Blockchain-based Privacy-preserving Fair Data Trading Protocol

Yao Jiang Galteland* and Shuang Wu

Norwegian University of Science and Technology, NTNU, Norway

{yao.jiang, shuang.wu}@ntnu.no

Abstract

Fair data trading online is a challenging task when there is mistrust between data providers and data collectors. The trust issue leads to an unsolvable situation where the data collector is unwilling to pay until she receives the data while the data provider will not send the data unless she receives the payment. The traditional solutions toward fair data trading rely on the trust-third party. After the emergence of the blockchain, many researchers use a smart contract on blockchain as a trust-less third party to address the mistrust deadlock. However, involving a smart contract in the protocol inevitably exposes some information to the public if the smart contract is on public blockchain cryptocurrency systems. We observe that the existing fair data trading protocols do not take privacy into account, which, for instance, is critical when trading the sensitive data or the players simply do not want to leak any information about the tradings on the public blockchain. In this paper, we construct a fair trading protocol based on a smart contract that provides better privacy to the participants. We introduce new security notions for privacy-preserving blockchain-based fair data trading protocol and prove our protocol is secure under our new notions. Furthermore, we give a prototype implementation on Ethereum smart contract.

1 Introduction

Trading data on the internet is important for digital ecosystems, especially nowadays a large amount of electronic data are being generated, forming a large digital resource for different usage. A lot of specific data have economic values and being used for commercial purposes. A fair data trading means that at the end of a trading process, either the data provider gets the payment for its data and the data collector gets the data it expects or neither of them succeeds to receive what they want. Achieving fairness for online trading is not an easy task since the trading might be carried out over insecure networks, adversaries might corrupt the systems that are used by the players. In these electronic commerce scenarios, neither of the trading players trust the other, thus the process ends up in a deadlock problem, where the data collector is unwilling to pay until it receives the data while the data provider will not send the data unless it receives the payment.

There are various solutions to the fair data trading problem in the literature. The early solutions [ASW98, CD00, DR03, HYWS08, KL10] are based on trusted third party (TTP) or a trusted arbiter that only shows up to address disputes. Even though [PD99] proves that it is impossible to solve fair exchange without a TTP, the emergence of the blockchain can replace the TTP by a trust-less smart contract that holds no secrets and always executes the protocol honestly. Blockchain suits this role because of its decentralisation, transparency and financial properties. Getting benefits from this observation, recent solutions [CGGN17, DEF18a, EFS20] are based on blockchain/smart contract. These solutions are tailored to a data trading case where the data buyer is willing to pay for some data x that satisfied a public function ϕ such that $\phi(x) = 1$. For instance, when a data provider and a data collector are trading a movie, the data collector can find the hash of the movie from some public database beforehand and later verify the file it received is indeed the pre-image of the hash.

*This work has been co-funded by the IKTPLUSS program of the Research Council of Norway under the scope of and as part of the outcome from the research project Reinforcing the Health Data Infrastructure in Mobility and Assurance through Data Democratization (Health Democratization, 2019 – 2024, project number 288856).

These protocols share the same communication model where data provider firstly sends encrypted data to the data collector off-chain, then later data provider sends the encryption key to the blockchain and exchange for payment. We notify that there are two challenging tasks in this model: first, proving the encrypted data indeed is an encryption of desired data, i.e. a way to verify the validity of the encrypted data; second, proving the key material that sent to blockchain is valid (valid means the key material can truly help with decrypting the encrypted data), i.e. a way to check the validity of key material. The protocols proposed by [CGGN17] use zero-knowledge proof and protocols [DEF18a, EFS20] use a technique called proof of misbehaviour to solve the above two challenges.

While observed that all these blockchain-based solutions require to record some information about the trading activities on the public blockchain, few of them mention the potential privacy issues it might cause. For instance, if an adversary can identify different smart contrasts are trading the same data or they are from the same person, it can build a wide 'trading flow' of a certain data provider. The linkability between different data tradings can cause severe affects, especially, when it comes to tradings with sensitive data. Given medical data trading as an example, from a single medical data leakage, a malicious interested entity, such as an insurance company, identifies a specific medical record flows to other destinations where the data collectors are targeting patients who have particular health conditions, the insurance company can make an educated guess about the health condition of the data provider. Furthermore, the existing solutions are targeting a limited use case where the validation of the trading data can be evaluated by a public function ϕ, this function can be found before executing the fair exchange protocol. However, when comes to sensitive data trading, such as medical data, data generated from personal IoT devices, certified emails and so on, the validation of these data cannot be evaluated simply by a function, moreover, no public information will be given before trading in most of the cases. Therefore, a privacy-preserving fair exchange protocol is needed for trading sensitive data, we aim to solve the aforementioned two tasks without compromising privacy.

1.1 Overview

In this paper, we construct a novel fair data trading (FDT) protocol that provides better privacy (confidentiality of the trading data and unlinkability between different tradings) and the protocol is suitable for trading sensitive data. We follow the communication model in the previous works where the data provider sends encrypted data to the data collector off-chain, and then sends only key materials that can be used to decrypt the encrypted data on-chain for exchanging cryptocurrencies. While in our system, the published key materials will not reveal any information about the traded data, and the different trading instances are indistinguishable from each other, i.e. the public information is unlinkable, we call it *public unlinkability*.

We observe that in the case of trading sensitive data, it is nature to find an authority that can authorise the truth of the data. For example, the hospital can verify the truth of a patient's medical records; the email server can verify the origination and destination of an email message. Therefore we build our system on the assumption of the existence of a trusted authority. This authority can give authorisation to the truth of the data along with its encryption. We call such authority as *data manager*. Note that the role of a data manager is different from the TTP in traditional fair exchange protocols, because it is only involved in pre-trading phase, we do not rely on it to perform any actual trading procedure with the data collector. Introducing such a data manager trivially solve the first challenge (verify the encrypted data), however, it is not reasonable to bother the data manager to do authorisation for every single trading. We intend to minimise the participation of the data manager, namely, with one time authorisation, a data provider can trade for many times and still keep fairness, confidentiality and unlinkability among those tradings.

Technically speaking, our protocol is inspired from proxy re-encryption (PRE), which provides a solution to trade data such that the data provider can generate a re-encryption key (re-key) to let the data collector re-fresh the original ciphertext to a ciphertext under its own secret key. This re-key is expected to not reveal anything about the trading message. We build an extra functionality beyond PRE that allows one to verify the validity of the re-key but without reliving any information about the original data (zero-knowledge proof), which solves the second challenging task, that is verifying the re-key. In addition, the re-key can be randomised in each trading, which achieves public unlinkability.

Work Flow. Our fairing data trading protocol involves two independent phases: pre-trading and trading phase. The pre-trading phase is done before the actual data trading phase, that is, a data provider gets her data encrypted and authenticated by a data manager. In the trading phase, firstly, a data collector publishes a smart contract on blockchain. The smart contract specifies the price for the data and the data collector deposit the money. A data provider who has such type of data may register her address to receive rewards on the smart contract and send the encrypted data (the ciphertext is authorized by data manager in the pre-trading phase) to the data collector in a confidential channel, the data collector verify the data and confirm to start a data trading with this data provider, the data collector in this stage also sends blind statements to the smart contract, which is later used to verify the correctness of the rekey. In the end, the data provider submits a re-key that allows the data collector to transfer the ciphertext that is encrypted by the data provider's key to a ciphertext that is encrypted by the data collector's key. The smart contract check if the re-key is valid, if so, it sends rewards to the data provider and records the re-key on the smart contract, afterwards the data collector can read the re-key from the smart contract and obtain the expected data.

1.2 Contributions

Our first contribution is constructing a first-of-its-kind privacy-preserving decentralised fair data trading protocol, which provides fair exchange and privacy. Our second contribution is defining four security notions for fair data trading protocols: message real or random indistinguishability (M-RoR), public real or random indistinguishability (Pub-RoR), signature unforgeability (SigUF) and re-key unforgeability (RkUF). Furthermore, we define fairness, and prove if our protocol satisfies M-RoR then our protocol is fair to data subject and if our protocol satisfies SigUF (and soundness) then our protocol is fair to data collector. Our third contribution is providing a concrete construction for fair data trading protocol, our protocol is named by PFDT and we prove the PFDT protocol achieves all security requirements we discussed above. We also give a proof-of-concept implementation of our protocol in Ethereum and give its evaluation.

1.3 Applications of Our Protocol

Monetising Medical Data Trading. Researches [Hal19, JWCBK15] show that building economy business models that provide incentive structures to facilitate medical data collection is beneficial. Patients who have legal rights can monetise their medical data and get economic benefits from trading it. With blockchain and our privacy-preserving fair data exchange protocol, various consent model sand payment for data flows can be built without the needs for a central bank and a centralised trading platform. In this application case, the data manager is a hospital (or any other healthcare services), data collector can be any entities (pharmaceutical industries, government organisations, healthcare professionals or even individuals) who need personal data for their development.

Blockchain-based Certified Email. Certified email provides the email sender with a mailing receipt and electronic verification that an article was delivered to the receiver. The fair exchange deadlock faced by this service is that the receiver refuses to provide the proof of receipt until the message is delivered while the sender refuses to send the message if there is no guarantee that the receiver will provide the proof of receipt. In this application case, the data manager is the email server, the sender can use our protocol to get certified email where the key materials that sent to blockchain can be consider as receipt.

1.4 Related Work

Proxy Re-Encryption (PRE). Proxy re-encryption was introduced by Blaze, Bleumer, and Strauss in 1998 [BBS98]. It is a public key encryption with an additional function such that a proxy can re-encrypt a ciphertext under one public key to another public key. Proxy re-encryption has been studied extensively in the past few decades.

Symmetric Key v.s. Public Key. *Symmetric-key* proxy re-encryption [BLMR13, CH07, SNS11] is a variant of PRE where the proxy can translate a ciphertext to a different symmetric key. However, the re-key

generation requires both symmetric keys of the data provider and the data collector as the input, which is not desired in our problem setting. It is not reasonable and practical for the data provider to have the data collector's private key. *Public key* proxy re-encryption [ABH09, LV11] allows the data provider generate a re-key by only using the data collector's public key, which is suitable in our scenario.

Ciphertext-Independent v.s. Ciphertext-Dependent. If the re-key generation is independent from ciphertext, we call such PRE is *ciphertext-independent*. If the re-key generation depends on the ciphertext to be refreshed then the PRE is *ciphertext-dependent*. For ciphertext-independent PRE schemes, a single re-key can be used to refresh all ciphertexts from a data provider to a data collector. Additionally, if a cloud is involved in storing data, then the data provider needs to download the ciphertext before generating a re-key. Hence, in terms of bandwidth, ciphertext-independent PRE schemes are considerably more efficient than ciphertext-dependent PRE schemes. Most recent works [ABH09, BLMR13, CH07, LV11, SNS11] focus on constructing ciphertext-independent proxy re-encryption schemes. However, ciphertext-independent PRE implies a problem that if the data collector has a re-key from a data provider then it can open all potential ciphertexts from the data provider, which is not desired in our setting. If the data collector has a re-key from a data provider, next time it buy data from the same data provider, it can reject the payment and gain the data by the help of the previously received re-key. In Section 6, we construct a suitable ciphertext dependent public key PRE to solve our trading scenario.

1.5 Organization

The preliminaries of hard problems and background of signature scheme, proxy re-encryption and zero knowledge proof are given in Section 2. In Section 3 we introduce a generic protocol for fair data trading, the formal definition is provided in Section 5. We define our new security notions in Section 4, and prove our concrete construction is secure under the new notions in Section 6. In Section 7 we give the implementation and evaluation.

2 Preliminaries

Let λ be the security parameter throughout the paper.

2.1 Pairing and Hard Assumptions

Pairing. Let $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ be three multiplicative groups of prime order q. g_1, g_2 are generators of group $\mathbb{G}_1, \mathbb{G}_2$, resp.. e is an efficiently computable bilinear map with the following properties:

- Bilinear: for any $a, b \in \mathbb{Z}_q$, we have that $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.
- Non-degenerate: $e(g_1, g_2) \neq 1$.

Next, we provide two variants of Diffie-Hellman problems. From the work of [BDZ03], we know that Diffie-Hellman problems are equivalent to divisible Diffie-Hellman problems. We define co-divisible Diffie-Hellman problems from co-Diffie-Hellman problems [BGLS03] as follows.

Definition 1 (co-Divisible Computation Diffie-Hellman (co-DCDH))**.** For $i \in \{1, 2, T\}$, let $\mathbb{G}_i$ be a cyclic group of prime order q with generator g_i. The advantage of an algorithm $\mathcal{A}$ solving the *co-Divisible Computation Diffie-Hellman (co-DCDH)* problem for $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ is

$$\mathbf{Adv}^{\mathsf{co\text{-}DCDH}}_{(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T),\mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}^{\mathsf{co\text{-}DCDH}}_{(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T),\mathcal{A}} = 1],$$

where the experiment $\mathbf{Exp}^{\mathsf{co\text{-}DCDH}}_{(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T),\mathcal{A}}$ is given in Fig. 1 (left).

$\underline{\mathbf{Exp}^{\mathsf{co\text{-}DCDH}}_{(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T),\mathcal{A}}:}$

$x, y \xleftarrow{\$} \mathbb{Z}_q$
$Z \leftarrow \mathcal{A}(g_1^x, g_2^y)$
if $Z = g_1^{x/y}$
  return 1
else
  return 0

$\underline{\mathbf{Exp}^{\mathsf{co\text{-}DDBDH\text{-}b}}_{(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T),\mathcal{A}}:}$

$x, y, z, w \xleftarrow{\$} \mathbb{Z}_q$
if $\mathrm{b} = 1$
  $c \leftarrow e(g_1, g_2)^{\frac{yz}{x}}$
else
  $c \leftarrow e(g_1, g_2)^w$
$\mathrm{b}' \leftarrow \mathcal{A}(g_2^x, g_1^y, g_2^z, c)$
return b'

Figure 1: co-DCDH experiment (left), co-DDBDH experiment (right).

Definition 2 (co-Divisible Decision Bilinear Diffie-Hellman (co-DDBDH))**.** For $i \in \{1, 2, T\}$, let $\mathbb{G}_i$ be a cyclic group of prime order q with generator g_i. The advantage of an algorithm $\mathcal{A}$ solving the *1-Quotient Decision Bilinear Diffie-Hellman (co-DDBDH)* problem for $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ is

$$\mathbf{Adv}^{\mathsf{co\text{-}DDBDH}}_{(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T),\mathcal{A}}(\lambda) = \left|\Pr[\mathbf{Exp}^{\mathsf{co\text{-}DDBDH\text{-}1}}_{(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T),\mathcal{A}} = 1] - \Pr[\mathbf{Exp}^{\mathsf{co\text{-}DDBDH\text{-}0}}_{(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T),\mathcal{A}} = 1]\right|,$$

where the experiment $\mathbf{Exp}^{\mathsf{co\text{-}DDBDH}}_{(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T),\mathcal{A}}$ is given in Fig. 1 (right).

2.2 Signature Schemes

We define signature scheme and its unforgeability notion in this section.

Definition 3 (Signature Scheme)**.** A *signature scheme* Σ consists of three algorithms: $\mathsf{KG}, \mathsf{Sign}, \mathsf{Ver}$.

- $\mathsf{KG}(\lambda) \to (\mathsf{S}, \mathsf{V})$. On input a security parameter λ, the key generation algorithm outputs a secret signing key S and a public verification key V.
- $\mathsf{Sign}(\mathsf{S}, m) \to \sigma$: On input a secret signing key S and a message $m \in \mathcal{M}$, the signing algorithm outputs a signature σ.
- $\mathsf{Ver}(\mathsf{V}, m, \sigma) \to \mathrm{b}$: On input a public verification key V, a message $m \in \mathcal{M}$ and a signature σ, the verifying algorithm outputs a bit $\mathrm{b} \in \{0, 1\}$.

The correctness of a signature is defined as follows.

Definition 4 (Correctness of a Signature Scheme)**.** Let $\Sigma = (\mathsf{KG}, \mathsf{Sign}, \mathsf{Ver})$ be a signature scheme, we say Σ is *correct*, if for any key pair $(\mathsf{S}, \mathsf{V}) \leftarrow \mathsf{KG}(\lambda)$, for any message $m \in \mathcal{M}$, we have that $\mathbf{Pr}[\mathsf{Ver}(\mathsf{V}, m, \mathsf{Sign}(\mathsf{S}, m)) = 1] = 1$.

The existential unforgeability under adaptively chosen message attacks ($\mathsf{EUF\text{-}CMA}$ security) of a signature is defined as follows.

Definition 5 ($\mathsf{EUF\text{-}CMA}$ security)**.** Let $\Sigma = (\mathsf{KG}, \mathsf{Sign}, \mathsf{Ver})$ be a signature scheme, the $\mathsf{EUF\text{-}CMA}$ advantage of an adversary $\mathcal{A}$ against Σ is defined as

$$\mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\Sigma,\mathcal{A}} = \mathbf{Pr}[\mathbf{Exp}^{\mathsf{EUF\text{-}CMA}}_{\Sigma,\mathcal{A}} = 1],$$

where the experiment $\mathbf{Exp}^{\mathsf{EUF\text{-}CMA}}_{\Sigma,\mathcal{A}}$ is given in Fig. 2.

2.3 Proxy Re-Encryption

We define proxy re-encryption (PRE) scheme in this section.

Definition 6 (Proxy Re-Encryption Scheme)**.** A *proxy re-encryption scheme* PRE are parameterized by a tuple of algorithms $\{\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReKG}, \mathsf{ReEnc}\}$.

$\mathbf{Exp}^{\mathsf{EUF\text{-}CMA}}_{\Sigma,\mathcal{A}}$:
$(\mathsf{S},\mathsf{V}) \leftarrow \mathsf{KG}(\lambda)$
$\mathcal{Q} \leftarrow \emptyset$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}.\mathsf{Sign}}(\mathsf{V})$
if $m^* \notin \mathcal{Q}$ **and** $\mathsf{Ver}(\mathsf{V}, m^*, \sigma^*) = 1$
return 1
else
return 0

$\mathcal{O}.\mathsf{Sign}(m)$:
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m\}$
$\sigma \leftarrow \mathsf{Sign}(\mathsf{S}, m)$
return σ

Figure 2: Experiment $\mathbf{Exp}^{\mathsf{EUF\text{-}CMA}}_{\Sigma,\mathcal{A}}$ for signature scheme Σ and adversary $\mathcal{A}$.

- $\mathsf{KG}(\lambda) \rightarrow (\mathsf{sk}, \mathsf{pk})$. On input a security parameter λ, the key generation algorithm outputs a secret key sk and a public key pk.
- $\mathsf{Enc}(\mathsf{pk}, \mathsf{m}) \rightarrow \mathsf{c}$: On input a public key pk and a message $\mathsf{m} \in \mathcal{M}$, the encryption algorithm outputs a ciphertext c.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{c}) \rightarrow \mathsf{m}$: On input a secret key sk and a ciphertext c, the decryption algorithm outputs a message m.
- $\mathsf{ReKG}(\mathsf{sk}_i, \mathsf{pk}_j, \mathsf{c}_i^*) \rightarrow \mathsf{rk}_{i,j}$: On input a secret key sk_i, a public key pk_j and a ciphertext c_i, the re-key generation algorithm outputs a re-key $\mathsf{rk}_{i,j}$. $*$ means the ciphertext is omitted if the re-key generation does not need the ciphertext as an input.
- $\mathsf{ReEnc}(\mathsf{rk}_{i,j}, \mathsf{c}_i) \rightarrow \mathsf{c}_j$: On input a re-key $\mathsf{rk}_{i,j}$ and a ciphertext c_i, the re-encryption algorithm outputs a ciphertext c_j.

2.4 Zero-Knowledge Proof

A zero-knowledge proof is a protocol in which a prover wants to convince a verifier that a statement is true without revealing any private information. A zero-knowledge proof protocol consists of three PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$. These are the common reference generator $\mathcal{G}$, the interactive prover $\mathcal{P}$ and verifier $\mathcal{V}$. Take input as 1^λ, $\mathcal{G}$ outputs the common reference σ. The communication transcript between $\mathcal{P}$ and $\mathcal{V}$ when interacting on inputs s and t is denoted by $tr \leftarrow \langle \mathcal{P}(s), \mathcal{V}(t) \rangle$. We write the output of the protocol as $\langle \mathcal{P}(s), \mathcal{V}(t) \rangle = b$. If verifier accepts, $b = 0$, otherwise $b = 1$. The language of zero-knowledge proof is defined over a polynomial time decidable relation $R \subset \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^*$, given σ, w is a witness for statement u if $(\sigma, w, u) \in R$. Define the language

$$\mathcal{L}_\sigma = \{x \mid \exists w \text{ s.t. } (\sigma, x, w) \in R\}$$

as the set of statements x that have a witness w in the relation R.

Definition 7 (Perfect Completeness)**.** The triple $(\mathcal{G}, \mathcal{V}, \mathcal{P})$ has perfect completeness if for all non-uniform PPT adversary $\mathcal{A}$ such that

$$\Pr\left[\begin{array}{l|l} (\sigma, u, w) \notin R & \sigma \leftarrow \mathcal{G}(1^\lambda) \\ \vee \langle \mathcal{P}(\sigma, u, w), \mathcal{V}(\sigma, u) \rangle = 1 & (u, w) \leftarrow \mathcal{A}(\sigma) \end{array}\right] = 1$$

Definition 8 (Computational Soundness)**.** $(\mathcal{G}, \mathcal{V}, \mathcal{P})$ has computational soundness if it is not possible to prove a false statement where no witness exist, i.e. for all non-uniform polynomial time interactive adversary $\mathcal{A}_1, \mathcal{A}_2$, the function $\mathsf{negl}[\lambda]$ is negligible.

$$\Pr\left[\begin{array}{l|l} \mathcal{A}_1(tr) = 1 \text{ (\textit{i.e. tr is accepting})} \wedge & \sigma \leftarrow \mathcal{G}(1^\lambda) \\ (\sigma, u, w) \notin R & (u, w) \leftarrow \mathcal{A}_2(\sigma) \end{array}\right] \leq \mathsf{negl}[\lambda]$$

Definition 9 (Computational Knowledge Soundness)**.** $(\mathcal{G}, \mathcal{V}, \mathcal{P})$ has computational knowledge soundness if for all deterministic polynomial time $\mathcal{P}^*$, there exists an polynomial time knowledge extractor $\mathcal{E}$ such that for all non-uniform polynomial time interactive adversary $\mathcal{A}_1, \mathcal{A}_2$, the function $\mathsf{negl}[\lambda]$ is negligible.

$$\left| \Pr\left[\mathcal{A}_1(tr) = 1 \;\middle|\; \begin{array}{l} \sigma \leftarrow \mathcal{G}(1^\lambda), (u,s) \leftarrow \mathcal{A}_2(\sigma) \\ tr \leftarrow \langle \mathcal{P}^*(\sigma,u,s), \mathcal{V}(\sigma,u) \end{array} \right] - \Pr\left[\begin{array}{l} \mathcal{A}_1(tr) = 1 \wedge \\ (tr \text{ is accepting i.e. } (\sigma,u,w) \in R) \end{array} \;\middle|\; \begin{array}{l} \sigma \leftarrow \mathcal{G}(1^\lambda) \\ (u,s) \leftarrow \mathcal{A}_2(\sigma) \\ (tr,w) \leftarrow \mathcal{E}^{\mathcal{O}}(\sigma,u) \end{array} \right] \right| \leq \mathsf{negl}[\lambda]$$

where the oracle is given by $\mathcal{O} = \langle \mathcal{P}^*(\sigma,u,s), \mathcal{V}(\sigma,u) \rangle$.

The oracle $\mathcal{O}$ permits rewinding to a specific point and resuming with fresh randomness for the verifier from this point onwards. Informally, if there is an adversary that can produce an proof that satisfies the verifier with some probability, then there exists an emulator that can extract the witness. The value s is the internal state of $\mathcal{P}^*$, including randomness. The emulator is permitted to rewind the interaction between the prover and verifier to any move, then resuming with fresh randomness for the verifier.

Definition 10 (Perfect Special Honest-Verifier Zero-Knowledge)**.** A triple $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a perfect special honest verifier zero knowledge argument of knowledge for R if there exists a probabilistic polynomial time simulator $\mathcal{S}$ such that for all pairs of interactive adversaries $\mathcal{A}_1, \mathcal{A}_2$

$$\Pr\left[(\sigma,u,w) \in R \;\wedge\; \mathcal{A}_1(tr) = 1 \;\middle|\; \begin{array}{l} \sigma \leftarrow \mathcal{G}(1^\lambda), (u,w,\rho) \leftarrow \mathcal{A}_2(\sigma) \\ tr \leftarrow \langle \mathcal{P}^*(\sigma,u,w), \mathcal{V}(\sigma,u;\rho) \rangle \end{array} \right]$$

$$= \Pr\left[(\sigma,u,w) \in R \;\wedge\; \mathcal{A}_1(tr) = 1 \;\middle|\; \begin{array}{l} \sigma \leftarrow \mathcal{G}(1^\lambda), (u,w,\rho) \leftarrow \mathcal{A}_2(\sigma) \\ tr \leftarrow \mathcal{S}(u,\rho) \end{array} \right]$$

where ρ is the randomness used by the verifier.

Definition 11 (Zero-knowledge Proof of Knowledge)**.** The triple $(\mathcal{S}, \mathcal{P}, \mathcal{V})$ is a commit-and-prove zero-knowledge argument of knowledge for a family of relations R if it satisfies the perfect completeness, perfect special honest-verifier zero-knowledge and computational soundness or computational knowledge soundness.

3 Fair Data Trading Protocol (Overview)

We introduce a generic protocol for secure fair data trading. A conceptual overview of our construction is given in this section and the formal definition is provided in Section 5.

Entities. We define four roles in fair data trading protocol.

- **Data manager**: The main role of a data manager is to verify the truth of the trading data and its encryption. It prohibits sellers to fake data for profits. For example, a data manager can be a hospital or an IoT device provider. The data manager is reliable for keeping sellers' privacy.
- **Data subject (provider):** the party from which the data was originated. It can be a data provider who wants to sell data for profits.
- **Data collector:** the entity that wants to buy data. Data collector initiates the data trading process, registers the Smart Contract and deposits rewards.
- **Smart contract/Blockchain:** the entity that provides a decentralised and trust-less platform to trade data with cryptocurrencies. The smart contracts process the players' registrations, re-key verifications and send rewards to data providers.

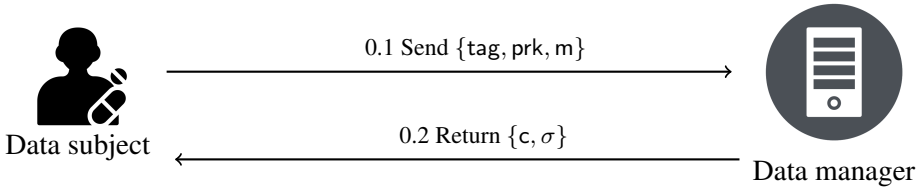

Figure 3: Pre-trading phase. Data manager encrypts the message and signs the ciphertext.

Initialization. Before the protocol starts running, each protocol participant runs a key generation algorithm to generate a secret signing key and a public verification key pair $(\mathsf{S}, \mathsf{V}) \leftarrow \Sigma.\mathsf{KG}(\lambda)$. Additionally, data collectors generate their PRE key pairs as $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{PRE.KG}(\lambda)$.

Pre-Trading Phase. In our architecture, the data manager is not directly involved in the actual trading communication, while it keeps a critical role before the data trading happens on the smart contract, i.e. authorising the truth of the trading data along with its encryption. This is the preparation that needs to be done in the pre-trading phase. Our protocol uses symmetric encryption scheme to encrypt data since the trading data might be large. As a result, the data trading problem is turning into a symmetric key trading problem, which we use proxy re-encryption scheme to address.

A conceptual overview of the pre-trading phase is given in Fig. 3. Formally, when a data manager receives [1] a pre-trading request $(\mathsf{pre\text{-}trading} : \mathsf{tag}, \mathsf{prk}, \mathsf{m})$ from a data subject, where prk is the data subject's one time public key [2], the tag specify what m is about (for example, diabetics, blood test, antibody test, etc.. in medical sharing case), the data manager does the following.

1. Fetch the data subject's relevant data m according to tag, validate the truth of the data.
2. Generate a one time symmetric encryption key: $\mathsf{k} \xleftarrow{\$} \mathcal{K}$, use this key to encrypt data: $\mathsf{c_m} \leftarrow \mathsf{SKE.Enc}(\mathsf{k}, \mathsf{m})$.
3. Take the symmetric key as a plaintext and encrypt it to get a ciphertext of the key k: $\mathsf{c_k} \leftarrow \mathsf{PRE.Enc}(\mathsf{prk}, \mathsf{k})$.
4. Compute the hash of prk, $\mathsf{c_k}$ and $\mathsf{c_m}$ respectively. Then compute a signature over these hashes and tag: $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{S}, \langle \mathsf{H}(\mathsf{prk}), \mathsf{H}(\mathsf{c_k}), \mathsf{H}(\mathsf{c_m}), \mathsf{tag} \rangle)$ where S is the signing key of the data manager.
5. Denote $\mathsf{c} \leftarrow (\mathsf{prk}, \mathsf{c_k}, \mathsf{c_m})$ and send (c, σ) to the data subject.

Trading Phase. The trading phase has on-chain and off-chain processes. We use proxy re-encryption scheme to transfer the ciphertext that is encrypted under data subject's public key to the data collector's public key by the help of a re-key. The re-key is the key material on-chain and it plays the central role on fair data trading without losing privacy. To prove the re-key is a valid key material, a re-key verification process is involved and zero knowledge proof technique is deployed to solve this problem. A conceptual overview of the trading phase is given in Fig. 4. Formally, the data subject DS, data collector DC and smart contract SC follow the process:

1. DC publishes a smart contract on blockchain. The smart contract at the beginning includes: data collector's public key $\mathsf{pk_{DC}}$, common reference of the zero-knowledge proof: $\mathsf{crs} \leftarrow \mathcal{G}(1^\lambda)$[3], reward amount R, smart contract balance $\mathsf{deposit}$ and data requirement tags $\{\mathsf{tag}\}_{\mathsf{tag} \in T}$.
2. DS gets the required data tag set T from the smart contract, then it registers its address that is to receive the reward on the smart contract. DS also initiates an inner state $\mathsf{st} : \perp$ that records the chosen randomnesses of DS during the trading phase.

[1] All communications between different participants are running inside a confidential secure channel, hence, data in transmission is resistant to overhearing.

[2] The one time public key prk makes sure no one can identify the data subject from a random data subject, which provides the data subject anonymity.

[3] The zero-knowledge proof does not need trust-setup, the common reference generator can be run on the smart contract.

3. DS runs re-key generation algorithm to generate re-keys rk for each tag tag: parse $\mathsf{c} = (\mathsf{prk}, \mathsf{c_k}, \mathsf{c_m})$, $\mathsf{rk} \leftarrow \mathsf{PRE.ReKG}(\mathsf{srk}, \mathsf{pk_{DC}}, \mathsf{c_k})$, computes auxiliary information: $\mathsf{aux} \leftarrow \mathsf{Statement}(\mathsf{pk_{DC}}, \mathsf{srk}, \mathsf{prk}, \mathsf{st}, \mathsf{c_k})$ and sends tag related ciphertext $\{(\mathsf{tag}, \mathsf{c}, \sigma, \mathsf{aux})\}_{\mathsf{tag}\in T}$ to DC off-chain. The auxiliary information: aux is a set of parameters that is used later by DC to generate blind statements (see Section 6).

4. DC checks the identity of data manager, verifies the ciphertext and signature for each tag by computing the signature verification algorithm $\Sigma.\mathsf{Ver}(\mathsf{V}, (\mathsf{c}, \sigma))$, If they are all valid, DC generates a blind statement: $\mathsf{bs} \leftarrow \mathsf{Blind}(\mathsf{srk}, \mathsf{c}, \mathsf{aux})$ for every tag. Then DC confirms to start the data exchanging with DS on the smart contract and sends blinded statements $\{\mathsf{bs}\}_{\mathsf{tag}\in T}$ to the smart contract. Otherwise, if any verification fails or Blind returns $\perp$, DC rejects. Note that the blind statements plays a central role to verify the correctness of the rekey, it also randomises the verification such that no information about the ciphertext is revealed, see further discussion in Remark. 1.

5. After seeing the confirmation and blind statements on the smart contract, DS generates a proof π for its re-key: $\pi \leftarrow \mathsf{RkProve}(\mathsf{rk}, \mathsf{srk}, \mathsf{st})$, and then computes the re-key verification algorithm $\mathsf{RkVerify}(\mathsf{rk}, \mathsf{bs}, \pi)$ (see Remark. 1) to make sure the smart cotract will accept the re-key and proof. Next, DS sends re-keys and proofs $\{\mathsf{rk}, \pi\}_{\mathsf{tag}\in T}$ to the smart contract.

6. Smart contract verifies if re-keys are valid by $\mathsf{RkVerify}(\mathsf{rk}, \mathsf{bs}, \pi)$. Send rewards to DS and record re-keys $\{\mathsf{rk}\}_{\mathsf{tag}\in T}$ if and only if the verifications are successful for all tags in T.

7. DC can re-encrypt the ciphertext $\mathsf{c_k}$ to a ciphertext under DC's private key, by using the re-key rk: parse $\mathsf{c} = (\mathsf{prk}, \mathsf{c_k}, \mathsf{c_m})$, $\mathsf{c}' \leftarrow \mathsf{PRE.ReEnc}(\mathsf{rk}, \mathsf{c_k})$. Key $\mathsf{sk_{DC}}$ can be used to decrypt ciphertext c': $\mathsf{k} \leftarrow \mathsf{PRE.Dec}(\mathsf{sk_{DC}}, \mathsf{c}')$, then this key k can be used to get the data that DC wants from DS: $\mathsf{m} \leftarrow \mathsf{SKE.Dec}(\mathsf{k}, \mathsf{c_m})$.

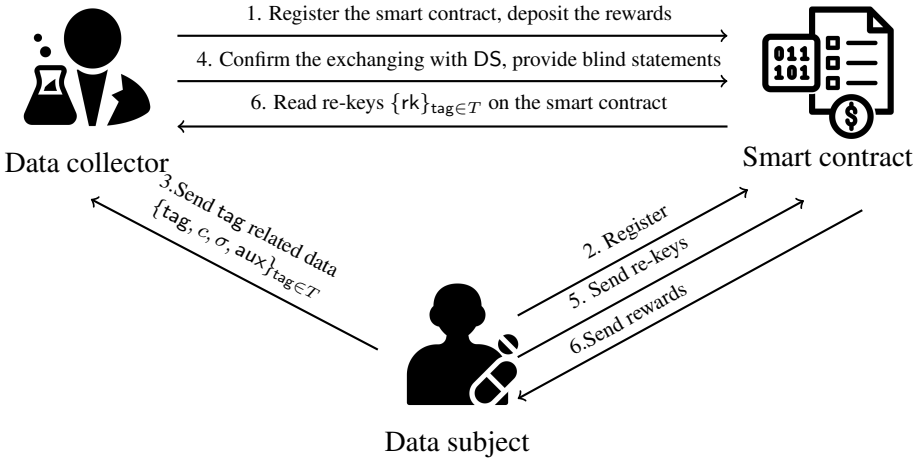

Figure 4: Trading phase. After step 3, the data collector verifies the validity of the data $\{\mathsf{tag}, c, \sigma, \mathsf{aux}\}_{\mathsf{tag}\in T}$ by checking data manager's signature. Only if the signatures are valid, it confirms the data exchanging and sends blind statements to the smart contract in step 4. In step 5, the data subject pre-check if the smart contract will accept its re-key, sends the re-key to the smart contract if and only if all the verification can pass. In step 6, the smart contract run the re-key verification and sends reward to the data subject if the verification is successful. Eventually, in step 6, the data collector rotates the ciphertext it received in step 3 to ciphertext under its own key and read the original message.

4 Security Notions

In this section, we provide a communication model and security notions for fair data trading protocol. We define the security notions use game-based method and follow the syntax from prior works [BCP02, BM08]. Furthermore, we define fairness and show that our security notions implies it.

4.1 Communication Model

In this section, we give a communication model for our fair data trading protocol.

Protocol Participants, Long-Term Keys and Instance Oracles. A participant V in the protocol can be a data manager DM, a data subject DS, a data collector DC, or a smart contract SC. Each participant holds a long-term signing key S, each data collector DC additionally holds a long-term secret key sk, the corresponding verification key V and public key pk of all V are known to all. Protocol participants have several instances called *oracles*, denoted $\prod_{\mathsf{V}}^{\alpha}$ for $\alpha \in \mathbb{N}$. Each oracle $\prod_{\mathsf{V}}^{\alpha}$ is associated with the variables $\mathsf{state}_{\mathsf{V}}^{\alpha}$, $\mathsf{role}_{\mathsf{V}}^{\alpha}$, $\mathsf{pid}_{\mathsf{V}}^{\alpha}$, $\mathsf{sid}_{\mathsf{V}}^{\alpha}$, $\mathsf{m}_{\mathsf{V}}^{\alpha}$, $\mathsf{rk}_{\mathsf{V}}^{\alpha}$, $\mathsf{bs}_{\mathsf{V}}^{\alpha}$ and $\pi_{\mathsf{V}}^{\alpha}$ as follows:

- $\mathsf{state}_{\mathsf{V}}^{\alpha}$ takes a value from $\{\mathsf{unused}, \mathsf{ready}, \mathsf{processing}, \mathsf{accepted}, \mathsf{rejected}\}$.
- $\mathsf{role}_{\mathsf{V}}^{\alpha}$ takes a value from: $\mathsf{DM}, \mathsf{SC}, \mathsf{DS}, \mathsf{DC}$.
- $\mathsf{pid}_{\mathsf{V}}^{\alpha}$ contains a set of participants, which are partners.
- $\mathsf{sid}_{\mathsf{V}}^{\alpha}$ contains a string defined by the protocol.
- $\mathsf{m}_{\mathsf{V}}^{\alpha}$ the agreed message.
- $\mathsf{rk}_{\mathsf{V}}^{\alpha}$ the session re-key.
- $\mathsf{bs}_{\mathsf{V}}^{\alpha}$ the (blinded) statement.
- $\pi_{\mathsf{V}}^{\alpha}$ the proof.

Each session is identified by a unique, publicly-known session identifier, denoted sid. The partner identifier, denoted pid, contains the identities of all participants in a session. There are two types of sessions, one for pre-trading phase and another for trading phase. Sessions in the pre-trading phase are running between data manager and data subject. Sessions in the trading phase are running among data subject, data collector and smart contract.

Each oracle $\prod_{\mathsf{V}}^{\alpha}$ stays unused until it is initialised be a data manager, smart contract, a data subject or a data collector along with the corresponding long-term key. It then begins with $\mathsf{state}_{\mathsf{V}}^{\alpha} = \mathsf{ready}$ and $\mathsf{role}_{\mathsf{V}}^{\alpha}$, $\mathsf{pid}_{\mathsf{V}}^{\alpha}$, $\mathsf{sid}_{\mathsf{V}}^{\alpha}$, $\mathsf{m}_{\mathsf{V}}^{\alpha}$, $\mathsf{rk}_{\mathsf{V}}^{\alpha}$, $\mathsf{bs}_{\mathsf{V}}^{\alpha}$ and $\pi_{\mathsf{V}}^{\alpha}$ are all set to $\perp$. After the protocol starts, each oracle $\prod_{\mathsf{V}}^{\alpha}$ learns its partner's identifier $\mathsf{pid}_{\mathsf{V}}^{\alpha}$ (and possibly $\mathsf{sid}_{\mathsf{V}}^{\alpha}$) and turns into a $\mathsf{processing}$ state, where it sends, receives and processes messages. If the protocol at oracle $\prod_{\mathsf{V}}^{\alpha}$ *fails*, for example if a certain verification fails, then the oracle terminates and sets its state to $\mathsf{rejected}$. Otherwise, after computing $\mathsf{m}_{\mathsf{V}}^{\alpha}$ (or sending out $\mathsf{rk}_{\mathsf{V}}^{\alpha}$ or sending out reward) oracle $\prod_{\mathsf{V}}^{\alpha}$ changes its state to $\mathsf{accepted}$ and no longer responds to protocol messages. More precisely, oracles in the pre-trading phase accepts after computing session message. In the trading phase, data collector oracles accepts after computing session message, data subject oracles accept after sending out re-keys and smart contract oracles accept after sending out reward.

Correctness. Correctness is defined to make sure that an honest run of the protocol, all partners will agree with a traded message. We say a FDT protocol has *correctness* if for all $\mathsf{DC}, \mathsf{DS}, \mathsf{SC}, \alpha, \beta, \gamma$, such that $\mathsf{pid}_{\mathsf{DC}}^{\alpha} = \mathsf{pid}_{\mathsf{DS}}^{\beta} = \mathsf{pid}_{\mathsf{SC}}^{\gamma}$, $\mathsf{sid}_{\mathsf{DC}}^{\alpha} = \mathsf{sid}_{\mathsf{DS}}^{\beta} = \mathsf{sid}_{\mathsf{SC}}^{\gamma}$, $\mathsf{state}_{\mathsf{DC}}^{\alpha} = \mathsf{state}_{\mathsf{DS}}^{\beta} = \mathsf{state}_{\mathsf{SC}}^{\gamma} = \mathsf{accepted}$ and $\mathsf{m}_{\mathsf{DC}}^{\alpha} = \mathsf{m}_{\mathsf{DS}}^{\beta} \neq \emptyset$.

Soundness. Soundness ensures that if the verification of the key material is accepted, the data collector can get the desired message with the help of this key material. We define a FDT protocol has *soundness* if for all $\mathsf{DC}, \mathsf{DS}, \alpha, \beta$, and correctly [4] generated statement $\mathsf{bs}_{\mathsf{DC}}^{\beta}$, if $\mathsf{RkVerify}(\mathsf{rk}_{\mathsf{DS}}^{\alpha}, \mathsf{bs}_{\mathsf{DC}}^{\beta}, \pi_{\mathsf{DS}}^{\alpha}) = 1$ then $\mathsf{m}_{\mathsf{DC}}^{\alpha} = \mathsf{m}_{\mathsf{DS}}^{\beta} \neq \emptyset$.

[4] We specify that an honest data collector does not have incentive to compute an incorrect statement to make the re-key verification fails, because the verification failure will result to the data collector receiving nothing. In addition, the honest data collector wishes to get valid re-keys, which can be checked by correct statements.

4.2 Security Notions

Adversarial Model. Suppose the adversary $\mathcal{A}$ have complete control on communications in the network, determining which instances run and obtaining access to some useful information. Informally, the adversary can adaptively corrupt keys (except for corrupting partners in the challenge query), obtain public values on smart contract (including re-keys) and transcripts. Formally, It can interact with protocol participants by asking for queries to their oracles as follows.

- $\mathsf{Execute}(\mathcal{S})$: Executes the protocol among a set of unused oracles $\mathcal{S}$ and outputs the transcript of this execution. If the data subject sends the same data to the same data collector, it will be noticed by the data collector and the data collector will reject paying for the same data, we restricts such execution on honest data subjects. This restriction is required in the Pub-RoRChall query as well.
- $\mathsf{Send}(\prod_{\mathsf{V}}^{\alpha}, \mathsf{trans})$: Sends transcript trans to oracle $\prod_{\mathsf{V}}^{\alpha}$ and outputs the reply (if any) by the oracle.
- $\mathsf{Corrupt}(\mathsf{V})$: Outputs the long-term privite key of participant V.
- M-RoRChall$(\prod_{\mathsf{V}}^{\alpha})$: This query can only be asked once by $\mathcal{A}$ in the trading phase. If oracle $\prod_{\mathsf{V}}^{\alpha}$ has status accepted, holding a session message $\mathsf{m}_{\mathsf{V}}^{\alpha}$, then this query outputs the session message $\mathsf{m}_{\mathsf{V}}^{\alpha}$ if $\mathrm{b} = 1$, or a random string from the session message space if $\mathrm{b} = 0$.
- Pub-RoRChall$(\mathcal{S})$: Input a set of unused oracles $\mathcal{S}$. All oracles in $\mathcal{S}$ execute an honest run of the protocol with respect to a real ciphertext if $\mathrm{b} = 1$. All oracles in $\mathcal{S}$ execute an honest run of the protocol with respect to a random ciphertext if $\mathrm{b} = 0$. Record public values on the smart contract.
- SigUFChall$(\prod_{\mathsf{DC}}^{\alpha}, \mathsf{trans})$: This query can only be asked once by $\mathcal{A}$ and executed as follows, sends transcript trans to oracle $\prod_{\mathsf{DC}}^{\alpha}$. If there is no reject happens, the data manager DM who signed the corresponding signature is not corrupted and this signature has never been generated before, outputs 1. Otherwise, outputs 0.
- RkUFChall$(\prod_{\mathsf{SC}}^{\alpha}, \mathsf{rk})$: This query can only be asked once by $\mathcal{A}$ and executed as follows, sends re-key rk to smart contract $\prod_{\mathsf{SC}}^{\alpha}$. If there is no reject happens and no data subject in $\mathsf{pid}_{\mathsf{SC}}^{\alpha}$ is corrupted, outputs 1. Otherwise, outputs 0.

Initialization. Before any game starts, each participant V runs the key generation algorithm to generate long-term key pairs. The private keys are only known to the principal, while public keys are revealed to every participant and the adversary.

Real or Random Indistinguishability. There are two types of *real or random indistinguishability*: message real or random indistinguishability (M-RoR) and public real or random indistinguishability (Pub-RoR). M-RoR guarantees that fresh session messages are random looking, therefore the adversary cannot see any information about the traded message unless the adversary has the knowledge of both the corresponding re-key and a data collector's secret key at the same time. It indicates that the public cannot infer the corresponding message from the re-key, it also ensures the data collector cannot learn the expected message until it pays for the re-key. Pub-RoR makes sure the real public values on the smart contract are indistinguiahable from public values generated from a random ciphertext, which breaks the link among different sessions, therefore, Pub-RoR implies public unlinkability.

Freshness. Freshness is defined to prevent trivial wins. In the M-RoR game, if any data subject or data manager partner is corrupted then the underlying message is learnt by the adversary. If any data collector partner is corrupted and the re-key is known to the adversary then the underlying message can be computed by the adversary. In the M-RoR game, an oracle $\prod_{\mathsf{V}}^{\alpha}$ is *fresh* if no $\mathsf{DS}, \mathsf{DM} \in \mathsf{pid}_{\mathsf{V}}^{\alpha}$ is ever corrupted and the following condition does not happens at the same time: $\mathsf{DC} \in \mathsf{pid}_{\mathsf{V}}^{\alpha}$ is ever corrupted and re-key $\mathsf{rk}_{\mathsf{V}}^{\alpha}$ is revealed. In the Pub-RoR game, if any partner is corrupted then the original ciphertext (which is used in the re-key generation) is learnt by the adversary. Hence, the adversary can potentially identify the real re-key

from a randomly generated re-key. In the Pub-RoR game, an oracle $\prod_{\mathsf{V}}^{\alpha}$ is *fresh* if no player in $\mathsf{pid}_{\mathsf{V}}^{\alpha}$[5] is ever corrupted. Formally, we define M-RoR and Pub-RoR notions as follows.

Definition 12. Let FDT be a correct FDT protocol. The xx advantage of any adversary $\mathcal{A}$ against FDT is

$$\mathbf{Adv}^{\mathsf{xx}}_{\mathsf{FDT},\,\mathcal{A}}(\lambda) = \left|\Pr[\mathbf{Exp}^{\mathsf{xx\text{-}1}}_{\mathsf{FDT},\,\mathcal{A}} = 1] - \Pr[\mathbf{Exp}^{\mathsf{xx\text{-}0}}_{\mathsf{FDT},\,\mathcal{A}} = 1]\right|,$$

where $\mathsf{xx} \in \{\mathsf{M\text{-}ROR}, \mathsf{Pub\text{-}RoR}\}$ and the experiment $\mathbf{Exp}^{\mathsf{xx\text{-}b}}_{\mathsf{FDT},\,\mathcal{A}}$ is given as follows:

- *Queries.* After initialization, the adversary $\mathcal{A}$ is allowed to ask for Execute, Send, Corrupt and xxChall queries. The xxChall query should keep fresh until the end of this phase.
- *Guessing.* $\mathcal{A}$ outputs its guess b', which is the output of this experiment.

Unforgeability. There are two types of forgeries: signature forgery and re-key forgery. A valid signature forgery is a valid signature which is not generated before by any data manager. The *signature unforgeability* notion is to measure the possibility of a data subject providing a valid signature forgery itself. *Re-key unforgeability* prevents data collector have successful data trading by using traded data. More precisely, if a data collector can create a valid re-key forgery for traded data, then this data collector can re-sell data by trading the received data and self-created re-key to other data collector. Formally, we define SigUF and RkUF notions as follows.

Definition 13. Let FDT be a correct FDT protocol. The xx advantage of any adversary $\mathcal{A}$ against FDT is

$$\mathbf{Adv}^{\mathsf{xx}}_{\mathsf{FDT},\,\mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}^{\mathsf{xx}}_{\mathsf{FDT},\,\mathcal{A}} = 1],$$

where $\mathsf{xx} \in \{\mathsf{SigUF}, \mathsf{RkUF}\}$ the experiment $\mathbf{Exp}^{\mathsf{xx}}_{\mathsf{FDT},\,\mathcal{A}}$ is given as follows:

- *Queries.* After initialization, the adversary $\mathcal{A}$ is allowed to ask for Execute, Send and Corrupt queries.
- *Forgery.* $\mathcal{A}$ provides a forgery to a xxChall query, the output of xxChall query is the output of this experiment.

4.3 Fairness

We follow the syntax from prior work [DEF18b] and define fairness in general. Moreover, we show that our security notions indicates fairness for our FDT protocol.

Definition 14. We say a data trading protocol has *data subject fairness* if: an honest data subject DS is guaranteed that the data collector DC only learns the desired data iff it pays the reward. We say a data trading protocol has *data collector fairness* if: an honest data collector DC is ensured that it only pays the reward iff it gets the desired data.

Theorem 1. If a FDT protocol has M-ROR security, then it has data subject fairness. If a FDT protocol has soundness and signature unforgeability, then it has data collector fairness.

Proof. Due to M-ROR security, the data collector cannot learn the desired data if he does not have the corresponding re-key. By the design of our protocol in step 5 and 6 (see page 9), an honest data subject DS sends out the re-key iff the re-key verification is correct, which implies the payment of reward. Hence, FDT protocol has data subject fairness.

Signature unforgeability prohibits forgery, that is, if the signature verification is successful, the data is truly generated from a trusted data manager. Re-key verification make sure the data collector can obtain a valid re-key when the reward is paid. Due to soundness, the valid re-key makes sure the data collector can get the desired data. Hence, FDT protocol has data collector fairness. □

[5] We extend this partner id when the data subject later trades the same data to different data collectors, this extended partner id is only used for checking if the challenge oracle is fresh.

4.4 Desired Functionalities

We summarise the following functionalities that a FDT protocol desires. Desired functionalities for data subjects are:

1. Message real or random indistinguishability M-ROR, which ensures traded message will not be revealed without re-key. It implies fairness to data subjects, see Theorem 1.
2. Public real or random indistinguishability Pub-RoR, public values in different trading periods are independent and random from each other, which implies public unlinkability. In other words, no one (except the trading partners) can identify the relations among different trading values, it is not noticeable even the data subject is trading the same data multiple times.
3. Re-key unforgeability RkUF, which implies data cannot be re-sold by the data collector, only the data subject can be a data provider.
4. Data subject anonymity, see footnote on page 8.

Desired functionalities for data collectors are:

1. Signature unforgeability SigUF. Signature unforgeability makes sure that no one can deploy this system to forge new valid data to gain profit.
2. Soundness. Together with signature unforgeability, it implies fairness to data collector. As a result, data collectors will get the desired data if they paid the reward.
3. Original data is unchanged, the benefit to data collector is that data collectors can notice if they buy the same data again.

5 Fair Data Trading Protocol (Formal)

In this section, we formally define the generic construction of our fair data trading (FDT) protocol using the communication model described in Section 4. FDT is a PRE with additional functions such that anyone can verify if the re-encryption key can truly update a ciphertext under one public key to another public key and if the original ciphertext is a valid one from some trust data manager.

Definition 15 (FDT)**.** A *fair data trading* protocol FDT is defined in Fig. 5 and Fig. 6 and is parameterized by the following components.

- a signature scheme Σ,
- a SKE scheme SKE,
- a PRE scheme PRE,
- the re-key verification process consists
  - a statement generation algorithm Statement,
  - a blind statement algorithm Blind,
  - a re-key prove algorithm RkProve
  - a re-key verification algorithm RkVerify.

Remark 1. The blind statement algorithm Blind is defined to generate a blinded statement, which can be used to prove the validity of a re-key without reveal anything about the original ciphertext. A re-key verification algorithm RkVerify is defined to check the validity of a re-key, it outputs a bit that is either 0 or 1 on input a re-key, a blinded statement and a proof.

DS running oracle $\prod^{\alpha}_{\mathsf{DS}}$ on input tag:

1. $(\mathsf{srk}, \mathsf{prk}) \xleftarrow{\$} \mathsf{PRE.KG}(\lambda)$
2. Send† $(\mathsf{tag}, \mathsf{prk}, \mathsf{m})$ to DM

DM running oracle $\prod^{\beta}_{\mathsf{DM}}$ on message $(\mathsf{tag}, \mathsf{prk}, \mathsf{m})$ from DS:

3. Authenticate† DS, check validity of $(\mathsf{tag}, \mathsf{m})$
4. $\mathsf{k} \xleftarrow{\$} \mathcal{K}$
5. $\mathsf{c_m} \leftarrow \mathsf{SKE.Enc}(\mathsf{k}, \mathsf{m})$
6. $\mathsf{c_k} \leftarrow \mathsf{PRE.Enc}(\mathsf{prk}, \mathsf{k})$
7. $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{S}, \langle \mathsf{H}(\mathsf{prk}), \mathsf{H}(\mathsf{c_k}), \mathsf{H}(\mathsf{c_m}), \mathsf{tag} \rangle)$
8. Send $(\mathsf{DM}, \mathsf{tag}, \mathsf{prk}, \mathsf{c_k}, \mathsf{c_m}, \sigma)$ to DS

DS running oracle $\prod^{\alpha}_{\mathsf{DS}}$ on input $(\mathsf{DM}, \mathsf{tag}, \mathsf{prk}, \mathsf{c_k}, \mathsf{c_m}, \sigma)$:

9. Verify that σ is DM's signature on $(\mathsf{tag}, \mathsf{prk}, \mathsf{c_k}, \mathsf{c_m})$
10. $\mathsf{k} \leftarrow \mathsf{PRE.Dec}(\mathsf{srk}_i, \mathsf{c_k})$
11. $\mathsf{m}^{\alpha}_{\mathsf{DS}} \leftarrow \mathsf{SKE.Dec}(\mathsf{k}, \mathsf{c_m})$
12. Store $\{\mathsf{srk}, \mathsf{DM}, \mathsf{tag}, \mathsf{prk}, \mathsf{c_k}, \mathsf{c_m}, \sigma\}$

Figure 5: Roles in the pre-trading phase. Note that the line numbering indicates the order of a protocol execution. †: we omit the details of the signing and verifying processes between DS and DM.

SC running oracle $\prod^{\omega}_{\mathsf{SC}}$ on input $(\mathsf{pk_{DC}}, \mathsf{crs}, \mathsf{R}, \mathsf{deposit}, \{\mathsf{tag}\}_{\mathsf{tag}\in T})$ from DC:

1. Register $(\mathsf{pk_{DC}}, \mathsf{crs}, \mathsf{R}, \mathsf{deposit}, \{\mathsf{tag}\}_{\mathsf{tag}\in T})$ on the smart contract

SC running oracle $\prod^{\omega}_{\mathsf{SC}}$ on input address from DS:

2. Register address on the smart contract

DS running oracle $\prod^{\gamma}_{\mathsf{DS}}$ on input $\{\mathsf{tag}\}_{\mathsf{tag}\in T}$:

3. Find $\{\mathsf{srk}, \mathsf{DM}, \mathsf{tag}, \mathsf{prk}, \mathsf{c_k}, \mathsf{c_m}, \sigma\}_{\mathsf{tag}\in T}$
4. $\mathsf{rk} \leftarrow \mathsf{PRE.ReKG}(\mathsf{srk}, \mathsf{pk_{DC}}, \mathsf{c_k})$
5. $\mathsf{aux} \leftarrow \mathsf{Statement}(\mathsf{pk_{DC}}, \mathsf{srk}, \mathsf{prk}, \mathsf{st}, \mathsf{c_k})$
6. Send $\{(\mathsf{DM}, \mathsf{tag}, \mathsf{prk}, \mathsf{c_k}, \mathsf{c_m}, \sigma, \mathsf{aux})\}_{\mathsf{tag}\in T}$ to DC

DC running oracle $\prod^{\upsilon}_{\mathsf{DC}}$ on message $\{(\mathsf{DM}, \mathsf{tag}, \mathsf{prk}, \mathsf{c_k}, \mathsf{c_m}, \sigma, \mathsf{aux})\}_{\mathsf{tag}\in T}$ from DS:

7. Verify that σ is DM's signature on $(\mathsf{tag}, \mathsf{prk}, \mathsf{c_k}, \mathsf{c_m})$ for each $\mathsf{tag} \in T$
8. If the above verification failed then
   send reject to SC
   else
   for $\mathsf{tag} \in T$ do
   compute $\mathsf{bs} \leftarrow \mathsf{Blind}(\mathsf{srk}, \mathsf{c}, \mathsf{aux})$
   send confirm and $\{\mathsf{bs}\}_{\mathsf{tag}\in T}$ to SC

SC running oracle $\prod^{\omega}_{\mathsf{SC}}$ on input decision and $\{\mathsf{bs}\}_{\mathsf{tag}\in T}$ from DC:

9. Record the decision on address
10. Record $\{\mathsf{bs}\}_{\mathsf{tag}\in T}$ on SC

DS running oracle $\prod^{\gamma}_{\mathsf{DS}}$ by reading the decision and $\{\mathsf{bs}\}_{\mathsf{tag}\in T}$ on SC:

11. If decision is confirm then
    For $\mathsf{tag} \in T$ do
    $\pi \leftarrow \mathsf{RkProve}(\mathsf{rk}, \mathsf{srk}, \mathsf{st})$
    Compute $\mathsf{RkVerify}(\mathsf{rk}, \mathsf{bs}, \pi)$
    If all above outputs are 1 then
    send $\{\mathsf{rk}, \pi\}_{\mathsf{tag}\in T}$ to SC
    else
    send reject to SC

SC running oracle $\prod^{\omega}_{\mathsf{SC}}$ on input $\{\mathsf{rk}, \pi\}_{\mathsf{tag}\in T}$ from DS:

12. Record $\{\mathsf{rk}\}_{\mathsf{tag}\in T}$ on SC
13. For $\mathsf{tag} \in T$ do
    If $\mathsf{RkVerify}(\mathsf{rk}, \mathsf{bs}, \pi) \neq 1$ then
    reject
    send reward R to DS

DC running oracle $\prod^{\upsilon}_{\mathsf{DC}}$ by reading $\{\mathsf{rk}\}_{\mathsf{tag}\in T}$ on SC:

14. For $\mathsf{tag} \in T$ do
    $\mathsf{c} \leftarrow \mathsf{PRE.ReEnc}(\mathsf{rk}, \mathsf{c_k})$
    $\mathsf{k} \leftarrow \mathsf{PRE.Dec}(\mathsf{sk_{DC}}, \mathsf{c})$
    $\mathsf{m} \leftarrow \mathsf{SKE.Dec}(\mathsf{k}, \mathsf{c_m})$

Figure 6: Roles in the trading phase. Note that the line numbering indicates the order of a protocol execution.

6 Constructing a Fair Data Trading Protocol

We construct a concrete instantiation of a fair data trading protocol using a pairing based PRE scheme PRE (see Section 6.1), blind algorithm Blind and re-key verification algorithm RkVerify (see Section 6.3), the pairing based FDT protocol is denoted by PFDT. We show that our PFDT protocol has correctness, soundness, real or random indistinguishability, unforgeability in Section 6.4. Hence, by Theorem 1, PFDT has fairness to both data subject and data collector.

6.1 Constructing a PRE Scheme

A pairing based PRE scheme PRE is defined in Fig. 7.

$\underline{\mathsf{KG}(\lambda, \mathbb{G}_1):}$
$\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_q$
$\mathsf{pk} \leftarrow g_1^{\mathsf{sk}}$
return $(\mathsf{sk}, \mathsf{pk})$

$\underline{\mathsf{KG}(\lambda, \mathbb{G}_2):}$
$\mathsf{srk} \xleftarrow{\$} \mathbb{Z}_q$
$\mathsf{prk} \leftarrow g_2^{\mathsf{srk}}$
return $(\mathsf{srk}, \mathsf{prk})$

$\underline{\mathsf{Enc}(\mathsf{prk}, \mathsf{k}):}$
$r_1, r_2 \xleftarrow{\$} \mathbb{Z}_q$
$\mathsf{c}_1 \leftarrow g_1^{r_1}$
$\mathsf{c}_2 \leftarrow \mathsf{prk}^{\frac{r_2}{r_1}}$
$\mathsf{c}_3 \leftarrow e(g_1, g_2)^{r_2} \cdot \mathsf{k}$
$\mathsf{c}_\mathsf{k} \leftarrow (\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3)$
return c_k

$\underline{\mathsf{Dec}(\mathsf{sk}^\dagger, \mathsf{c}_\mathsf{k}):}$
parse $\mathsf{c}_\mathsf{k} = (\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3)$
$\mathsf{k} \leftarrow \frac{\mathsf{c}_3}{e(\mathsf{c}_1, \mathsf{c}_2)^{\frac{1}{\mathsf{sk}}}}$
return k

$\underline{\mathsf{ReKG}(\mathsf{srk}_i, \mathsf{pk}_j, \mathsf{c}_\mathsf{k}):}$
parse $\mathsf{c}_\mathsf{k} = (\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3)$
$t \xleftarrow{\$} \mathbb{Z}_q$
$\mathsf{rk}_1 \leftarrow \mathsf{c}_1^{\frac{1}{\mathsf{srk}_i}} g_1^{-t}$
$\mathsf{rk}_2 \leftarrow \mathsf{pk}_j^t$
return $(\mathsf{rk}_1, \mathsf{rk}_2)$

$\underline{\mathsf{ReEnc}(\mathsf{rk}_{i,j}, \mathsf{c}_\mathsf{k}):}$
parse $\mathsf{rk}_{i,j} = (\mathsf{rk}_1, \mathsf{rk}_2)$
parse $\mathsf{c}_\mathsf{k} = (\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3)$
$\mathsf{c}'_3 \leftarrow \frac{\mathsf{c}_3}{e(\mathsf{rk}_1, \mathsf{c}_2)}$
$\mathsf{c}'_\mathsf{k} \leftarrow (\mathsf{rk}_2, \mathsf{c}_2, \mathsf{c}'_3)$
return c'_k

Figure 7: PRE scheme. In our PFDT protocol, data collector oracles run $\mathsf{KG}(\lambda, \mathbb{G}_1)$ to generate long-term key pair $(\mathsf{sk}, \mathsf{pk})$ and data subject oracles run $\mathsf{KG}(\lambda, \mathbb{G}_2)$ to generate one time key pair $(\mathsf{srk}, \mathsf{prk})$. †: srk can be the decryption key as well, srk is the decryption key for encrypted ciphertext and sk is the decryption key for re-encrypted ciphertext.

Correctness of PRE**.** We prove the correctness of PRE by proving the correctness of decryption algorithm on both the original cipehrtext and the re-encrypted cipehrtext. Let $\mathsf{c}_\mathsf{k} = (\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3) = (g_1^{r_1}, \mathsf{prk}^{\frac{r_2}{r_1}}, e(g_1, g_2)^{r_2} \cdot \mathsf{k})$ be an original ciphertext, the corresponding re-key is $\mathsf{rk}_{i,j} = (\mathsf{rk}_1, \mathsf{rk}_2) = (\mathsf{c}_1^{\frac{1}{\mathsf{srk}_i}} g_1^{-t}, \mathsf{pk}_j^t)$, the re-encrypted ciphertext is $\mathsf{ReEnc}(\mathsf{rk}_{i,j}, \mathsf{c}_\mathsf{k}) = (\mathsf{c}'_1, \mathsf{c}'_2, \mathsf{c}'_3) = (\mathsf{rk}_2, \mathsf{c}_2, \frac{\mathsf{c}_3}{e(\mathsf{rk}_1, \mathsf{c}_2)}) = (\mathsf{pk}_j^t, \mathsf{prk}_i^{\frac{r_2}{r_1}}, e(g_1, g_2)^{\frac{\mathsf{srk}_i t r_2}{r_1}} \cdot \mathsf{k})$. Then we have

- $\mathsf{Dec}(\mathsf{srk}, \mathsf{c}_\mathsf{k}) = \frac{\mathsf{c}_3}{e(\mathsf{c}_1, \mathsf{c}_2)^{\frac{1}{\mathsf{srk}}}} = \frac{e(g_1, g_2)^{r_2} \cdot \mathsf{k}}{e(g^{r_1}, \mathsf{prk}^{\frac{r_2}{r_1}})^{\frac{1}{\mathsf{srk}}}} = \mathsf{k}$
- $\mathsf{Dec}(\mathsf{sk}_j, \mathsf{ReEnc}(\mathsf{rk}_{i,j}, \mathsf{c}_\mathsf{k})) = \frac{\mathsf{c}'_3}{e(\mathsf{c}'_1, \mathsf{c}'_2)^{\frac{1}{\mathsf{sk}_j}}} = \frac{e(g_1, g_2)^{\frac{\mathsf{srk}_i t r_2}{r_1}} \cdot \mathsf{k}}{e(\mathsf{pk}_j^t, \mathsf{prk}_i^{\frac{r_2}{r_1}})^{\frac{1}{\mathsf{sk}_j}}} = \mathsf{k}$

6.2 Constructing Commit-and-Prove Zero-knowledge Proofs

In this section we construct two zero-knowledge proofs, their goal is to ensure the rekey of the PRE scheme $(\mathsf{rk}_1, \mathsf{rk}_2)$ is generated correctly. Precisely, the inner product of the exponents of rk_1 and prk_i has a linear relation with the exponent of c_1 and prk_i^t, that is

$$\underbrace{(\frac{r_1}{\mathsf{srk}_i} - t)}_{\text{exponent of } \mathsf{rk}_1} \cdot \mathsf{srk}_i = \underbrace{r_1}_{c_1} - \underbrace{\mathsf{srk}_i t}_{\mathsf{prk}_i^t}$$

$\mathcal{L}_1 = \{g, h, c_a, c_b, c \mid \exists t \text{ s.t. } c = c_a c_b h^t\}$

$\underline{\mathsf{ZK}_1\_\mathsf{Prove}(g, h, t):}$

$\mathcal{P}: \alpha \xleftarrow{\$} \mathbb{Z}_q$
$\quad d \leftarrow h^\alpha$
$\mathcal{P} \rightarrow \mathcal{V}: A$
$\mathcal{V}: x \xleftarrow{\$} \mathbb{Z}_q$
$\mathcal{P}: \theta = \alpha - xt$
$\quad \pi \leftarrow (d, x, \theta)$
$\mathcal{P} \rightarrow \mathcal{V}: \pi$

$\underline{\mathsf{ZK}_1\_\mathsf{Verify}(g, h, c_a, c_b, c, \pi):}$

$\mathcal{V}$: parse $\pi = (d, x, \theta)$
$\quad$ **if** $(\frac{c}{c_a c_b})^x h^\theta = d$ **then**
$\quad\quad$ **return** 1
$\quad$ **else**
$\quad\quad$ **return** 0

$\mathcal{L}_2 = \{g, h, c_a, c_b, c \mid \exists a, b, r_a, r_b, t \text{ s.t.}$
$\quad c_a = g^a h^{r_a}, c_b = g^b h^{r_b}, c = g^{ab} h^t\}$

$\underline{\mathsf{ZK}_2\_\mathsf{Prove}(g, h, a, b, r_a, r_b, t):}$

$\mathcal{P}: \alpha, \beta, r_1, r_2, s_0, s_1 \xleftarrow{\$} \mathbb{Z}_q$
$\quad d_1 \leftarrow g^\alpha h^{r_1}, d_2 \leftarrow g^\beta h^{r_2},$
$\quad c_0 = g^{\alpha b + \beta a} h^{s_0}, c_1 = g^{\alpha\beta} h^{s_1}$
$\mathcal{P} \rightarrow \mathcal{V}: (d_1, d_2, c_0, c_1)$
$\mathcal{V}: x \xleftarrow{\$} \mathbb{Z}_q$
$\mathcal{P}: \theta_a = \alpha - ax, \theta_b = \beta - bx,$
$\quad \theta_1 = r_1 - r_a x, \theta_2 = r_2 - r_b x$
$\quad \theta_{ab} = x^2 t - x s_0 + s_1$
$\quad \pi \leftarrow (d_1, d_2, c_0, c_1, x, \theta_a, \theta_b, \theta_1, \theta_2, \theta_{ab})$
$\mathcal{P} \rightarrow \mathcal{V}: \pi$

$\underline{\mathsf{ZK}_2\_\mathsf{Verify}(g, h, c_a, c_b, c, \pi):}$

$\mathcal{V}:$
parse $\pi = (d_1, d_2, c_0, c_1, x, \theta_a, \theta_b, \theta_1, \theta_2, \theta_{ab})$
$\quad$ **if** $c_a^x g^{\theta_a} h^{\theta_1} = d_1$ **and**
$\quad\quad c_b^x g^{\theta_b} h^{\theta_2} = d_2$ **and**
$\quad\quad g^{\theta_a \theta_b} h^{\theta_{ab}} c_0^x = c^{x^2} c_1$ **then**
$\quad\quad$ **return** 1
$\quad$ **else**
$\quad\quad$ **return** 0

Figure 8: Zero-knowledge proof ZK_1 and ZK_2.

We use Pedersen commitment scheme to hide relevant values on smart contract, and build two interactive zero-knowledge proofs: an addition proof (ZK_1) an and an inner product proof (ZK_2) that intended to prove the above relation among commitments. These zero-knowledge proofs can can be transferred to non-interactive zero-knowledge proofs $\mathsf{NIZK}_1, \mathsf{NIZK}_2$ respectively by using Fiat-Shamir heuristic [BR95]. The construction is in Fig.8. The security properties of $\mathsf{ZK}_1, \mathsf{ZK}_2$ are claimed in Theorem 2 and its proof is given in Appendix A.1

Theorem 2. The commit-and-proof zero-knowledge proofs ZK_1 and ZK_2 have perfect completeness, computational soundness and perfect special honest-verifier zero-knowledge.

6.3 Re-key Verification

The statement generation algorithm Statement, blind statement algorithm Blind, re-key prove and verification algorithm RkProve, RkVerify are defined in Fig. 9. Statement and RkProve are run by data subject; Blind is run by the data collector; RkVerify is run by smart contract. The output of the blind algorithm is submitted to the smart contract by the data collector. It is worthwhile to give some technical design intentions about the algorithms: (1) We use zero-knowledge proof instead of pairing to verify the correction of the rekey because the unlinkability between different verification instances follows trivially from the zero-knowledge property. (2) We move all the computation on the smart contract to be in group $\mathbb{G}_1$ instead of $\mathbb{G}_1, \mathbb{G}_2$, since the computation on $\mathbb{G}_1$ is more efficient than $\mathbb{G}_2$ in the elliptic curves used by the current blockchain.

6.4 Properties

We show our PFDT protocol has correctness and soundness in Theorem 3, and give their proofs in Appendix A. Furthermore, we prove our PFDT protocol satisfies real or random indistinguishability and unforgeability (recall the definitions in Section 4.2). Full proofs for Theorems 4, 5, 6 and 7, are given in Appendix A. Since proof techniques are similar, we give a proof sketch only for Theorem 4.

Theorem 3 (Correctness and Soundness)**.** PFDT has soundness and correctness if the underlying signature scheme Σ, SKE scheme SKE are correct.

$\underline{\mathsf{Statement}(\mathsf{pk}_j, \mathsf{prk}_i, \mathsf{srk}_i, \mathsf{c}_1, r_1):}$

$t \xleftarrow{\$} \mathbb{Z}_q$
$\tau_1 \leftarrow \mathsf{pk}_j^t$
$\tau_2 \leftarrow \mathsf{prk}_i^t$
$s_1 \xleftarrow{\$} \mathbb{Z}_q$
$s_2 \xleftarrow{\$} \mathbb{Z}_q$
$\tau_3 \leftarrow g_1^{r_1 - \mathsf{srk}_i t} h_1^{s_1}$
$\tau_4 \leftarrow g_1^{\mathsf{srk}_i}$
$\pi_1 \leftarrow \mathsf{NIZK_1\_Prove}(g_1, h_1, t)$
$\mathsf{aux} \leftarrow (\tau_1, \tau_2, \tau_3, \tau_4, s_2, \pi_1)$
return aux

$\underline{\mathsf{RkProve}(g_1, h_1, \mathsf{rk}_1, \mathsf{srk}_i, t, s_1, s_2):}$

$a \leftarrow \frac{r_1}{\mathsf{srk}_i} - t$
$\pi_2 \leftarrow \mathsf{NIZK_2\_Prove}(g_1, h_1, a, \mathsf{srk}_i, 0, s_2, s_1)$
return rk_1, π_2

$\underline{\mathsf{Blind}(\mathsf{sk}_j, \mathsf{c}_i, \mathsf{aux}):}$

parse $\mathsf{c}_i = (\mathsf{prk}_i, \mathsf{c_k}, \mathsf{c_m})$
parse $\mathsf{c_k} = (\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3)$
parse $\mathsf{aux} = (\tau_1, \tau_2, \tau_3, \tau_4, s_2, \pi_1)$
if $e(\mathsf{pk}, \tau_2) = e(\tau_1, \mathsf{prk})$ **and**
  $e(\tau_4, g_2) = e(g_1, \mathsf{prk})$ **and**
  $\mathsf{NIZK_1\_Verify}(g_1, h_1, c_1, \tau_2, \tau_3, \pi_1)$ **then**
  $\mathsf{bs}_1 \leftarrow \tau_3$
  $\mathsf{bs}_2 \leftarrow \tau_4 h_1^{s_2}$
  $\mathsf{bs} \leftarrow (\mathsf{bs}_1, \mathsf{bs}_2)$
  return bs, τ_1
else
  return $\perp$

$\underline{\mathsf{RkVerify}(\mathsf{rk}_1, \mathsf{rk}_2, \mathsf{bs}, \tau_1, \pi_2):}$

parse $\mathsf{bs} = (\mathsf{bs}_1, \mathsf{bs}_2)$
if $\mathsf{NIZK_2\_Verify}(g_1, h_1, \mathsf{rk}_1, \mathsf{bs}_2, \mathsf{bs}_1, \pi_2)$ **and**
  $\tau_1 = \mathsf{rk}_2$ **then**
  return 1
else
  return 0

Figure 9: The blind algorithm Blind and the re-key verification algorithm RkVerify.

6.4.1 Real or Random Indistinguishability.

Theorem 4. For any M-RoR adversary $\mathcal{A}$ against PFDT running with n data subjects, having at most s sessions in the trading phase, then there exist adversaries $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$, such that

$$\mathbf{Adv}^{\mathsf{M\text{-}ROR}}_{\mathsf{PFDT},\mathcal{A}}(\lambda) \leq sn(\mathbf{Adv}^{\mathsf{Pub\text{-}RoR}}_{\mathsf{PFDT},\mathcal{B}_1}(\lambda) + \mathbf{Adv}^{\mathsf{co\text{-}DDBDH}}_{(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T),\mathcal{B}_2} + \mathbf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SKE},\mathcal{B}_3}(\lambda)).$$

Proof Sketch. The proof of this theorem consists a sequence of games. In the first game, we guess which session in the trading phase the adversary is going to issue the M-RoRChall query for and which data subject is involved in this session. The game returns a random bit for b' if the guess is wrong. The security lose is upper bounded by sn.

In the second game, if the adversary issues a corruption query that would make the challenge session non-fresh the game returns a random bit for b'. The advantage of any adversary wins the second game is unchanged.

In the third game, we change all the public values involved in this session to be public values generated from a random ciphertext. The advantage of noticing this modification is upper bounded by the Pub-RoR advantage for PFDT.

In the fourth game, we replace the $\mathsf{c_k}$ values of the involved data subject to random ciphertexts, the advantage of distinguishing this change is upper bounded by the co-DDBDH advantage.

In the end, we change the $\mathsf{c_m}$ term to random ciphertexts, the advantage of identifying this change is upper bounded by the IND-CPA advantage for SKE. Now, all values relate to the M-RoRChall query are random. So the advantage of winning the last game is 0.

Theorem 5. For any Pub-RoR adversary $\mathcal{A}$ against PFDT, $\mathbf{Adv}^{\mathsf{Pub\text{-}RoR}}_{\mathsf{PFDT},\mathcal{A}}(\lambda) = 0$.

6.4.2 Unforgeability

Theorem 6. For any SigUF adversary $\mathcal{A}$ against PFDT running with m data managers, then there exists an adversary $\mathcal{B}$, such that

$$\mathbf{Adv}^{\mathsf{SigUF}}_{\mathsf{PFDT},\mathcal{A}}(\lambda) \leq m\mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\Sigma,\mathcal{B}}.$$

Theorem 7. For any RkUF adversary $\mathcal{A}$ against PFDT running with n data subjects, having at most s sessions in the trading phase, then there exist adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$, such that

$$\mathbf{Adv}^{\mathsf{RkUF}}_{\mathsf{PFDT},\,\mathcal{A}}(\lambda) \leq sn(\mathbf{Adv}^{\mathsf{co\text{-}DDBDH}}_{(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T),\mathcal{B}_1}(\lambda) + \mathbf{Adv}^{\mathsf{co\text{-}DCDH}}_{(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T),\mathcal{B}_2}(\lambda)).$$

7 PFDT Evaluation and Fees

Function	Cost in gas	Transaction size (byte)
Deploy Smart Contract	2915663	26168
Data Subject Register	51072	10
Data Collector Confirm	137862	330
Re-key Verification	133154	1034

Table 1: Evaluation of PFDT

In this section we discuss our implementation and evaluation on the PFDT protocol. The on-chain part: trading procedures that need smart contract are evaluated on Ethereum Ropsten testnet. We use the elliptic curve BN-128 [bn1] to implement the algorithm, since Ethereum supports pre-compiled contract for this elliptic curve. Compared to using other elliptic curve library contracts and implementing them directly, using precompiled contracts can reduce cost. Table 1 shows the gas cost for different function calls.

8 Further Discussion

The most financial cost in our system is the verification algorithm on the smart contract. Depending on the underlying cryptocurrencies, the price for verification varies a lot. With current Ethereum blockchain, one may not desire to do a data trading with at least around 6 dollars (simplest transaction cost) extra cost for a blockchain transaction, but it can be possible for other cheaper blockchain based cryptocurrencies. In addition, in some use cases the smart contract can be run on a permission/private blockchain. In our PFDT protocol, We require the smart contract compute the re-key verification algorithm for each tag related information. It is possible to design a method that aggregate multiple re-key verification into one, this can be done based on the updatability of PRE: data subject first updates multiple $\mathsf{c_k}$ into ciphertexts that are encrypted by the same public key, and then trade only one re-key with the data collector.

References

[ABH09] Giuseppe Ateniese, Karyn Benson, and Susan Hohenberger. Key-private proxy re-encryption. In Marc Fischlin, editor, *Topics in Cryptology - CT-RSA 2009, The Cryptographers' Track at the RSA Conference 2009, San Francisco, CA, USA, April 20-24, 2009. Proceedings*, volume 5473 of *Lecture Notes in Computer Science*, pages 279–294. Springer, 2009. Cited on page 4.

[ASW98] N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. In Kaisa Nyberg, editor, *Advances in Cryptology — EUROCRYPT'98*, pages 591–606, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. Cited on page 1.

[BBS98] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In Kaisa Nyberg, editor, *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, volume 1403 of *Lecture Notes in Computer Science*, pages 127–144. Springer, 1998. Cited on page 3.

[BCP02] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Dynamic group diffie-hellman key exchange under standard assumptions. In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*,

volume 2332 of *Lecture Notes in Computer Science*, pages 321–336. Springer, 2002. Cited on page 9.

[BDZ03] Feng Bao, Robert H. Deng, and Huafei Zhu. Variations of diffie-hellman problem. In Sihan Qing, Dieter Gollmann, and Jianying Zhou, editors, *Information and Communications Security, 5th International Conference, ICICS 2003, Huhehaote, China, October 10-13, 2003, Proceedings*, volume 2836 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 2003. Cited on page 4.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer, 2003. Cited on page 4.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 410–428. Springer, 2013. Cited on pages 3 and 4.

[BM08] Emmanuel Bresson and Mark Manulis. Securing group key exchange against strong corruptions. In Masayuki Abe and Virgil D. Gligor, editors, *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2008, Tokyo, Japan, March 18-20, 2008*, pages 249–260. ACM, 2008. Cited on page 9.

[bn1] Precompiled contracts for addition and scalar multiplication on the elliptic curve alt-bn128. Cited on page 18.

[BR95] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. pages 62–73. ACM Press, 1995. Cited on page 16.

[CD00] Jan Camenisch and Ivan Damgård. Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In *Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '00, page 331–345, Berlin, Heidelberg, 2000. Springer-Verlag. Cited on page 1.

[CGGN17] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. *Zero-Knowledge Contingent Payments Revisited: Attacks and Payments for Services*, page 229–243. Association for Computing Machinery, New York, NY, USA, 2017. Cited on pages 1 and 2.

[CH07] Ran Canetti and Susan Hohenberger. Chosen-ciphertext secure proxy re-encryption. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 185–194. ACM, 2007. Cited on pages 3 and 4.

[DEF18a] Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. Fairswap: How to fairly exchange digital goods. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 967–984, New York, NY, USA, 2018. Association for Computing Machinery. Cited on pages 1 and 2.

[DEF18b] Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. Fairswap: How to fairly exchange digital goods. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 967–984. ACM, 2018. Cited on page 12.

[DR03] Yevgeniy Dodis and Leonid Reyzin. Breaking and repairing optimistic fair exchange from podc 2003. In *Proceedings of the 3rd ACM Workshop on Digital Rights Management*, DRM '03, page 47–54, New York, NY, USA, 2003. Association for Computing Machinery. Cited on page 1.

[EFS20] Lisa Eckey, Sebastian Faust, and Benjamin Schlosser. Optiswap: Fast optimistic fair exchange. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '20, page 543–557, New York, NY, USA, 2020. Association for Computing Machinery. Cited on pages 1 and 2.

[Hal19] John David Halamka. Real blockchain use cases for healthcare. In *Blockchain in healthcare Today*, 2019. Cited on page 3.

[HYWS08] Qiong Huang, Guomin Yang, Duncan S. Wong, and Willy Susilo. Ambiguous optimistic fair exchange. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, pages 74–89, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. Cited on page 1.

[JWCBK15] Bram JT, Obeysekare E Warwick-Clark B, and Mehta K. Utilization and monetization of healthcare data in developing countries. *Big Data*, 2015. https://viral.media.mit.edu/pub/medrec. Cited on page 3.

[KL10] Alptekin Küpçü and Anna Lysyanskaya. Usable optimistic fair exchange. In Josef Pieprzyk, editor, *Topics in Cryptology - CT-RSA 2010*, pages 252–267, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. Cited on page 1.

[LV11] Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. *IEEE Trans. Inf. Theory*, 57(3):1786–1802, 2011. Cited on page 4.

[PD99] H. Pagnia and Felix C. Gartner Darmstadt. On the impossibility of fair exchange without a trusted third party. 1999. Cited on page 1.

[SNS11] Amril Syalim, Takashi Nishide, and Kouichi Sakurai. Realizing proxy re-encryption in the symmetric world. In Azizah Abd Manaf, Akram Zeki, Mazdak Zamani, Suriayati Chuprat, and Eyas El-Qawasmeh, editors, *Informatics Engineering and Information Science*, pages 259–274, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. Cited on pages 3 and 4.

A Proofs of Theorems

A.1 Proof of Theorem 2

The proof for ZK_1

Proof. The perfect completeness follows by the protocol. To prove the soundness, we rewind the prover til it generates d, and run the protocol again but with a different challenger x', we two accepting transcripts $(d, x, \theta), (d, x', \theta')$.

$$(\frac{c}{c_a c_b})^x h^\theta = (\frac{c}{c_a c_b})^{x'} h^{\theta'} = d$$

Assume $c = g^z h^t$, the above equation implies $g^{z-a-b} h^{t(x-x')} h^{\theta-\theta'} = 1$, hence, if $z \neq a + b$ one can get a non-trial relation between g, h. To prove the perfect special honest-verifier zero-knowledge, the simulator randomly chooses θ and a randomly chooses a challenge x, compute $d = (\frac{c}{c_a c_b})^x h^\theta$, then the simulator simulates a valid transcript (d, x, θ) which has the identical distribution with the real proof.

□

The proof for ZK_2

Proof. The perfect completeness follows by the protocol.

Soundness. By rewinding the prover til it generates the commitment d_a, d_2, c_0, c_1, the extractor $\mathcal{X}$ gets two valid transcripts that have the same commitments:

$(d_1, d_2, c_0, c_1, x, \theta_a, \theta_b, \theta_1, \theta_2, \theta_{ab})$, $(d_1, d_2, c_0, c_1, x', \theta'_a, \theta'_b, \theta'_1, \theta'_2, \theta'_{ab})$

from the verification, we get equations

$$c_a^x g^{\theta_a} h^{\theta_1} = d_1 \qquad c_a^{x'} g^{\theta'_a} h^{\theta'_1} = d_1$$

By the binding property of Pedersen commitment, This implies $a = \frac{\theta'_a - \theta_a}{x - x'}$, by the same technique, $\mathcal{X}$ can compute $b = \frac{\theta'_b - \theta_b}{x - x'}$ and α, β.

Next, assume c is a commitment that committed to z, we will prove $z = ab$. Assume $c_0 = g^u h^{r_{c_0}}, c_1 = g^v h^{r_{c_1}}$, observe that $g^{\theta_a \theta_b} h^{\theta_{ab}} c_0^x = c^{x^2} c_1$, it implies

$$g^{abx^2 - (a\beta + b\alpha)x + \alpha\beta + ux} h^{\theta_{ab} + x r_{c_0}} = g^{zx^2 + v} h^{r_c x^2 + r_{c_1}}$$

Since $a, b, \alpha, \beta, u, v$ are all predefine value, either $\mathcal{X}$ can extract non-trivial relation between g, h or $u = \alpha b + \beta a$ and the extractor can extract $z = ab = \frac{\theta_a \theta_b - \theta'_a \theta'_b + (\alpha b + \beta a)(x - x')}{x^2 - x'^2}$

Perfect special honest-verifier zero-knowledge. The simulator randomly chooses $\theta_1, \theta_2, \theta_a, \theta_b, \theta_{ab}, u, r \leftarrow \mathbb{Z}_p$ and randomly chooses a challenge $x \leftarrow \mathbb{Z}_p$, it computes $d_1 = c_a^x g^{\theta_a} h^{\theta_1}$, $d_2 = c_b^x g^{\theta_b} h^{\theta_2}$, $c_0 = g^u h^r$, $c_1 = g^{\theta_a \theta_b} h^{\theta_{ab}} c_0^x / c^{x^2}$. Thus the simulator produces a valid transcript $(d_1, d_2, c_0, c_1, x, \theta_a, \theta_b, \theta_1, \theta_2, \theta_{ab})$ that has the identical probability distributions with the real proof. □

A.2 Proof of Theorem 3

Proof. We prove PFDT has soundness first. When we have $\mathsf{RkVerify}(\mathsf{rk}, \mathsf{bs}, \pi) = 1$, according to the soundness of zero-knowlege proof ZK_2 (Theorem 2), we get relation

$$\{\mathsf{rk}_1, \mathsf{bs}_1, \mathsf{bs}_2 | \exists a, b, z, r_b, r_z \text{ s.t. } ab = z, \mathsf{rk}_1 = g_1^a, \mathsf{bs}_1 = g_1^z h_1^{r_z}, \mathsf{bs}_2 = g_1^b h^{r_b}\}$$

Since DC computes algorithm Blind honestly, from the equations $e(\tau_4, g_2) = e(g_1, \mathsf{prk}), e(\mathsf{pk}, \tau_2) = e(\tau_1, \mathsf{prk})$, we get $\tau_4 = g_1^{\mathsf{srk}}$ and $\tau_2 = \mathsf{prk}^t, \tau_1 = \mathsf{pk}^t$.

According to the soundness of zero-knowledge proof ZK_1, we get relation

$$\{c_1, \mathsf{prk}^t, \mathsf{bs}_1 | \exists r_1, z \text{ s.t. } r_1 - \mathsf{srk} t = z, c_1 = g_1^{r_1}\}$$

$\mathsf{bs}_2 = \tau_4 h_1^{s_2} = g_1^{\mathsf{srk}} h_1^{s_2}$, this implies $a = \frac{r_1}{\mathsf{srk}} + t$, therefore $\mathsf{rk}_1 = g_1^{\frac{r_1}{\mathsf{srk}} + t}, rk_2 = \tau_1 = \mathsf{pk}^t$, which are correct re-keys for the PRE scheme.

Next, we prove correctness. Correctness of PRE is proved in Section 6.1. Soundness of our protocol ensures that if the re-key verification algorithm outputs 1, then the data collector and data subject will agree on the same message. For any honestly generated blind statement $\mathsf{bs} = (\mathsf{bs}_1, \mathsf{bs}_2) = (g_1^{r_1 - \mathsf{srk}_i t}, \mathsf{prk}_i h_1^{s_2})$ and corresponding re-key $\mathsf{rk}_1 = \mathsf{c}_1^{\frac{1}{\mathsf{srk}_i}} g^{-t} = g_1^{\frac{r_1}{\mathsf{srk}_i} - t}$, they satisfy the inner product relation $(\frac{r_1}{\mathsf{srk}_i} - t) \cdot \mathsf{srk}_i = r_1 - \mathsf{srk}_i t$, thus its proof will be accepted by NIZK_2_Verify, which means after an honest run of our protocol, the re-key verification algorithm will output 1.

□

A.3 Proof of Theorem 4

Proof. The proof of this theorem consists a sequence of games. For $\mathrm{b} \in \{0, 1\}$,

Game 0

The first game $\mathcal{G}_0^{\mathrm{b}}$ is $\mathbf{Exp}^{\mathsf{M\text{-}ROR\text{-}b}}_{\mathsf{FDT}, \mathcal{A}}$, which is given in Def. 12, we have

$$\mathbf{Adv}^{\mathsf{M\text{-}ROR}}_{\mathsf{PFDT}, \mathcal{A}}(\lambda) = \left| \mathbf{Pr}[\mathcal{G}_0^1 = 1] - \mathbf{Pr}[\mathcal{G}_0^0 = 1] \right|.$$

Game 1

This game is the same as $\mathcal{G}_0^{\mathrm{b}}$, except for it guesses which session in the trading phase the adversary is going to issue the M-RoRChall query for and which data subject is involved in this session. The game returns a random bit for b' if the guess is wrong.

We claim that the security lose is upper bounded by sn. Let E_1 be the event that the guessed values are correct. Notice that if any guessed value is incorrect, the game returns a random bit, which means $\mathbf{Pr}[\mathcal{G}_1^{\mathrm{b}} = 1 \mid \neg E_1] = \frac{1}{2}$. In addition, if the guessed values are correct, then $\mathcal{G}_1^{\mathrm{b}}$ is $\mathcal{G}_0^{\mathrm{b}}$. We have

$$\begin{aligned}
\Big|\mathbf{Pr}[\mathcal{G}_1^1 = 1] - \mathbf{Pr}[\mathcal{G}_1^0 = 1]\Big| &= \Big|\mathbf{Pr}[\mathcal{G}_1^1 = 1 \,|\, E_1] - \mathbf{Pr}[\mathcal{G}_1^0 = 1 \,|\, E_1]\Big| \cdot \mathbf{Pr}[E_1] \\
&+ \Big|\mathbf{Pr}[\mathcal{G}_1^1 = 1 \,|\, \neg E_1] - \mathbf{Pr}[\mathcal{G}_1^0 = 1 \,|\, \neg E_1]\Big| \cdot \mathbf{Pr}[\neg E_1] \\
&= \Big|\mathbf{Pr}[\mathcal{G}_1^1 = 1 \,|\, E_1] - \mathbf{Pr}[\mathcal{G}_1^0 = 1 \,|\, E_1]\Big| \cdot \mathbf{Pr}[E_1] \\
&= \Big|\mathbf{Pr}[\mathcal{G}_0^1 = 1] - \mathbf{Pr}[\mathcal{G}_0^0 = 1]\Big| \cdot \frac{1}{sn}.
\end{aligned}$$

Game 2

We modify the game so that if the adversary issues a corruption query that would make the challenge session non-fresh the game returns a random bit for b'.

We prove that the advantage of $\mathcal{G}_2^{\mathrm{b}}$ is the same advantage as $\mathcal{G}_1^{\mathrm{b}}$. Let E_2 be the event that any partner of the challenge session is corrupted. If E_2 does not happen, $\mathcal{G}_2^{\mathrm{b}}$ is $\mathcal{G}_1^{\mathrm{b}}$. If E_2 happens, the adversary will always lose $\mathcal{G}_1^{\mathrm{b}}$, that is, $\mathbf{Pr}[\mathcal{G}_1^{\mathrm{b}} = 1 \mid E_2] = \frac{1}{2}$. Additionally, we have $\mathbf{Pr}[\mathcal{G}_2^{\mathrm{b}} = 1 \mid E_2] = \frac{1}{2}$ by the definition of $\mathcal{G}_2^{\mathrm{b}}$. Then we have

$$\mathbf{Pr}[\mathcal{G}_2^{\mathrm{b}} = 1] = \mathbf{Pr}[\mathcal{G}_1^{\mathrm{b}} = 1].$$

Game 3

In this game, we modify the actions in the data collector oracle (which has a partner as the guessed challenge data subject), instead of computing the session message, it simply output the one outputted by the data subject oracle if all the verification are correct. Then we have

$$\mathbf{Pr}[\mathcal{G}_3^{\mathrm{b}} = 1] = \mathbf{Pr}[\mathcal{G}_2^{\mathrm{b}} = 1].$$

Game 4

We change all the public values involved in the guessed challenge session to be public values generated from a random ciphertext. The advantage is upper bounded by the Pub-RoR advantage for PFDT. That is

$$\Big|\mathbf{Pr}[\mathcal{G}_4^{\mathrm{b}} = 1] - \mathbf{Pr}[\mathcal{G}_3^{\mathrm{b}} = 1]\Big| \leq \mathbf{Adv}^{\mathsf{Pub\text{-}RoR}}_{\mathsf{PFDT},\,\mathcal{B}_1}(\lambda).$$

Game 5

The next modification we make is to replace the ciphertext term $\mathsf{c_k}$ with respect to the guessed data subject to random ciphertexts.

Suppose that $\mathcal{A}_2$ is an adversary attempting to distinguish $\mathcal{G}_5^{\mathrm{b}}$ from $\mathcal{G}_4^{\mathrm{b}}$. We construct a reduction $\mathcal{B}_2$ playing the co-DDBDH game and will simulate the responses of queries made by adversary $\mathcal{A}_2$.

Initially, $\mathcal{B}_2$ obtains $(X, Y, Z, W) = (g_2^x, g_1^y, g_2^z, e(g_1, g_2)^w)$ from its co-DDBDH challenger. To simulate the encryptions $\mathsf{c_k}$ and public values for the guessed data subject, $\mathcal{B}_2$ does following.

- Randomly generates a symmetric key k

- Generates three random numbers $s, r_1, r_2 \xleftarrow{\$} \mathbb{Z}_q$
- Computes $\mathsf{prk} \leftarrow X^s, \mathsf{c}_1 \leftarrow Y^{r_1}, \mathsf{c}_2 \leftarrow Z^{\frac{sr_2}{r_1}}, \mathsf{c}_3 \leftarrow W^{r_2} \cdot \mathsf{k}$, and set $\mathsf{c}_\mathsf{k} \leftarrow (\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3)$.
- Computes public values as $\mathcal{G}_4$ specifies. Since the adversary is not allowed to corrupt both data collector and re-key involved in the same oracle, it will not notice whether the public values are generated from the ciphertext c_k or not.

Eventually, $\mathcal{A}_2$ submits a guess. If $\mathcal{A}_2$ guesses it is $\mathcal{G}_4^{\mathrm{b}}$ then $\mathcal{B}_2$ guesses 1 (real co-DDBDH tuple) to its co-DDBDH challenger. Otherwise, $\mathcal{B}_2$ guesses 0 (random co-DDBDH tuple) to its co-DDBDH challenger.

If $(X, Y, Z, W) = (g_2^x, g_1^y, g_2^z, e(g_1, g_2)^{\frac{yz}{x}})$ is a real co-DDBDH tuple, then the reduction simulated ciphertext $(\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3) = (g_1^{yr_1}, g_2^{\frac{(xs)(\frac{yzr_2}{x})}{(yr_1)}}, e(g_1, g_2)^{\frac{yzr_2}{x}} \cdot \mathsf{k})$ is valid ciphertext encrypted under public key $\mathsf{prk} = g_2^{xs}$. Hence, $\mathcal{B}_2$ perfectly simulates the environment of $\mathcal{A}_2$ in $\mathcal{G}_4^{\mathrm{b}}$. If (X, Y, Z, W) is a random co-DDBDH tuple, then c_k is simulated as a random ciphertext, $\mathcal{B}_2$ perfectly simulates the environment of $\mathcal{A}_2$ in $\mathcal{G}_5^{\mathrm{b}}$. Then

$$\left|\mathbf{Pr}[\mathcal{G}_5^{\mathrm{b}} = 1] - \mathbf{Pr}[\mathcal{G}_4^{\mathrm{b}} = 1]\right| \leq \mathbf{Adv}^{\mathsf{co\text{-}DDBDH}}_{(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T), \mathcal{B}_2}(\lambda).$$

Game 6

We change the challenge ciphertext term c_m to random ciphertexts. We have that the probability of any adversary notice this change is upper bounded by the IND-CPA advantage for SKE. Hence,

$$\left|\mathbf{Pr}[\mathcal{G}_6^{\mathrm{b}} = 1] - \mathbf{Pr}[\mathcal{G}_5^{\mathrm{b}} = 1]\right| \leq \mathbf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{SKE}, \mathcal{B}_3}(\lambda).$$

Now, all values relate to the M-RoRChall query are random. So

$$\left|\mathbf{Pr}[\mathcal{G}_6^1 = 1] - \mathbf{Pr}[\mathbf{Pr}[\mathcal{G}_6^0 = 1]\right| = 0.$$

□

A.4 Proof of Theorem 5

The proof is similar to the proof of Theorem 4. For $\mathrm{b} \in \{0, 1\}$,

Game 0

The first game $\mathcal{G}_0^{\mathrm{b}}$ is $\mathbf{Exp}^{\mathsf{Pub\text{-}RoR\text{-}b}}_{\mathsf{FDT}, \mathcal{A}}$, which is given in Def. 12, we have

$$\mathbf{Adv}^{\mathsf{Pub\text{-}RoR}}_{\mathsf{PFDT}, \mathcal{A}}(\lambda) = \left|\mathbf{Pr}[\mathcal{G}_0^1 = 1] - \mathbf{Pr}[\mathcal{G}_0^0 = 1]\right|.$$

Game 1

This game is the same as $\mathcal{G}_0^{\mathrm{b}}$, except for it guesses which session in the trading phase the adversary is going to issue the Pub-RoRChall query for and which data collector is involved in this session. The game returns a random bit for b' if the guess is wrong. The security lose is upper bounded by sl. That is

$$\left|\mathbf{Pr}[\mathcal{G}_1^1 = 1] - \mathbf{Pr}[\mathcal{G}_1^0 = 1]\right| = \left|\mathbf{Pr}[\mathcal{G}_0^1 = 1] - \mathbf{Pr}[\mathcal{G}_0^0 = 1]\right| \cdot \frac{1}{sl}$$

Game 2

We modify the game so that if the adversary issues a corruption query that would make the challenge session non-fresh the game returns a random bit for b'. Then

$$\mathbf{Pr}[\mathcal{G}_2^{\mathrm{b}} = 1] = \mathbf{Pr}[\mathcal{G}_1^{\mathrm{b}} = 1]$$

Game 3

In this game, we modify the actions of participants in the challenge session: all session oracles directly output 1 for each verification, and instead of computing the session message, the data collector simply output the one output by the data subject oracle. Then we have

$$\mathbf{Pr}[\mathcal{G}_3^{\mathrm{b}} = 1] = \mathbf{Pr}[\mathcal{G}_2^{\mathrm{b}} = 1].$$

Game 4

We replace the zero-knowledge proof that is submitted by data subject in the verification algorithm to be simulated proof. From the perfect special honest-verifier zero-knowledge property, we have

$$\mathbf{Pr}[\mathcal{G}_3^{\mathrm{b}} = 1] = \mathbf{Pr}[\mathcal{G}_4^{\mathrm{b}} = 1].$$

Game 5

We replace the statements that are submitted by data collector to be Pedersen commitments that are committed to random value, since the hiding property of the Pedersen commitment, the adversary cannot distinguish the differences. Therefore, in this game, the published values on the blockchain are all generated independently from the data subject's ciphertext, we have

$$\left|\mathbf{Pr}[\mathcal{G}_4^{\mathrm{b}} = 1] - \mathbf{Pr}[\mathcal{G}_5^{\mathrm{b}} = 1]\right| = 0.$$

$$\mathbf{Adv}^{\mathsf{Pub\text{-}RoR}}_{\mathsf{PFDT},\mathcal{A}}(\lambda) = \left|\mathbf{Pr}[\mathcal{G}_5^{1} = 1] - \mathbf{Pr}[\mathcal{G}_5^{0} = 1]\right| = 0.$$

A.5 Proof of Theorem 6

Game 0

The first game is the SigUF game given in Def. 13, we have

$$\mathbf{Adv}^{\mathsf{SigUF}}_{\mathsf{PFDT},\mathcal{A}}(\lambda) = \mathbf{Pr}[\mathcal{G}_0 = 1].$$

Game 1

This game is the same as $\mathcal{G}_0$, except for it guesses which data manager is involved in the challenge query SigUFChall. The game returns a random bit for b' if the guess is wrong. The security lose is upper bounded by m. That is

$$\mathbf{Pr}[\mathcal{G}_1 = 1] = \mathbf{Pr}[\mathcal{G}_0 = 1] \cdot \frac{1}{m}$$

Game 2

We modify the game so that if the adversary issues a corruption query on the guessed data manager the game returns 0. Then

$$\mathbf{Pr}[\mathcal{G}_2 = 1] = \mathbf{Pr}[\mathcal{G}_1 = 1]\Big|$$

Suppose that $\mathcal{A}$ is an adversary attempting to win $\mathcal{G}_2$. We construct a reduction $\mathcal{B}$ playing the EUF-CMA game and will simulate the responses of queries made by adversary $\mathcal{A}$. During the game $\mathcal{B}$ sends all signing computations of the guess data manager to the EUF-CMA challenger. Eventually, $\mathcal{A}$ submits a forgery, $\mathcal{B}$

forwards this forgery to the EUF-CMA challenger. If $\mathcal{A}$ provides a valid forgery, then $\mathcal{B}$ wins the EUF-CMA game. Then

$$\mathbf{Pr}[\mathcal{G}_2 = 1] = \mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\Sigma,\mathcal{B}}$$

A.6 Proof of Theorem 7

Game 0

The first game is the RkUF game given in Def. 13, we have

$$\mathbf{Adv}^{\mathsf{RkUF}}_{\mathsf{PFDT},\,\mathcal{A}}(\lambda) = \mathbf{Pr}[\mathcal{G}_0 = 1].$$

Game 1

This game is the same as $\mathcal{G}_0$, except for it guesses which session in the trading phase the adversary is going to issue the RkUFChall query for and which data subject is involved in this session. The game returns a random bit for b' if the guess is wrong. Then

$$\mathbf{Pr}[\mathcal{G}_1 = 1] = \mathbf{Pr}[\mathcal{G}_0 = 1] \cdot \frac{1}{sn}$$

Game 2

The same as $\mathcal{G}_2$ in the proof of Theorem 4.

Game 3

The same as $\mathcal{G}_3$ in the proof of Theorem 4.

Game 4

The same as $\mathcal{G}_5$ in the proof of Theorem 4. Then we have

$$\left|\mathbf{Pr}[\mathcal{G}_4 = 1] - \mathbf{Pr}[\mathcal{G}_1 = 1]\right| \leq \mathbf{Adv}^{\mathsf{co\text{-}DDBDH}}_{(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T)}(\lambda).$$

In the end, we prove that if there exists an adversary $\mathcal{A}$ winning $\mathcal{G}_4$ with probability p, then there exists a reduction $\mathcal{B}_2$ breaking co-DCDH with probability p. We construct a reduction $\mathcal{B}_2$ that takes input $(X, Y) = (g_1^x, g_2^y)$, the reduction simulates ciphertexts as $\mathsf{c_k} = (X^{r_1}, Y^{r_2}, g_T^{r_3})$ for random values $r_1, r_2, r_3 \xleftarrow{\$} \mathbb{Z}_q$. Suppose the adversary ask for challenge query in a session with $\mathsf{rk} = (\mathsf{rk}_1, \mathsf{rk}_2)$. The reduction returns $(\mathsf{rk}_1 \cdot \mathsf{rk}_2^{1/\mathsf{sk_{DC}}})^{\frac{r_2}{r_1}}$ to its co-DCDH challenger. If the adversary $\mathcal{A}$ creates a valid forgery for rk, then $(\mathsf{rk}_1 \cdot \mathsf{rk}_2^{1/\mathsf{sk_{DC}}})^{\frac{r_2}{r_1}} = g_1^{x/y}$. Which means $\mathcal{B}_2$ outputs a valid co-DCDH output to its co-DCDH challenger.