TimedPub* verifies the validity of the shares that mailmen publish by checking $g^{ss_i} \stackrel{?}{=} S_i$. If ss_i is valid, $\mathcal{M}_i$ gets the salary of the current mission.
- After t valid shares $\{ss_{j_1}, ..., ss_{j_t}\}$ are published, $\mathcal{R}$ reconstruct k using Lagrange interpolation formula and further decrypts c using k to obtain m.

Refund. After the mission is finished, $\mathcal{S}$ and an arbitrary mailman $\mathcal{M}_i, i \in [1, n]$ can invoke *TimedPub.Refund* (ID_T) by generating a transaction to get the rest of the coins locked in the smart contract.

6.4 Limitations

Any complaint-supported system inherently encounters a certain non-technical limitation: the detection of malicious behavior is not automated [37, 41]. In DataUber, the enforcement of the punishment for a premature leakage relies on a valid complaint. If no reporter complains about the leakage, the misbehavior would not be detected by the smart contract. We stress that in general, any complaint system (and similar one such as a tracing system [33]) serves more as a deterrence rather than providing foolproof security.

Moreover, in this work, we consider a model where the number of colluding mailmen is less than the threshold t. If more than t mailmen come together and pool their shares, they could jointly execute a distributed protocol and get the output of the protocol. Specifically, the mailmen might run a general MPC protocol that reconstructs the decryption key [55]. Nevertheless, the assumption limiting the collaboration to up to $t-1$ mailmen is crucial to all systems that adopt threshold secret sharing. If a larger collusion is allowed, the security enhancement achieved through the distribution of trust would be undermined in these systems.

7 SECURITY ANALYSIS

Based on the discussion in Section 5.2, we capture the security of DataUber (i.e. time-locked confidentiality, mailman fairness, and reporter fairness) in the following theorems.

Theorem 1. *If Pedersen's commitment [50] is unconditional hiding, then for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$*

$$|\Pr[\text{TL-CONF}_{\mathcal{A}}(1^\ell) \Rightarrow 1]| \leq \frac{1}{2} + \text{negl}(1^\ell).$$

Theorem 2. *Let $\Pr[\text{Solve}_G]$ denote the probability of solving a DL problem in group G, then for any PPT adversary $\mathcal{A}$*

$$\Pr[\text{HM-FAIR}_{\mathcal{A}} \Rightarrow 1] \leq 2\Pr[\text{Solve}_G] + \text{negl}(1^\ell).$$

Theorem 3. *If the smart contract used in DataUber always executes its code, DataUber achieves lazy mailman fairness and malicious mailman fairness.*

Theorem 4. *If the hash function H used in $\pi_{\text{DL}}^{(ID)}$ is modeled as a random oracle HO, then for any PPT adversary $\mathcal{A}$*

$$\Pr[\text{R-FAIR}_{\mathcal{A}} \Rightarrow 1] \leq \text{negl}(1^\ell).$$

We defer the detailed proofs to Appendix C.

8 IMPLEMENTATION AND EVALUATION

We implement a DataUber prototype using Python and Solidity, and the source code is available at

https://anonymous.4open.science/r/DataUber-1E48.

In our implementation, we choose a security level of 112 bits. For Pedersen's commitment, we have employed the 224-bit random ECP groups as defined in [1]. The Ethereum public test network, Goerli [4], serves as the underlying blockchain platform. The development of smart contract is facilitated by the elliptic-curve-solidity library[5], enabling computations within the elliptic curve group.

Based on the prototype, we evaluate the performance of DataUber by conducting experiments on a laptop equipped with an Intel Core i5 CPU and 16 GB LPDDR4X of RAM. It is important to underline that the evaluation presented herein does not account for the costs associated with encrypting the data and transmitting the ciphertext to the recipient.

8.1 Computation costs

On the sender side, the computation costs primarily arise from the distribution of the decryption key to the mailmen in a (t, n)-threshold manner. These costs escalate with the values of n and the threshold t. We evaluate the computation costs incurred by the sender for varying (t, n) configurations and present the experiment results in Table 2a. From the results, we observe that the computation costs on the sender side are greatly influenced by the value of n. For instance, when the sender employs 10 mailmen and sets the threshold to 7, the computation delay on the sender side is approximately 18.8 s.

Delay (s) n \ t	3	4	5	6	7	8	9	10
5	2.82	4.48	6.08	-	-	-	-	-
7	3.90	6.33	8.65	10.83	13.22	-	-	-
10	5.56	8.90	12.41	15.53	18.79	22.20	25.55	29.10

(a) On the sender side

t	3	4	5	6	7	8	9	10
Delay (s)	0.43	0.49	0.51	0.54	0.58	0.59	0.65	0.70

(b) On the mailman side

Table 2: Computation delay

On the mailman side, the computation costs mainly emerge from obtaining a share of the decryption key from the sender, which increase with the value of t. If a mailman submits a complaint to the smart contract regarding a premature leakage, additional computation costs are incurred for generating a proof of the knowledge of the leaked share. We evaluate the computation costs on the mailman side with diverse values of t. Based on the evaluation results presented in Table 2b, we observe that for the configuration $(t, n) = (7, 10)$, the average computation delay for the mailman stands at approximately 0.5 s.

[5]Implementation of Elliptic Curve arithmetic operations written in Solidity, https://github.com/witnet/elliptic-curve-solidity.

Scheme	Computation costs		
	Sender	Mailman	Recipient
tlock [30]	$4\mathsf{H} + \mathsf{P} + 2\mathsf{E}_G$	$(2tn + 2n + t - 2)\mathsf{E}_G +$ $(2tn - 4n)\mathsf{E}_{Z_q} + (2tn - 2n)\mathsf{M}_{Z_q}$	$\mathsf{P} + 3\mathsf{H} + \mathsf{E}_G$
McFly [24]	$(2n + 1)\mathsf{E}_G +$ $n\mathsf{M}_G + 3n\mathsf{P} + n\mathsf{H}$	$\mathsf{H} + \mathsf{E}_G$	$t\mathsf{L} + t\mathsf{E}_G + t\mathsf{M}_G + \mathsf{P}$
i-TiRE [9][6]	$\lvert w\rvert\mathsf{H} + (\lvert w\rvert + 2)\mathsf{E}_G + \mathsf{M}_G + \mathsf{P}$	$(\lvert w\rvert + 1)(\mathsf{H}\mathsf{M}_G) + 3\mathsf{E}_G + \mathsf{ZKP.Prove}$	$(\lvert w\rvert + 1)(\mathsf{H} + \mathsf{P}) + t\mathsf{ZKP.Vrfy} +$ $t\mathsf{L} + (t + \lvert w\rvert + 1)\mathsf{M}_G + (t + 2)\mathsf{E}_G$
NDHC19 [47]	$(2nt - 2n + t)\mathsf{E}_G + n\mathsf{Paillier.Enc}$ $+(nt - n)(\mathsf{M}_{Z_{N^2}} + \mathsf{E}_{Z_{N^2}} + 2\mathsf{M}_G)$	$(t - 1)(\mathsf{M}_G + \mathsf{E}_{Z_q} + \mathsf{M}_{Z_q} + \mathsf{Paillier.Enc})$ $+(3t - 2)\mathsf{E}_{Z_q} + 2\mathsf{Paillier.Dec}$	$t\mathsf{L} + t\mathsf{M}_{Z_q}$
DataUber	$t(2\mathsf{E}_G + \mathsf{M}_G) + (n + 1)\mathsf{ZKP.Prove}$ $+2n\mathsf{Paillier.Enc} + 3n\mathsf{ZKP.Vrfy}$ $+(nt - n)(2\mathsf{E}_{Z_{N^2}} + \mathsf{M}_{Z_{N^2}})$	$2\mathsf{ZKP.Vrfy} + (t - 2)\mathsf{E}_{Z_q} +$ $(t - 1)\mathsf{Paillier.Enc} + 3\mathsf{ZKP.Prove}$ $+2\mathsf{Paillier.Dec} + (t + 2)\mathsf{E}_G + \mathsf{M}_G$	$t\mathsf{L} + t\mathsf{M}_{Z_q}$

Table 3: Comparison of computation costs between DataUber and related works.

8.2 Communication costs

On the sender side, the communication costs primarily arise from employing n mailmen from a pool of applicants, publishing a mission, and distributing secret shares to the employed mailmen. We conduct experiments to evaluate the costs for different (t, n) configurations and present the results in Table 4a. In our experiments, with 20 applicants available, employing 10 mailmen out of the pool and setting the threshold to 7 takes communication costs of approximately 0.28 MB.

Cost (KB) n \ t	3	4	5	6	7	8	9	10
5	70.99	88.49	106.00	-	-	-	-	-
7	98.24	122.74	147.24	171.74	196.24	-	-	-
10	139.11	174.11	209.11	244.11	279.11	314.11	349.11	384.11

(a) On the sender side

t	3	4	5	6	7	8	9	10
Costs (KB)	13.96	17.46	20.96	24.46	27.96	31.46	34.96	38.46

(b) On the mailman side

Table 4: Communication costs

On the mailman side, the communication costs stem from activities including applying for a mission, registering with the smart contract, interacting with the sender to obtain a share, and releasing the share. These costs are independent of the value of n. We evaluate the communication costs for the mailman with different t values and present the results in Table 4b. Notably, when $t = 7$, the communication costs for a mailman are approximately 0.028 MB.

8.3 On-chain costs

We conduct an evaluation of the on-chain costs associated with various transactions in DataUber. The transactions subjected to evaluation encompass contract deployment, mission publication, application submission, registration, leakage complaint, shares' commitment upload, share publication, and refund process. A consolidated overview of the on-chain costs for each transaction is presented in Table 5.

Algorithm	Operation	Gas (t, n)
Setup	*TimedPub* deployment	4021202
PubMission	Mission publication	839181 (3, 5)
		922242 (5, 7)
		1005267 (7, 10)
Registration	application submission	21672
	Registration	27088
KeyDis	Commitment upload	293798 (3, 5)
		376847 (5, 7)
		501419 (7, 10)
Complaint	Leakage complaint	2673605
PubShare	Share publication	1455352

Table 5: On-chain costs

The most financially demanding on-chain operation is the complaint, which incurs a gas cost of 2673605. This is attributed to the necessity of validating a NIZKP as part of ensuring reporter fairness. We stress that in order to motivate complaints, the amount of the reward needs to exceed the cost of the complaint itself. This balance can be achieved through established strategies, such as game-theoretic analysis [18, 58], allowing for judiciously adjustment of the deposit and reward amounts, which are orthogonal with DataUber.

[6] In i-TiRE, $\lvert w\rvert$ depends on the time when the data should be delivered.

8.4 Comparison

To further demonstrate the practicality of DataUber, we compare the computation costs of DataUber with those of related mailmen-assisted timed-data delivery schemes, as presented in Table 3. In the comparison, we omit the computation costs incurred by establishing secure channels between entities. H stands for hashing, P stands for the bilinear map, E_S (resp. M_G) stands for exponentiation (resp. multiplication) in the set S, PKE.Enc stands for public-key encryption, L stands for Lagrange's interpolation, Paillier.Enc (resp. Paillier.Dec) stands for Paillier encryption (resp. decryption), ZKP.Prove (resp. ZKP.Vrfy) stands for proving using ZKP (resp. verifying the validity of the proof). Specifically, in comparison with tlock [30], when $(t, n) = (3, 5)$ and the security level is 112 bits, the computation delay on the sender side of DataUber is approximately increased by 2.7 s, while on the mailman side and the recipient side, the computation delay of DataUber is comparatively reduced. This comparison demonstrates that while DataUber ensures fairness at the expenses of higher computation costs compared with certain related schemes, it does not yield unacceptable costs.

9 CONCLUSION

In this paper, we have proposed DataUber, a blockchain-based mailmen-assisted timed data delivery scheme with fairness. DataUber utilizes a smart contract as a judger to check the trustworthiness of mailmen, where a reporter can complain about the misbehavior of mailmen and would receive rewards. To ensure security against a greedy sender and curious mailmen, we have proposed an oblivious and verifiable secret sharing OVTSS and integrated it into DataUber. We have formally defined the fairness of mailmen-assisted timed data delivery and proven the security of DataUber under the definition. We have also implemented a DataUber prototype and evaluated the performance to demonstrate its practicality. In the future work, we will investigate how to reduce the gas costs of the on-chain activities in DataUber.

REFERENCES

[1] 2008. Additional Diffie-Hellman Groups for Use with IETF Standards. https://www.rfc-editor.org/rfc/rfc5114.

[2] 2021. Shutter - In-Depth Explanation of How We Prevent Front Running. https://blog.shutter.network/shutter-in-depth-explanation-of-how-we-prevent-frontrunning/

[3] 2023. Boomerang. https://www.boomeranggmail.com.

[4] 2023. Goerli. https://goerli.net.

[5] 2023. Now Released (Fall 2010): Autobiography of Mark Twain, Volume 1. https://www.marktwainproject.org/about_absample.shtml

[6] 2023. Postfity. https://postfity.com.

[7] Aydin Abadi and Aggelos Kiayias. 2021. Multi-instance publicly verifiable time-lock puzzle and its applications. In *Proc. FC*. 541–559.

[8] Ghada Almashaqbeh, Fabrice Benhamouda, Seungwook Han, Daniel Jaroslawicz, Tal Malkin, Alex Nicita, Tal Rabin, Abhishek Shah, and Eran Tromer. 2021. Gage MPC: Bypassing residual function leakage for non-interactive MPC. In *Proc. PETS*, Vol. 4. 528–548.

[9] Leemon Baird, Pratyay Mukherjee, and Rohit Sinha. 2022. i-TiRE: Incremental timed-release encryption or how to use timed-release encryption on blockchains?. In *Proc. CCS*. 235–248.

[10] Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. 2021. TARDIS: a foundation of time-lock puzzles in UC. In *Proc. EUROCRYPT*. 429–459.

[11] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. 1997. A concrete security treatment of symmetric encryption. In *Proc. FOCS*. 394–403.

[12] Mihir Bellare and Shafi Goldwasser. 1996. Encapsulated key escrow.

[13] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. 2011. Semi-homomorphic encryption and multiparty computation. In *Proc. EUROCRYPT*. 169–188.

[14] Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. 2020. Can a public blockchain keep a secret?. In *Proc. TCC*. 260–290.

[15] Iddo Bentov, Yan Ji, Fan Zhang, Lorenz Breidenbach, Philip Daian, and Ari Juels. 2019. Tesseract: Real-Time Cryptocurrency Exchange Using Trusted Hardware. In *Proc. CCS*. 1521–1538.

[16] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. 2016. Time-lock puzzles from randomized encodings. In *Proc. ITCS*. 345–356.

[17] Thomas Bocek and Burkhard Stiller. 2018. Smart contracts–blockchains in the wings. In *Digital Marketplaces Unleashed*. 169–184.

[18] Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. 2003. A game theoretic framework for incentives in P2P systems. In *Proc. P2P*. 48–56.

[19] Dario Catalano, Dario Fiore, and Ida Tucker. 2022. Additive-homomorphic functional commitments and applications to homomorphic signatures. In *Proc. ASIACRYPT*. 159–188.

[20] Jung Hee Cheon, Nicholas Hopper, Yongdae Kim, and Ivan Osipkov. 2008. Provably secure timed-release public key encryption. *ACM Transactions on Information and System Security* 11, 2 (2008), 1–44.

[21] Henry Corrigan-Gibbs and Dmitry Kogan. 2018. The discrete-logarithm problem with preprocessing. In *Proc. EUROCRYPT*. 415–447.

[22] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. 2020. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *Proc. S&P*. 910–927.

[23] Bernardo David, Lorenzo Gentile, and Mohsen Pourpouneh. 2022. FAST: Fair auctions via secret transactions. In *Proc. ACNS*. 727–747.

[24] Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wohnig. 2022. McFly: Verifiable encryption to the future made practical. *Cryptology ePrint Archive* (2022).

[25] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. 2015. Proofs of space. In *Proc. CRYPTO*. 585–605.

[26] Karim Eldefrawy, Sashidhar Jakkamsetti, Ben Terner, and Moti Yung. 2023. Standard Model Time-Lock Puzzles: Defining Security and Constructing via Composition. Cryptology ePrint Archive. (2023). https://eprint.iacr.org/2023/439

[27] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. 2020. SoK: Transparent dishonesty: Front-running attacks on blockchain. In *Proc. FC*. 170–189.

[28] Amos Fiat and Adi Shamir. 1986. How to Prove Yourself: Practical Solutions to Identification and Signature Problems.. In *Proc. CRYPTO*, Vol. 86. 186–194.

[29] Michael J Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. 2005. Keyword search and oblivious pseudorandom functions. In *Proc. TCC*. 303–324.

[30] Nicolas Gailly, Kelsey Melissaris, and Yolan Romailler. 2023. tlock: Practical timelock encryption from threshold BLS. *Cryptology ePrint Archive* (2023).

[31] Hisham S Galal and Amr M Youssef. 2019. Trustee: Full privacy preserving vickrey auction on top of ethereum. In *Proc. FC*. 190–207.

[32] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. 2021. YOSO: You only speak once: secure MPC with stateless ephemeral roles. In *Proc. CRYPTO*. 64–93.

[33] Vipul Goyal, Yifan Song, and Akshayaram Srinivasan. 2021. Traceable secret sharing and applications. In *Proc. CRYPTO*. 718–747.

[34] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. 2021. Poseidon: A new hash function for zero-knowledge proof systems. In *Proc. USENIX Security*. 519–535.

[35] Carmit Hazay and Yehuda Lindell. 2009. Efficient oblivious polynomial evaluation with simulation-based security. *Cryptology ePrint Archive* (2009).

[36] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, Tomas Toft, and Angelo Agatino Nicolosi. 2019. Efficient RSA key generation and threshold paillier in the two-party setting. *Journal of Cryptology* 32, 2 (2019), 265–323.

[37] Rawane Issa, Nicolas Alhaddad, and Mayank Varia. 2022. Hecate: Abuse reporting in secure messengers with sealed sender. In *Proc. USENIX Security*. 2335–2352.

[38] Jonathan Katz, Julian Loss, and Jiayu Xu. 2020. On the security of time-lock puzzles and timed commitments. In *Proc. TCC*. 390–413.

[39] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2016. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Proc. S&P*. 839–858.

[40] Chao Li and Balaji Palanisamy. 2021. Silentdelivery: Practical timed-delivery of private information using smart contracts. *IEEE Transactions on Services Computing* 15, 6 (2021), 3528–3540.

[41] Linsheng Liu, Daniel S Roche, Austin Theriault, and Arkady Yerukhimovich. 2021. Fighting fake news in encrypted messaging with the fuzzy anonymous complaint tally system (facts). In *Proc. NDSS*.

[42] Varun Madathil, Sri AravindaKrishnan Thyagarajan, Dimitrios Vasilopoulos, Lloyd Fournier, Giulio Malavolta, and Pedro Moreno-Sanchez. 2023. Cryptographic oracle-based conditional payments. In *Proc. NDSS*.

[43] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. 2019. Homomorphic time-lock puzzles and applications. In *Proc. CRYPTO*. 620–649.

[44] Harjasleen Malvai, Lefteris Kokoris-Kogias, Alberto Sonnino, Esha Ghosh, Ercan Oztürk, Kevin Lewi, and Sean Lawlor. 2023. Parakeet: Practical key transparency

for end-to-end encrypted messaging. In *Proc. NDSS*.
[45] T. C. May. 1993. *Timed-release crypto.* Technical Report.
[46] Satoshi Nakamoto. 2008. *Bitcoin: A peer-to-peer electronic cash system.* https://bitcoin.org/bitcoin.pdf
[47] Jianting Ning, Hung Dang, Ruomu Hou, and Ee Chien Chang. 2018. Keeping time-release secrets through smart contracts. *Cryptology ePrint Archive* (2018).
[48] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. EUROCRYPT*. 223–238.
[49] Kenneth G Paterson and Elizabeth A Quaglia. 2010. Time-specific encryption. In *Proc. SCN*. 1–16.
[50] Torben Pryds Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proc. CRYPTO*. 129–140.
[51] Josef Pieprzyk and Eiji Okamoto. 1999. Verifiable secret sharing and time capsules. In *Proc. ICISC*. 169–183.
[52] Ronald L Rivest, Adi Shamir, and David A Wagner. 1996. Time-lock puzzles and timed-release crypto. (1996).
[53] Tara Salman, Maede Zolanvari, Aiman Erbad, Raj Jain, and Mohammed Samaka. 2018. Security services using blockchains: A state of the art survey. *IEEE Communications Surveys & Tutorials* 21, 1 (2018), 858–880.
[54] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.
[55] Sarisht Wadhwa, Luca Zanolini, Francesco D'Amato, Aditya Asgaonkar, Fan Zhang, and Kartik Nayak. 2023. Breaking the Chains of Rationality: Understanding the Limitations to and Obtaining Order Policy Enforcement. *Cryptology ePrint Archive* (2023). https://eprint.iacr.org/2023/868
[56] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper* (2014), 1–32.
[57] Zhe Xia, Bo Yang, Mingwu Zhang, and Yi Mu. 2018. An efficient and provably secure private polynomial evaluation scheme. In *Proc. ISPEC*. 595–609.
[58] Sixie Yu, Kai Zhou, Jeffrey Brantingham, and Yevgeniy Vorobeychik. 2020. Computing equilibria in binary networked public goods games. In *Proc. AAAI*, Vol. 34. 2310–2317.

A OVTSS

A.1 Syntax and correctness

OVTSS enables a sender to distribute a secret to n participants such that any t of them can pool their shares to reconstruct the secret, while the sender learns nothing about each share, each participant learns nothing about the secret, and the correctness of each share can be verified by the corresponding participant.

Definition 8. *(OVTSS). An OVTSS* OVTSS *is a tuple of five algorithms* (Setup, Prep, Share, Vrfy, Combine) *that satisfies the correctness property below.*

- $(\{sk_i\}_1^n, pp) \leftarrow \mathsf{Setup}(1^\ell, t, n)$: it generates a secret key sk_i for each participant $\mathcal{M}_i$ and public parameters pp (pp will be implicitly input to the algorithms below.).
- $(aux, \alpha) \leftarrow \mathsf{Prep}(k, t, n)$: it is run by the sender $\mathcal{S}$ to generate some auxiliary information aux to share the secret k and some commitment(s) α to support verifications.
- $\begin{pmatrix} v_i \\ ss_i \end{pmatrix} \leftarrow \mathsf{Share}\left\langle \begin{matrix} \mathcal{S}(k, aux) \\ \mathcal{M}_i(u_i, sk_i) \end{matrix} \right\rangle (pp)$: it is an interactive algorithm between $\mathcal{S}$ and $\mathcal{M}_i$, where $\mathcal{M}_i$ obtains a share ss_i using u_i, and $\mathcal{S}$ outputs the corresponding verification information v_i.
- $0/1 \leftarrow \mathsf{Vrfy}(\alpha, ss_i, v_i)$: it is run by $\mathcal{M}_i$ to verify the validity of ss_i. Output 1 if it is valid, and 0 otherwise.
- $k/\perp \leftarrow \mathsf{Combine}(\{u_i, ss_i\}_{i\in S}, t)$: it combines the shares received from participants in the set S to reconstruct the secret k. If the algorithm fails, it outputs $\perp$.

Definition 9. *(Correctness). For all $\ell \in \mathbb{N}$, any $t, n \in \mathbb{N}$ such that $t \le n$, all $(\{sk_i\}_1^n, pp)$ generated by* $\mathsf{Setup}(1^\ell, t, n)$*, any (aux, c) generated by* Prep*, and all* $\begin{pmatrix} ss_i \\ v_i \end{pmatrix} \leftarrow \mathsf{Share}\left\langle \begin{matrix} \mathcal{S}(aux, k) \\ \mathcal{M}_i(u_i, sk_i) \end{matrix} \right\rangle (pp)$*, if $\mathcal{S}$ and $\{\mathcal{M}_i\}_{i\in S}$ are honest, then* $1 \leftarrow \mathsf{Vrfy}(\alpha, ss_i, v_i)$*. Moreover, if $|S| \ge t$, then* $k \leftarrow \mathsf{Combine}(\{u_i, ss_i\}_{i\in S})$.

A.2 Security properties

We elaborate on the security properties that OVTSS should satisfy (i.e. verifiability, participant obliviousness, and dealer obliviousness) one by one and depicted them in Fig. 6.

Verifiability. Verifiability requires that when a participant requests a share ss_i of a secret k from a dealer, the participant can verify whether the obtained share $ss_i^\star$ corresponds to its input u_i. In other words, the dealer cannot forge verification information for an incorrectly computed share.

Definition 10. *(Verifiability). An OVTSS* OVTSS = (Setup, Prep, Share, Vrfy, Combine) *is verifiable if for all $t, n \in \mathbb{N}, t \le n$, and any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, there is a negligible function* negl *s.t.*

$$\Pr[\mathsf{Verifiability}_{\mathrm{OVTSS},\mathcal{A}}(1^\ell, t, n) = 1] \le \mathrm{negl}(1^\ell).$$

Obliviousness. We formalize obliviousness by defining *participant obliviousness* and *dealer obliviousness*.

Participant obliviousness requires that as long as the number of colluding mailmen is less than the threshold, they cannot obtain any information about the shared secret.

Definition 11. *(Participant obliviousness). An OVTSS* OVTSS = (Setup, Prep, Share, Vrfy, Combine) *is oblivious for participants if for all $t, n \in \mathbb{N}, t \le n$ and any PPT adversary $\mathcal{A}$, there is a negligible function* negl *s.t.*

$$\Pr[\mathsf{PObliviousness}_{\mathrm{OVTSS},\mathcal{A}}(1^\ell, t, n) = 1] \le \frac{1}{2} + \mathrm{negl}(1^\ell).$$

Dealer obliviousness requires that the sender cannot obtain any information about the shares requested by the participants.

Definition 12. *(Dealer obliviousness). An OVTSS* OVTSS = (Setup, Prep, Share, Vrfy, Combine) *is oblivious for dealers if for all $t, n \in \mathbb{N}, t \le n$, and any PPT adversaries $\mathcal{A}$, there is a negligible function* negl *s.t.*

$$\Pr[\mathsf{DObliviousness}_{\mathrm{OVTSS},\mathcal{A}}] \le \frac{1}{2} + \mathrm{negl}(1^\ell).$$

A.3 Construction

Our construction of OVTSS can be easily extracted from the algorithm KeyDis presented in Section 6.3, where the sender and the mailmen play the role of dealer and participants, respectively, and k is the secret to be distributed. We do not repeat the construction here to avoid redundancy.

B ZKP DETAILS

ZKP for consistent product π_{PRO} [35]. Given ciphertexts $c_1, c_2, c_3 \in Z_{N^2}$ output by Paillier.Enc, it proves that c_3 is generated by encrypting the product of the underlying plaintexts of c_1 and c_2. Formally,

$$R_{\mathsf{PRO}} = \{(pk = N, \{c_i \in Z_{N^2}\}_1^3); (\{m_i \in Z_N\}_1^2, \{r_i \in Z_N^*\}_1^3) |$$
$$c_1 \leftarrow \mathsf{Paillier.Enc}(pk, m_1; r_1) \wedge\ c_2 \leftarrow \mathsf{Paillier.Enc}(pk, m_2; r_2)$$
$$\wedge\ c_3 \leftarrow \mathsf{Paillier.Enc}(pk, m_1 \cdot m_2; r_3)\}.$$

We present the concrete steps of the 3-round Sigma protocol for R_{PRO} below, which can be converted to be non-interactive using Fiat-Shamir transform.

$\underline{\mathsf{Verifiability}_{\mathsf{OVTSS},\mathcal{A}}(1^\ell,t,n)}$

$1:(\{sk_i\}_1^n,pp)\leftarrow \mathsf{Setup}(1^\ell,t,n)$

$2:(k,aux,\alpha,u_i,ss_i^\star,v_i^\star)\leftarrow \mathcal{A}(1^\ell,pp,t,n)$

$3:\begin{pmatrix} v_i \\ ss_i \end{pmatrix}\leftarrow \mathsf{Share}\left\langle \begin{matrix} \mathcal{S}(k,aux) \\ \mathcal{M}_i(u_i,sk_i) \end{matrix}\right\rangle (pp)$

$4:z_i\leftarrow \mathsf{Vrfy}(\alpha,ss_i^\star,v_i^\star)$

$5:$ **If** $0\leftarrow \mathsf{Vrfy}(\alpha,ss_i,v_i)$ **Restart**

$6:$ **Return** $z_i\wedge(ss_i\neq ss_i^\star)$

$\underline{\mathsf{PObliviousness}_{\mathsf{OVTSS},\mathcal{A}}(1^\ell,t,n)}$

$1:(\{sk_i\}_1^n,pp)\leftarrow \mathsf{Setup}(1^\ell,t,n)$

$2:I\leftarrow \mathcal{A}_1(1^\ell,pp,t,n)$

$3:(k_1,k_2)\leftarrow \mathcal{A}_2(1^\ell,pp,\{sk_i\}_{i\in I})$

$4:b\xleftarrow{\$}\{0,1\}$

$5:(aux,\alpha)\leftarrow \mathsf{Prep}(k,t,n)$

$6:$ **For each** $i\in I$ **do**

$\begin{pmatrix} v_i \\ ss_i \end{pmatrix}\leftarrow \mathsf{Share}\left\langle \begin{matrix} \mathcal{S}(k_b,aux) \\ \mathcal{A}(u_i,sk_i) \end{matrix}\right\rangle (pp)$

$7:b^\star\leftarrow \mathcal{A}$

$8:$ **Return** $(b^\star=b)\wedge(|I|<t)$

$\underline{\mathsf{DObliviousness}_{\mathsf{OVTSS},\mathcal{A}}(1^\ell,t,n)}$

$1:(\{sk_i\}_1^n,pp)\leftarrow \mathsf{Setup}(1^\ell,t,n)$

$2:(k,aux,\alpha,u_{0,1},...,u_{0,n},u_{1,1},...,u_{1,n})\leftarrow \mathcal{A}(1^\ell,pp,t,n)$

$3:b\xleftarrow{\$}\{0,1\}$

$4:$ **For each** $i\in[1,n]$ **do**

$\begin{pmatrix} v_i \\ ss_i \end{pmatrix}\leftarrow \mathsf{Share}\left\langle \begin{matrix} \mathcal{A}(k,aux) \\ \mathcal{M}_i(u_{b,i},sk_i) \end{matrix}\right\rangle (pp)$

$5:b^\star\leftarrow \mathcal{A}$

$6:$ **Return** $b=b^\star$

Figure 6: Security definitions of OVTSS

- The prover chooses $m_4\xleftarrow{\$}Z_N$, computes
$$c_4\leftarrow \mathsf{Paillier.Enc}(pk,m_4;r_4),$$
$$c_{2\cdot4}\leftarrow \mathsf{Paillier.Enc}(pk,m_2\cdot m_4;r_{2\cdot4}),$$
and sends $c_4,c_{2\cdot4}$ to the verifier.
- The verifier chooses $a\xleftarrow{\$}Z_N$ and sends it to the prover.
- The prover computes $b=a\cdot m_1+m_4, z_1=r_1^a r_4, z_2=r_2^b(r_{2\cdot4}r_3^a)^{-1}$ to open the encryptions
$$c_1^a c_4\leftarrow \mathsf{Paillier.Enc}(pk,am_1+m_4;r_1^a r_4),$$
$$c_2^b(c_{2\cdot4}c_3^a)^{-1}\leftarrow \mathsf{Paillier.Enc}(pk,0;r_2^b(r_{2\cdot4}r_3^a)^{-1}).$$
- The verifier verifies correctness of the encryption openings in the above step and accepts iff it is correct.

ZKP for ordered-exponential computation on the same base π_{OEC} [36]. A ZKP for the relation that multiple plaintexts are a sequence of powers. Formally, the relation is

$$R_{\mathsf{OEC}}=\{(pk=N,\{cu^{(i)}\in Z_{N^2}\}_{i=1}^{t-1});(u\in Z_N)|$$
$$cu^{(i)}\leftarrow \mathsf{Paillier.Enc}(pk,u^i;r_i)\ \forall i\in[1,t-1]\}.$$

π_{OEC} can be constructed by using π_{PRO} as follows [36]: the prover proves to the verifier that $\forall\, i\in[2,t-1]$, the plaintext corresponding to cu_i is the product of the plaintexts corresponding to cu_1 and cu_{i-1}, i.e.,

$$R_{\mathsf{PRO}}=\{(pk,cu^{(1)},cu^{(i-1)},cu^{(i)});(u^1,u^{i-1},u^i,r_1,r_{i-1},r_i)|$$
$$cu^{(1)}\leftarrow \mathsf{Paillier.Enc}(pk,u^1;r_1)\wedge$$
$$cu^{(i-1)}\leftarrow \mathsf{Paillier.Enc}(pk,u^{i-1};r_{i-1})\wedge$$
$$cu^{(i)}\leftarrow \mathsf{Paillier.Enc}(pk,u^1\cdot u^{i-1};r_i)\}.$$

ZKP for non-zero plaintext π_{NZP}. A ZKP for the relation that the value of the underlying plaintext of a ciphertext is not 0. Formally,

$$R_{\mathsf{NZP}}=\{(pk=N,c_1\in Z_{N^2});(m_1\in Z_N,r_1\in Z_N^*)|$$
$$c_1\leftarrow \mathsf{Paillier.Enc}(pk,m_1;r_1)\wedge m_1\neq 0\}.$$

π_{NZP} can be constructed based on π_{PRO} as follows: The prover chooses $m_2\xleftarrow{\$}Z_N$ s.t. $m_3=m_1\cdot m_2\neq 0$, encrypts m_2,m_3 as $c_2=\mathsf{Paillier.Enc}(pk,m_2;r_2)$, $c_3=\mathsf{Paillier.Enc}(pk,m_3;r_3)$, respectively, sends c_2,c_3 to the verifier, opens the encryption of m_3, and generates π_{PRO} to prove that the underlying plaintext of c_3 is the product of m_1 and m_2. Iff $m_3\neq 0$ and π_{PRO} is valid, $\mathcal{S}$ accepts that $m_1\neq 0$.

The security analysis of π_{PRO} and π_{OEC} can be found in in [35] and [36], respectively, and π_{NZP} is constructed based on π_{PRO}. We do not repeat their security proofs here.

ZKP for non-zero message π_{NZM}. A ZKP for the relation that given a Pedersen's commitment, the value of its underlying message is not 0. Formally,

$$R_{\mathsf{NZM}}=\{(G,p,q,g\in G,h\in G);(\alpha_1\in G,a_1\in Z_q,r_1\in Z_q)|$$
$$\alpha_1\leftarrow \mathsf{Commit.Com}(a_1;r_1)\wedge a_1\neq 0\}.$$

An interactive protocol for R_{NZM} can be constructed as follows, which can be easily converted to be non-interactive.

- The prover chooses $a_2\xleftarrow{\$}Z_q^*, r_2\xleftarrow{\$}Z_q$, computes $\alpha_2=\mathsf{Commit.Com}(a_2;r_2)$, $a_3=a_1\cdot a_2$ and sends a_3 to the verifier.
- Iff $a_3\neq 0$, the verifier continues the following protocol.
- The prover further proves
$$R_1=\{(G,p,q,g,h,\alpha_1,\alpha_2,a_3);(a_1,a_2,r_1,r_2)|$$
$$\alpha_1\leftarrow \mathsf{Commit.Com}(a_1;r_1)\wedge$$
$$\alpha_2\leftarrow \mathsf{Commit.Com}(a_2;r_2)\wedge a_3=a_1\cdot a_2\}.$$
We observe that
$$R_1\Leftrightarrow R_2=\{(G,p,q,g,h,\alpha_1,\alpha_2,a_3);(a_1,r_1,r_2)|$$
$$\alpha_1\leftarrow \mathsf{Commit.Com}(a_1;r_1)\wedge$$
$$\alpha_2\leftarrow \mathsf{Commit.Com}(a_3\cdot a_1^{-1};r_2)\}.$$
With this observation, we can implement a Sigma protocol for R_2 as follows.
  - The prover chooses $t_a,t_r,t_r'\xleftarrow{\$}Z_q$, computes $\alpha_t=g^{t_a}h^{t_r}$, $\alpha_t'=g^{a_1\cdot t_a}h^{t_r'}$ and sends $\{\alpha_t,\alpha_t'\}$ to the verifier.

– The verifier chooses $c \xleftarrow{\$} Z_q$ and sends it to the prover.
– The prover computes $z_a = t_a + c \cdot a_1, z_r = t_r + c \cdot r_1, z'_r = r_2 \cdot t_a + c \cdot a_1 \cdot r_2 - t'_r$ and sends $\{z_a, z_r, z'_r\}$ to the verifier.
– The verifier verifies $\mathsf{Commit.Com}(z_a; z_r) \stackrel{?}{=} \alpha_t \cdot (\alpha_1)^c$ and $(\alpha_2)^{z_a} \stackrel{?}{=} \mathsf{Commit.Com}(ca_3; z'_r) \cdot \alpha'_t$.

The Completeness of the above Sigma protocol for R_2 is clear, and we prove its Special soundness, and Special honest verifier zero-knowledge (SHVZK) as follows.

Special soundness. Given two accepting conversations $((\alpha_t, \alpha'_t), c, (z_a, z_r, z'_r))$ and $((\alpha_t, \alpha'_t), c^\star, (z^\star_a, z^\star_r, z'^\star_r))$ for the statement (α_1, α_2), where $c \neq c^\star$. It is efficient to compute $a_1 = (z^\star_a - z_a)/(c^\star - c), r_1 = (z^\star_r - z_r)/(c^\star - c), r_2 = (z^\star_r - z_r)/(z^\star_a - z_a)$.

SHVZK. Given challenge $c \xleftarrow{\$} Z_q$, choose $z_a, z_r, z'_r \xleftarrow{\$} Z_q$ and compute $\alpha_t = \alpha_z/(\alpha_1)^c, \alpha'_t = (\alpha_2)^{z_a}/\mathsf{Commit.Com}(ca_3; z'_r)$. Clear, it is an accepting conversation. Moreover, in the real conversation, c, z_a, z_r, z'_r are mutually independent and all uniformly distributed over Z_q. Given (c, z_a, z_r, z'_r), α_t and α'_t are uniquely determined. It should be clear that this is the same as the distribution of the simulated output.

ZKP for same derivation π_{SD}. A ZKP for the relation that the exponent of an exponentiation in the group G is the corresponding plaintext of a Paillier ciphertext. Formally, $R_{\mathsf{SD}} = \{(N, cu \in Z_{N^2}, G, g \in G, U \in G); (u, R) | cu = (1+N)^u \cdot R \bmod N^2 \wedge U = g^u\}$.

- The prover chooses $s \xleftarrow{\$} Z_q, r_s \xleftarrow{\$} Z^*_N$, computes $S = g^s, cs = (1+N)^s \cdot r_s^N \bmod N^2$ and sends $\{S, cs\}$ to the verifier.
- The verifier chooses $c \xleftarrow{\$} Z_q$ and sends it to the prover.
- The prover computes $z = s + c \cdot u, R_z = R^c \cdot r_s^N \bmod Z_N^2$ and sends (S, cs, z, R_z) to the verifier.
- The verifier verifies the proof by checking $g^z \stackrel{?}{=} S \cdot U^c$ and $(cu)^c \cdot cs \stackrel{?}{=} (1+N)^z \cdot R_z \bmod N^2$.

The Completeness of the above Sigma protocol for π_{SD} is clear, and we prove its Special soundness, and Special honest verifier zero-knowledge (SHVZK) as follows.

Special soundness. Given two accepting conversations (S, cs, c, z, R_z) and (S, cs, c', z', R'_z), it is efficient to compute $u = (z' - z)/(c' - c)$ and $R = (R'_z/R_z)^{(c'-c)^{-1}}$.

SHVZK. Given challenge $c \xleftarrow{\$} Z_q$, choose $z \xleftarrow{\$} Z_q$ and $R_z \xleftarrow{\$} Z^2_N$. Then, compute $S = g^z/U$ and $cs = ((1+N)^z \cdot R_z/cs)^{c^{-1}}$. This always yields an accepting conversation. With the randomness of $c, z \in Z_q$ and $R_z \in Z^2_N$, it is clear that the distribution of the simulated output is the same as the one in real conversation.

C SECURITY ANALYSIS OF DATAUBER

We analyze the security of DataUber regarding time-locked confidentiality, mailman fairness, and reporter fairness. As we have analyzed the security of the ZKP schemes used in DataUber before, we model them as "ideal functionalities" shown in Fig. 7.

C.1 Time-locked confidentiality

Theorem 5. *If Pedersen's commitment [50] is unconditional hiding, π_{OEC} and π_{NZP} are modeled as the ideal functionalities $\mathcal{F}_{\mathsf{OEC}}$ and $\mathcal{F}_{\mathsf{NZP}}$ presented in Fig. 7, respectively, then for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$*

$$|\Pr[\text{TL-CONF}_{\mathcal{A}}(1^\ell) \Rightarrow 1]| \leq \frac{1}{2} + \mathrm{negl}(1^\ell).$$

Proof. To prove Theorem 5, we introduce a series of games presented in Fig. 8 and elaborate on them below.

G_0^{TLC} is identical with $\text{TL-CONF}_{\mathcal{A}}$ except that ZKP schemes are replaced with ideal functionalities in G_0^{TLC},

$$|\Pr[G_0^{\mathrm{TLC}} \Rightarrow 1] - \Pr[\text{TL-CONF}_{\mathcal{A}} \Rightarrow 1]| \leq \mathrm{negl}(1^\ell).$$

G_1^{TLC} is identical with G_0^{TLC}, except that the secret shares $ss_i (i \in I)$ are computed directly using u_i. Due to the correctness of Paillier encryption, $\mathcal{F}_{\mathsf{NZP}}$, and $\mathcal{F}_{\mathsf{OEC}}$, ss_i in G_1^{TLC} is same as the one in G_0^{TLC} for each $i \in I$. Then, $\Pr[G_1^{\mathrm{TLC}} \Rightarrow 1] = \Pr[G_0^{\mathrm{TLC}} \Rightarrow 1]$.

G_2^{TLC} is identical with G_1^{TLC}, except that $ss_i (i \in I)$ is replaced with random strings in G_2^{TLC}. Since $a_1, ..., a_{t-1}$ are random and independent with k_b, ss_i leaks nothing about k_b. We have $\Pr[G_2^{\mathrm{TLC}} \Rightarrow 1] = \Pr[G_1^{\mathrm{TLC}} \Rightarrow 1]$.

Now we prove that

$$|\Pr[G_2^{\mathrm{TLC}} \Rightarrow 1] - \Pr[\mathrm{Hiding}_{\mathcal{A}',\mathsf{Commit}} \Rightarrow 1]| \leq \mathrm{negl}(1^\ell),$$

where $\mathrm{Hiding}_{\mathcal{A}',\mathsf{Commit}}$ is the hiding game and $\mathcal{A}'$ is a PPT adversary. $\mathcal{A}'$ is given given input (G, p, q, g, h) and oracle access to some function O, and its goal is to determine whether the returned commitment is computed using k_0 or k_1. In detail:

(1) $\mathcal{A}$ chooses (k_0, k_1) and sends them to $\mathcal{A}'$.
(2) $\mathcal{A}'$ queries O on (k_0, k_1). O chooses $b \xleftarrow{\$} \{0,1\}, r \xleftarrow{\$} Z^*_q$, computes $\alpha_b \leftarrow g^{k_b} h^r$ and sends α_b to $\mathcal{A}'$.
(3) $\mathcal{A}'$ chooses $a_i, r_i \xleftarrow{\$} Z^*_q$, computes $\alpha_i = g^{a_i} \cdot h^{r_i}, \forall i \in [1, t-1]$ and sends $\{\alpha, \alpha_1, ..., \alpha_{t-1}\}$ to $\mathcal{A}$.

$\mathcal{F}_{\mathsf{OEC}}^{(pk, \{cu^{(i)}\}_1^{t-1})}$
On receiving $(u, \{r_i\}_1^{t-1})$ from $\mathcal{P}$, if
$cu^{(i)} \leftarrow \mathsf{Paillier.Enc}(pk, u^i; r_i) \forall i \in [1, t-1]$,
return 1, otherwise, return 0.

$\mathcal{F}_{\mathsf{NZP}}^{(pk, c)}$
On receiving (m, r) from $\mathcal{P}$, if
$c \leftarrow \mathsf{Paillier.Enc}(pk, m; r) \wedge m \neq 0$,
return 1, otherwise, return 0.

$\mathcal{F}_{\mathsf{NZM}}^{(G,p,q,g,h)}$
On receiving $(\alpha_{t-1}, a_{t-1}, r_{t-1})$ from $\mathcal{P}$, if
$\alpha_1 = g^{a_{t-1}} h^{r_{t-1}} \wedge a_{t-1} \neq 0$,
return 1, otherwise, return 0.

$\mathcal{F}_{\mathsf{SD}}^{(pk, cu, G, g, U)}$
On receiving (u, r^N) from $\mathcal{P}$, if
$cu \leftarrow \mathsf{Paillier.Enc}(pk, u; r) \wedge U = g^u$,
return 1, otherwise, return 0.

Figure 7: Ideal functionalities of ZKP schemes

$G_0^{\mathrm{TLC}}(G,p,q,h,g)$

$1: sk_i \xleftarrow{\$} Z_q, pk_i = g^{sk_i}$

$2: I \leftarrow \mathcal{A}_1(\{pk_i\}_1^n, G, p, q, h, g)$ s.t. $|I \cap [1,n]| < t$

$3: (k_0, k_1) \leftarrow \mathcal{A}_2(\{sk_i\}_{i\in I})$

$4: b \xleftarrow{\$} \{0,1\}, a_1, ..., a_{t-1}, r_0, ..., r_{t-1} \xleftarrow{\$} Z_q^*$

$5: \alpha_0 \leftarrow \mathsf{Commit.Com}(k_b; r_0)$

$\alpha_i \leftarrow \mathsf{Commit.Com}(a_i; r_i)\ \forall i \in [1,n]$

$6:$ **For each** $i \in I: (u_i, \{cu_i^{(j)}\}_{j=1}^{t-1}) \leftarrow \mathcal{A}(\alpha_0, ..., \alpha_{t-1})$

s.t. $\mathcal{F}_{\mathsf{NZP}}(pk_i, cu_i^{(1)}, u_i) = 1 \wedge \mathcal{F}_{\mathsf{OEC}}(pk_i, \{cu_i^{(j)}\}_{j=1}^{t-1}) = 1$

$7: css_i \leftarrow \mathsf{Paillier.Enc}(pk_i, k_b) \cdot \prod_{j=1}^{t-1}(cu_i^{(j)})^{a_j}$

$8: ss_i \leftarrow \mathsf{Paillier.Dec}(sk_i, css_i)$

$9: b^\star \leftarrow \mathcal{A}(\{ss_i\}_{i\in I}, \{\alpha_i\}_0^{t-1})$

$G_1^{\mathrm{TLC}}(G,p,q,g,h)$ $\boxed{G_2^{\mathrm{TLC}}(G,p,q,g,h)}$

$1: sk_i \xleftarrow{\$} Z_q, pk_i = g^{sk_i}$

$2: I \leftarrow \mathcal{A}(\{pk_i\}_1^n, G, p, q, h, g)$ s.t. $|I \cap [1,n]| < t$

$3: (k_0, k_1) \leftarrow \mathcal{A}(\{sk_i\}_{i\in I})$

$4: b \xleftarrow{\$} \{0,1\}, a_1, ..., a_{t-1}, r_0, ..., r_{t-1} \xleftarrow{\$} Z_q^*$

$5: \alpha_0 \leftarrow \mathsf{Commit.Com}(k_b; r_0)$

$\alpha_i \leftarrow \mathsf{Commit.Com}(a_i; r_i)\ \forall i \in [1,n]$

$6:$ **For each** $i \in I:$

$7:$ $u_i \neq 0 \leftarrow \mathcal{A}(\alpha_0, ..., \alpha_{t-1})\ \forall i \in I$

$8:$ $ss_i = k_b + \sum_{j=1}^{t-1} a_j \cdot u_i \bmod Z_q$ $\boxed{ss_i \xleftarrow{\$} Z_q}$

$9: b^\star \leftarrow \mathcal{A}(\{ss_i\}_{i\in I}, \{\alpha_i\}_0^{t-1})$

$\mathrm{Hiding}_{\mathcal{A}',\mathsf{Commit}}(1^\ell)$

$1: (G,p,q,g,h) \leftarrow \mathsf{Commit.Gen}(1^\ell)$

$2: (k_0, k_1) \leftarrow \mathcal{A}'(G,p,q,g,h)$

$3: b \xleftarrow{\$} \{0,1\}$

$4: r \xleftarrow{\$} Z_q^*, \alpha_b \leftarrow g^{k_b} h^r$

$5: b^\star \leftarrow \mathcal{A}'(\alpha_b)$

$6:$ **Return** $(b = b^\star)$

Figure 8: Game used in the proof of Theorem 5

(4) After $\mathcal{A}$ outputs $b^\star$, $\mathcal{A}'$ sends $b^\star$ to O.

The view of $\mathcal{A}$ when run as a subroutine by $\mathcal{A}'$ in the above procedure is identical with the view of $\mathcal{A}$ in G_2^{TLC}. Therefore, $|\Pr[G_2^{\mathrm{TLC}} \Rightarrow 1] - \Pr[\mathrm{Hiding}_{\mathcal{A}',\mathsf{Commit}} \Rightarrow 1]| \leq \mathrm{negl}(1^\ell)$.

Since Pedersen's commitment is unconditional hiding, $\Pr[\mathrm{Hiding}_{\mathcal{A}',\mathsf{Commit}} \Rightarrow 1] = \frac{1}{2}$.

This concludes the proof. □

C.2 Mailman fairness

Theorem 6. *Let* $\Pr[\mathrm{Solve}_G]$ *denote the probability of solving a DL problem in* G*, if* π_{NZM} *is modeled as the ideal functionality* $\mathcal{F}_{\mathsf{NZM}}$ *shown in Fig. 7, then for any PPT adversary* $\mathcal{A}$

$$\Pr[\text{HM-FAIR}_{\mathcal{A}} \Rightarrow 1] \leq 2\Pr[\mathrm{Solve}_G] + \mathrm{negl}(1^\ell).$$

Proof. To prove Theorem 6, we introduce a series of games presented in Fig. 9 and elaborate on them below.

$G_0^{\mathrm{HMF}}(G,p,q,g,h,N)$

$1: sk \xleftarrow{\$} Z_q^*, pk = g^{sk}$

$2: (k, \{\alpha_i\}_0^{t-1}, r_0, \{r_i, a_i\}_1^{t-1}) \leftarrow \mathcal{A}(G,p,q,g,h,pk)$

s.t. $\mathcal{F}_{\mathsf{NZM}}^{(G,p,q,g,h)}(\alpha_{t-1}, a_{t-1}, r_{t-1}) = 1$

$3:$ randomly choose $u \neq 0$ s.t. $(u^t - 1)/(u - 1) \leq (pk/q)$

$4: cu^{(i)} \leftarrow \mathsf{Paillier.Enc}(pk, u^i)\ \forall i \in [1, t-1]$

$5: (css, cv) \leftarrow \mathcal{A}(\{cu^{(i)}\}_1^{t-1})$

$6: ss \leftarrow \mathsf{Paillier.Dec}(sk, css), v = \mathsf{Paillier.Dec}(sk, cv)$

$7: \alpha_0 = g^k h^{r_0}, \alpha_i = g^{a_i} h^{r_i}\ \forall i \in [1, t-1]$

$8:$ **If** $g^{ss}h^v \neq \cdot \prod_{i=0}^{t-1}(\alpha_i)^{u^i} \vee \prod_{i=0}^{t-1} \alpha_i / g^{ss} = h^v$ **abort**

$9: S = g^{ss}$

$10: ss^\star \leftarrow \mathcal{A}(\{cu^{(i)}\}_1^{t-1}, S)$

$11:$ **Return** $(ss = ss^\star)$

$G_1^{\mathrm{HMF}}(G,p,q,g,h,N)$

$1: sk \xleftarrow{\$} Z_q^*, pk = g^{sk}$

$2: (k, r_0, \{r_i, a_i\}_1^{t-1}) \leftarrow \mathcal{A}(G,p,q,g,h,pk)$

$3:$ **If** $a_1 = \cdots = a_{t-1} = 0$ **abort**

$4:$ randomly choose $u \neq 0$ s.t. $(u^t - 1)/(u - 1) \leq (pk/q)$

$5: cu^{(i)} \xleftarrow{\$} Z_{N^2}\ \forall i \in [1, t-1]$

$6: ss = k + \sum_{i=0}^{t-1} a_i \cdot u^i, v = \sum_0^{t-1} r_i \cdot u^i, S = g^{ss}$

$7: (ss^\star, v^\star) \leftarrow \mathcal{A}(\{cu^{(i)}\}_0^{t-1}, S)$

$8:$ **Return** $(g^{ss}h^v = g^{ss^\star}h^{v^\star})$

Figure 9: Game used in the proof of Theorem 6

First, as G_0^{HMF} is identical with HM-FAIR, $\Pr[\text{HM-FAIR} \Rightarrow 1] = \Pr[G_0^{(\mathrm{HMF})} \Rightarrow 1]$. In G_0^{HMF}, the 7-th step ensures the correctness of ss_i and that not $a_1, ..., a_{t-1}$ are all 0. Therefore, G_1^{HMF} is identical with G_0^{HMF}, except that $cu^{(i)}$ is replaced with a random element in the ciphertext space. With the security of Paillier encryption,

$$|\Pr[G_1^{(\mathrm{HMF})} \Rightarrow 1] - \Pr[G_0^{(\mathrm{HMF})} \Rightarrow 1]| \leq \mathrm{negl}(1^\ell).$$

Now we prove that $\Pr[G_1^{(\mathrm{HMF})} \Rightarrow 1] \leq 2\Pr[\mathrm{Solve}_G]$.

We use $\mathcal{A}$ to construct $\mathcal{A}'$ for the DL problem. $\mathcal{A}'$ is given two generators g, h of a group G, and its goal is to output the discrete logarithm of h to the base g, i.e. $d = \log_g h$.

In detail: $\mathcal{A}'$ is given (G,p,q,g,h), where p and q are primes and $q|p-1$, G is a group with the order q, g and h are two random generators of G.

(1) Run $\mathcal{A}$. $\mathcal{A}'$ gives (G,p,q,g,h) to $\mathcal{A}$.

$\underline{G^{RF}(G,p,q,g)}$
$1: Q_H := \emptyset, Q_{Wit} := \emptyset$
$2: ss \xleftarrow{\$} Z_q, S = g^{ss}$
$3: (z', R', Add') \leftarrow \mathcal{A}^{HO, WitO}(S, g, p, q, g)$
$4:$ **If** $(R' \cdot S^{HO(S,R',Add')} = g^{z'}) \wedge (Q_{Wit}[Add'] \neq 1)$
$5:$ **Return** 1
$6:$ **Return** 0

$\underline{WitO(Add)}$
$1: r \xleftarrow{\$} Z_q, R = g^r$
$2: z = r + ss \cdot HO(S, R, Add)$
$3: Q_{Wit}[Add] = 1$
$4:$ **Return** (R, z)

$\underline{HO(S,R,Add)}$
$1:$ **If** $(S,R,Add) \notin Q \quad y \xleftarrow{\$} Z_q, Q[S,R,Add] = y$
$2:$ **Else** $y = Q_H[S,R,Add]$
$3:$ **Return** y

Figure 10: Game used in the proof of Theorem 8

(2) When $\mathcal{A}$ chooses $\{k, r_0, \{r_i, a_i\}_1^{t-1}\}$, $\mathcal{A}'$ ensures that not $a_1, ..., a_{t-1}$ are all 0. Otherwise, $\mathcal{A}'$ aborts.
(3) $\mathcal{A}'$ chooses $u \xleftarrow{\$} Z_N, cu^{(i)} \xleftarrow{\$} Z_{N^2}\ \forall i \in [1, t-1]$ and sends $\{cu^{(i)}\}_1^{t-1}, S = g^{ss}$ to $\mathcal{A}$.
(4) Whenever $\mathcal{A}$ outputs $(ss^\star, v^\star)$, $\mathcal{A}$ verifies whether $ss^\star = ss$. If it is, $\mathcal{A}$ outputs a random $d' \in Z_p$. Otherwise, $\mathcal{A}$ outputs $d' = (ss - ss^\star)/(v^\star - v)$.

In the above steps, if $\mathcal{A}$ outputs a $v^\star$ for $ss^\star \neq ss$, then $ss^\star - ss \neq 0, v^\star - v \neq 0$, and $g^{ss^\star} h^{v^\star} = g^{ss} h^v$.

Thus, we have $g^{ss^\star} h^{v^\star} = g^{ss} \cdot g^{ss^\star - ss} h^v \cdot h^{v^\star - v}$, where $ss^\star - ss + (\log_g h) \cdot (v^\star - v) = 0$. With this equation,

$$d' = \frac{ss - ss^\star}{v^\star - v} = \frac{(\log_g h) \cdot (v^\star - v)}{v^\star - v} = \log_g h.$$

The view of $\mathcal{A}$ when run as a subroutine by $\mathcal{A}'$ is identical to the view of $\mathcal{A}$ in $G_1^{(\mathrm{HMF})}$. Therefore, $\Pr[G_1^{(\mathrm{HMF})} \Rightarrow 1] = \Pr[\log_g h = d' \leftarrow \mathcal{A}'^{(G,p,q,g,h)}] + \Pr[ss \leftarrow \mathcal{A}(S = g^{ss})] \leq 2\Pr[\mathrm{Solve}_G]$.

This concludes the proof. □

Theorem 7. *If the smart contract used in DataUber always executes its code and π_{SD} is modeled as the ideal functionality $\mathcal{F}_{\mathrm{SD}}$ presented in Fig. 7, DataUber achieves lazy mailman fairness and malicious mailman fairness.*

Proof. According to the logic behind the code of *TimedPub*, only after a mailman invokes the function *PubShare* to publish her/his share in the prescribed time slot, will the service charge be transferred to the mailman. If a mailman receives the service charge from *TimedPub* without invoking the *PubShare*, the execution of *TimedPub* does not follow its code. This conflicts with our security assumption that the smart contract always executes its code. Analogously, malicious mailman fairness is also guaranteed by the smart contract. In DataUber, the verification of complaints and punishments both rely on *TimedPub*. In the logic behind the code of the function *Complaint*, if a valid proof of a premature leakage is submitted, *TimedPub* will reassign the deposit of the malicious mailman to the sender and the reporter. Moreover, the ideal functionality $\mathcal{F}_{\mathrm{SD}}$ ensures that a mailman can only upload a commitment that is indeed corresponding to its share. In this case, the mailman cannot leak its true share while getting away with it due to the invalid commitment recorded in the contract. □

C.3 Reporter fairness

Theorem 8. *If the hash function H used in $\pi_{\mathrm{DL}}^{(ID)}$ is modeled as a random oracle HO, then* $\Pr[\text{R-FAIR}_{\mathcal{A}} \Rightarrow 1] \leq \mathrm{negl}(1^\ell)$.

Proof. R-FAIR is identical with G^{RF}, then $\Pr[\text{R-FAIR}_{\mathcal{A}} \Rightarrow 1] = \Pr[G^{RF}(G,p,q,g) \Rightarrow 1]$. Now we prove that $\Pr[G^{RF}(G,p,q,g) \Rightarrow 1] \leq \mathrm{negl}(1^\ell)$.

We use $\mathcal{A}$ to construct a PPT adversary $\mathcal{A}'$ to forge a proof for the statement $S = g^{ss}$ without knowing ss. $\mathcal{A}'$ is given access to some oracle O, when $\mathcal{A}'$ queries O on some challenge c, O returns (z, R, c) such that $g^z = R \cdot S^c$. Finally, if $\mathcal{A}'$ outputs (z', R', c') s.t. $g^{z'} = R' \cdot S^{c'}$, $\mathcal{A}'$ wins. In detail:

(1) Initialize empty lists L_H, L_{Wit}.
(2) Run $\mathcal{A}$. Whenever $\mathcal{A}$ queries $WitO$ on Add, $\mathcal{A}'$ stores Add in L_{Wit}, chooses $c \xleftarrow{\$} Z_q$ and queries O on c.
(3) O chooses $r \xleftarrow{\$} Z_q$, computes $z = r + c \cdot ss, R = g^r$, and returns (z, R) to $\mathcal{A}'$.
(4) After receiving the response (z, R) from O, $\mathcal{A}'$ stores (R, Add, c) in L_H and gives (z, R, Add) to $\mathcal{A}$.
(5) Whenever $\mathcal{A}$ queries HO on $(R^\star, Add^\star)$, if there is a item $(R^\star, Add^\star, c^\star)$ in L_H, $\mathcal{A}$ returns $c^\star$. Otherwise, $\mathcal{A}$ chooses $c^\star \xleftarrow{\$} Z_q$, stores $(R^\star, Add^\star, c^\star)$ in L_H, and returns $c^\star$ to $\mathcal{A}$.
(6) Finally, when $\mathcal{A}'$ outputs (z', R', Add'), $\mathcal{A}$ looks up L_H to find the item (R', Add', c') and outputs (z', R', c').

In the above procedure, $\mathcal{A}$'s view when run as a subroutine by $\mathcal{A}'$ is identical to that in G^{RF}. If $\mathcal{A}$ can output a valid proof (z', R', Add') s.t. $c' = HO(S, R', Add')$ and $g^{z'} = R' \cdot S^{c'}$, where Add' has not been queried on $WitO$, $\mathcal{A}'$ can output a valid proof (z', R', c') s.t. $c' = L_H[R'], g^{z'} = R' \cdot S^{c'}$. Let Forge denote the event that $\mathcal{A}'$ outputs a valid proof, $\Pr[G^{RF} \Rightarrow 1] \leq \Pr[\mathrm{Forge}]$. In the above game, the goal of $\mathcal{A}'$ is essentially to forge a proof for a statement in a NIZKP protocol converted from a sigma protocol using Fiat-Shamir transform [28]. Therefore, we have $\Pr[\mathrm{Forge}] \leq \mathrm{negl}(1^\ell)$.

This concludes the proof. □