Practical Exchange for Unique Digital Goods

Oğuzhan Ersoy*, Ziya Alper Genç†, Zekeriya Erkin*, Mauro Conti‡*
*Cyber Security Group, Delft University of Technology, Delft, Netherlands
{o.ersoy,z.erkin}@tudelft.nl
†SnT, University of Luxembourg, Luxembourg, Luxembourg
ziya.genc@uni.lu
‡Department of Mathematics, University of Padua, Padua, Italy
conti@math.unipd.it

***Abstract*—Smart contracts can be used for the fair exchange of digital goods. A smart contract can escrow the exchange where the receiver deposits the payment, and the sender claims it by providing the goods. In the case of misbehavior, the parties provide proof on whether the received goods match the pre-agreed description or not. In general, the description is assumed to be the hash of the goods, and it is publicly known. However, without trusting the description provided by the sender, this assumption is not plausible for the scenarios where the goods are uniquely created for a specific receiver. To overcome the trust issue, sampling-based exchange protocols have been introduced where the parties use a sample of the goods as the description. Yet, the existing sampling-based proposals suffer from high on- and off-chain computational and storage costs.**

In this paper, we present FairDEx: an efficient sampling-based protocol that is suitable for the exchange of unique goods. Our description protocol allows us to achieve low on- and off-chain costs, which are independent of the size of the goods. The off-chain part of the protocol only utilizes highly efficient algorithms, namely hashing and symmetric key encryption. To illustrate the feasibility of FairDEx, we evaluate a research prototype on the Ethereum test network. Our results show that the cost of running FairDEx is around 0.6M gas for reasonably large sample sets, which is 30% cheaper than the state-of-the-art Ethereum-based proposals.

***Index Terms*—Fair Exchange, Blockchain, Smart Contracts**

I. Introduction

Imagine that you are going to buy digital goods exclusively made for you. You accept to pay if and only if you receive the exact goods you requested. To convince you, the seller displays the *hash* of the goods that you can only verify when you receive it. Would you trust that the hash is calculated accordingly? In this paper, we address the fair exchange problem of such scenarios where there is a designated receiver of the goods that do not have public information (e.g. hash) accepted or verified by third parties.

In fair exchange, a receiver is willing to pay a price for the digital goods provided by a sender where either both parties receive their expected output or none do. it has been known that the fair exchange is not possible without a Trusted Third Party (TTP), which can be very costly or might not be available at all [1]. With the recent developments in the blockchain technology, it is possible to achieve a practical and cost friendly TTP.

A blockchain can escrow a fair exchange where the receiver deposits the payment for the goods and the sender claims the payment by providing the correct goods, and if the sender misbehaves, the receiver gets the deposit back. Here, the correctness of the goods is defined over the description of goods that is accepted by both parties. This process, also named as the claim-or-refund, can be implemented with smart contacts. If there is a description of goods agreed by both parties, the smart contract enables them to create a conditional payment that is spendable only by satisfying an input matching with the description.

There have been several works [2], [3], [4], [5], [6], [7], [8] using a blockchain as a TTP if there exists a description of goods that is accepted by the receiver. The commonly used description, hash of the goods, does not give any information about the goods, which is preferred from the security perspective of the sender. However, from the receiver side, it requires the trust to either (i) the sender, (ii) other receivers of the same goods, or (iii) a TTP that receives the goods and description and validates its correctness [9]. Note that a validating TTP would be much more expensive, if even exists, than an escrowing TTP such as blockchain. It may require special expertise and knowledge to validate the goods that might be possessed only by a dedicated receiver of the goods.

An example scenario in real-life would be crowdsourcing services like Amazon's Mechanical Turk (MTurk). In the machine learning community, researchers use MTurk for manual data annotation or labeling for applications like behavior signal processing, computer vision and so on [10], [11], [12]. Here, annotators need to be ensured that they will receive a payment for their work, and researchers need to be ensured on the quality and correctness of these annotations, which is a challenging problem [12]. Note that the researchers (receiver) cannot trust the hash of annotations given by the annotators (sender) and use Amazon (TTP) for only escrowing purposes. Thus, the receiver either requests (i) the TTP to validate the correctness of the goods (if a TTP has the expertise), and the description is correctly calculated, or (ii) a sample of the goods from the sender as a commitment that will be used to generate the description. It is also possible to extend the validation of goods into a consensus of third parties where the goods are divided into several subgoods and each of them are independently validated by several third parties [13].

Recently, alternative exchange protocols have been introduced that are suitable for scenarios where the description of the goods is not available [14], [15], [16], [17], [18]. They replace the description by a sample of the goods which is jointly selected. These protocols are suitable for digital goods that can be split into individually verifiable sub-goods from which the description is generated. For example, continuing to the data annotation use case, each data point would be a sub-good.

The sampling-based exchange protocols work as follows. First, sender shares the encryption of the goods as a commitment of the goods. Then, sender and receiver jointly select a small subset of the goods as a sample set, which are decrypted by the sender for validation. Later on, the receiver makes the payment conditional to the decryption key(s) of the goods, which matches with the one(s) used to decrypt the sample set. It can be seen that the decryption key of the sample takes the role of description of goods. However, the existing conditional payment mechanisms are based on either (i) public key (or hybrid) encryption schemes which are not efficient in off-chain computation or (ii) symmetric key encryption schemes that require on-chain publication of each key of the sub-goods.

In this paper, we present FairDEx that achieves the best of two approaches of sampling-based exchange protocols: it is based on symmetric key encryption which requires low off-chain computation and storage cost, and it requires publication of only a single (master) key on the blockchain. FairDEx can be defined with two phases, namely initialization and claim-and-fund. The initialization phase is executed off-chain where parties agree on the description using a sample of the goods. In the claim-and-refund phase, blockchain escrows the exchange regarding the description.

In the initialization phase, the sender encrypts each sub-good with a different subkey and shares the encrypted data with the receiver. Here, the subkeys are generated from a master key using a simple key generation algorithm. Then, sender and receiver jointly choose a subset of sub-good as the sample, and the receiver provides the subkeys of them to the receiver. Once receiver approves the sub-goods, the description is generated from the sample subkeys.

In the claim-and-refund phase, the receiver creates the conditional payment with the condition on the agreed description and publishes the payment on the blockchain. Later, as the sender reveals the master key matching with description, the sender obtains the payment, and the receiver obtains the goods. If the master key does not match with the description, the receiver can create a proof and reclaim the payment. The proof simply shows that the master key does not produce at least a subkey that is revealed in the initialization phase.

The advantages of FairDEx are summarized as follows:

- *Description generation*: FairDEx offers a jointly generated description that removes the need for the implicit trust in the description given by the sender. This is essential in the scenarios where the description is not public knowledge nor validated by a TTP.
- *Practical implementation*: We implemented FairDEx as an Ethereum smart contract and demonstrate the efficiency of our protocol in terms of gas. Finally, we present a comparison of execution costs between similar works. The source code of FairDEx smart contract and client applications can be found at https://github.com/ziyagenc/fairdex.
- *Low off-chain cost*: FairDEx has a simple architecture and utilizes only efficient cryptographic primitives, which are hashing and symmetric-key encryption algorithms. This ensures low off-chain computational complexity for both the sender and receiver and enables FairDEx to run on resource-constrained devices like smartphones.
- *Low on-chain cost*: In the optimistic case where parties are both honest, only the master key is revealed to claim the payment. In the pessimistic case, the proof-of-misbehavior is demonstrated by using only the subkeys. Our proof is independent of the size of the sub-goods and logarithmically dependent on the sample set, and the on-chain cost is around 0.6M gas for a set of 10K samples.

II. State-of-the-art Fair Exchange Protocols

The pioneering work of Asokan et al. [19] defines a fair exchange protocol between a sender $\mathcal{S}$ and a receiver $\mathcal{R}$ with a description function $\mathsf{Desc}(\cdot)$. $\mathcal{S}$ owns an item $\mathsf{i}_{\mathcal{S}}$ with the description $\mathsf{desc}_{\mathcal{S}} := \mathsf{Desc}(\mathsf{i}_{\mathcal{S}})$ and $\mathcal{R}$ owns an item $\mathsf{i}_{\mathcal{R}}$ with the description $\mathsf{desc}_{\mathcal{R}} := \mathsf{Desc}(\mathsf{i}_{\mathcal{R}})$. The goal of the protocol is to exchange the items between parties.

A fair exchange protocol can be defined in two phases: *initialization* and *claim-and-fund*. In the initialization phase, parties agree on the exchange conditions such as the description of the item and trusted third party (if necessary). After this phase, it is assumed that $\mathcal{S}$ has $(\mathsf{i}_{\mathcal{S}}, \mathsf{desc}_{\mathcal{R}})$ and $\mathcal{R}$ has $(\mathsf{i}_{\mathcal{R}}, \mathsf{desc}_{\mathcal{S}})$. In the second phase, claim-and-fund, parties execute the exchange of items where $\mathcal{S}$ obtains $\mathsf{o}_{\mathcal{S}}$ and $\mathcal{R}$ obtains $\mathsf{o}_{\mathcal{R}}$. Parties check the correctness of the obtained item using the description function. In the case of a dispute, the description allows parties to provide proof of misbehavior.

There are four security properties that a fair exchange protocol should satisfy [19], [20], [6], [1], namely *correctness*, *timeliness*, *sender fairness* and *receiver fairness*:

- *Correctness*: If no party misbehaves, upon termination, $\mathcal{R}$ obtains $\mathsf{o}_{\mathcal{R}}$ such that $\mathsf{Desc}(\mathsf{o}_{\mathcal{R}}) = \mathsf{desc}_{\mathcal{S}}$ and $\mathcal{S}$ obtains $\mathsf{o}_{\mathcal{S}}$ such that $\mathsf{Desc}(\mathsf{o}_{\mathcal{S}}) = \mathsf{desc}_{\mathcal{R}}$.
- *Timeliness*: Every honest party eventually terminates.

- *Receiver Fairness*: If honest $\mathcal{R}$ does not obtain any information about $\mathsf{i}_{\mathcal{S}}$ other than $\mathsf{desc}_{\mathcal{S}}$, then $\mathcal{S}$ cannot obtain any information about $\mathsf{i}_{\mathcal{R}}$ other than $\mathsf{desc}_{\mathcal{R}}$.
- *Sender Fairness*: If honest $\mathcal{S}$ does not obtain any information about $\mathsf{i}_{\mathcal{R}}$ other than $\mathsf{desc}_{\mathcal{R}}$, then $\mathcal{R}$ cannot obtain any information about $\mathsf{i}_{\mathcal{S}}$ other than $\mathsf{desc}_{\mathcal{S}}$.

A relaxed notion of fairness is defined as *probabilistic fairness* where the chance of violating fairness is restricted with an arbitrarily low probability [21], [22], [23].

It has been shown that fairness cannot be achieved without a trusted third party [1]. In a minimal setting of using TTP, named as optimistic fair exchange [24], a TTP needs to be involved to solve the disputes between parties. Here, parties prove their claims to TTP by comparing the obtained output and the initially agreed description. Recent proposals use a blockchain as a TTP since it is practical and cost friendly. Now, we first present the exchange protocols assuming the description is known in advance. Later, we explain the protocols where the description is not available, which is also the scenario assumed in this work.

A. Exchange protocols with known description

The recent exchange protocols use a blockchain as TTP where the description match is automated via conditional payments. To deploy and execute the conditional payment, parties need to pay a fee to the miners of blockchain for each on-chain operation and storage used. Therefore, a naive way of publishing the whole of goods on-chain to verify the description would require high fees. As the fee would be proportional to the size of the goods, this solution would not be feasible for the large-sized goods. To minimize the cost, the state-of-the-art fair exchange protocols use either Zero-Knowledge (ZK) proofs or proof-of-misbehavior structure.

ZK proofs are used to check the condition on payment, also named zero-knowledge contingent payments (ZKCP) [25], [26], [27]. The process of a ZKCP can be summarized as follows: the sender shares the encrypted goods with the receiver and provides ZK proof on the commitment on the key and receives the payment while revealing the commitment. The on-chain cost of a ZKCP would be relatively low because only the commitment is checked on-chain, which is usually a hash confirmation. In proof-of-misbehavior based protocols [6], [28], [29], [30], the payment condition is checked only in the case of a dispute. By moving the condition check to only disputes, in an optimistic case where parties are honest, they also achieve low on-chain cost.

B. Exchange protocols without known description

One approach to overcome the lack of description is having a third-party consensus mechanism that validates the correctness of the goods [13]. The idea is splitting the goods into sub-goods and asking the third-party validators to vote for the correctness of each sub-good. Then, using a consensus on the votes, the goods are validated or not. However, having a third-party validator can be costly or may not be available in scenarios where the goods are unique or customized for a specific receiver.

As an alternative, *sampling-based* exchange protocols [14], [15], [16], [17], [18] have been introduced where the sender and receiver jointly generate a description from a sample of the goods. This can be seen as a control mechanism where the sender reveals a sample of goods for verification. However, as the description is revealed to the receiver, it should be enough to convince the receiver, yet it should be infeasible to learn more about the goods. The joint description generation can be seen as a gradual release where the release is done only once. In a gradual release [31], [32], [33] system, the exchange items are divided into small sub-items and, at each step, parties exchange a sub-item so that the advantage of a party would be negligibly small. Applying the release mechanism on the entire goods would have high communication complexity since the round complexity is linear to the number of sub-items.

In [16], the authors present a basic sampling-based exchange protocol on Bitcoin. The protocol works as follows: First, the sender encrypts each sub-good with a different symmetric key. Then, she commits to goods and encryption keys by sharing the encryption of the goods and the hash of each key with the receiver. Then, the parties sample a subset of sub-goods, for which the sender provides the corresponding encryption keys. After the receiver verifies their correctness, he creates a conditional payment which can be claimed only by providing the preimages of each remaining key commitment. The protocol is simple and works on Bitcoin since payments with hash conditions are available in Bitcoin scripts. However, it is quite inefficient since the payment condition is about the size of sub-goods. In other words, the transaction size increases linearly with the number of sub-goods.

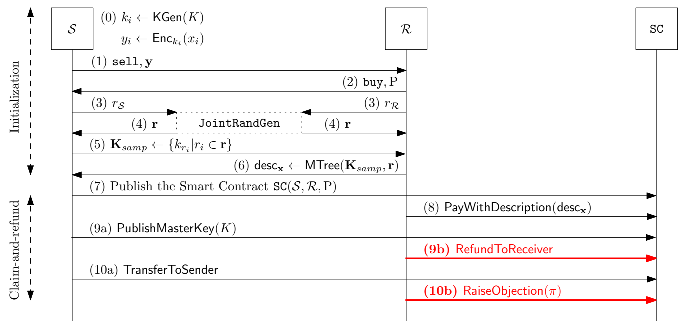
In [14], the authors propose a novel sampling-based exchange protocol that requires the publication of a single key on the blockchain. The protocol exploits the fact that ECDSA signature scheme [34] is a double authentication preventing signature (DAPS) [35] where if two different messages are signed with the same randomness, then the signing key can be easily calculated from the signatures. The idea is that the sender encrypts each sub-good with the same public key. Then, some of the sub-goods are opened by the sender as a sample where the receiver checks the validity of the sub-goods and if they are all encrypted with the same public key. After the validation, the sender signs a message with a chosen randomness and sends it to the receiver. Then, the receiver creates a payment transaction that requires the sender to sign with the same public key and the chosen randomness to receive the payment. Thus, once the payment is claimed, the receiver can use two signatures to compute the private key and decrypt the rest of the sub-goods. However, since public-key encryption is not efficient compared to symmetric ones, they suggest a hybrid model where each sub-good is encrypted with a different symmetric key, and these symmetric keys are encrypted with the public key and stored with the encrypted goods.

In [15], a racing attack is found in the protocol of [14] where a malicious receiver can try to reclaim the payment. The authors of [15] proposed an improved and secure one that requires an additional signature of the sender in the payment transaction. Moreover, their construction has an additional transaction to provide instant finality payment. In [18], the authors propose an Ethereum version of a similar exchange protocol that utilizes the DAPS idea. Finally, in [17], another Ethereum-based solution is proposed, which replaces DAPS-like secret key recovery with the explicit publication of the secret key and validating its correctness with a decryption of sampled sub-goods.

III. Our Protocol

In this section, we present our exchange protocol FairDEx. We assume a scenario where the goods $\mathbf{x}$, consisting of $|\mathbf{x}|$ sub-goods, i.e., $\mathbf{x} := \{x_i\}_{i=1}^{|\mathbf{x}|}$, owned by a sender $\mathcal{S}$ is exchanged for a price P paid by a unique receiver $\mathcal{R}$. We use a smart contract-based escrow mechanism, which solves disagreements between parties. More specifically, FairDEx is based on a conditional payment of price P such that the payment is sent to $\mathcal{S}$ if $\mathcal{R}$ obtains the goods $\mathbf{x}$. Here, the condition is defined regarding the description of goods $\mathsf{desc}_{\mathbf{x}}$, which is agreed by both parties, and the conditional payment is created by smart contracts. As shown in Fig. 1, we define FairDEx in two phases: (i) *initialization* where the description is created and (ii) *claim-and-refund* where the exchange and possibly disagreements are handled. Before explaining our protocol, we present the communication and security model and notations.

We assume there is an authentic and confidential channel between each entity. Sender and receiver use their channels for the off-chain interactions. Similarly, parties use the channels with the blockchain to send and receive on-chain interactions. We assume the synchronous network model where each message sent between entities are received at the beginning of the next round. A round defines the time unit and all entities have access to the global time. Moreover,



$\mathcal{S}$
$\mathcal{R}$
SC
(0) $k_i \leftarrow \mathsf{KGen}(K)$
$y_i \leftarrow \mathsf{Enc}_{k_i}(x_i)$
(1) sell, $\mathbf{y}$
(2) buy, P
(3) $r_{\mathcal{S}}$
(3) $r_{\mathcal{R}}$
(4) $\mathbf{r}$
JointRandGen
(4) $\mathbf{r}$
(5) $\mathbf{K}_{samp} \leftarrow \{k_{r_i} | r_i \in \mathbf{r}\}$
(6) $\mathsf{desc}_{\mathbf{x}} \leftarrow \mathsf{MTree}(\mathbf{K}_{samp}, \mathbf{r})$
(7) Publish the Smart Contract SC($\mathcal{S}, \mathcal{R}$, P)
(8) PayWithDescription($\mathsf{desc}_{\mathbf{x}}$)
(9a) PublishMasterKey(K)
(9b) RefundToReceiver
(10a) TransferToSender
(10b) RaiseObjection(π)
Initialization
Claim-and-refund

Figure 1: Illustration of our protocol FairDEx. Honest and malicious executions of the protocol are separated with (9a)-(9b) and (10a)-(10b). All messages are sent in secure and authenticated channels.

we assume that blockchain executes each valid request received from the parties within an upper bounded number of rounds. For simplicity, we omit this upper bound and assume that honest parties take that into consideration.

There is no trust assumption between the sender and receiver, and both parties agree on blockchain being the trusted third party (TTP). We assume that a PPT(probabilistic polynomial-time) adversary can corrupt parties at the beginning of the protocol. A corrupt party follows the instructions given by the adversary, whereas an honest party follows the instructions of the protocol. Regarding the security of the blockchain, we assume that no adversary can intervene in the process of the blockchain, i.e., preventing the execution of a published smart contract.

In our protocol, we utilize two cryptographic primitives: a cryptographic hash function and a symmetric encryption algorithm. We use the random oracle model (ROM) to model the hash function [36]. In ROM, hash functions return a uniformly selected random value $h \leftarrow \{0,1\}^{\mu}$ for each newly queried input string and return the same value if the same query is answered before. We assume the encryption algorithm Enc satisfies indistinguishability under chosen-plaintext attack (IND-CPA) security [36].

Regarding notations, we use $\|$ for concatenation of two objects, $:=$ for defining an object, and $\leftarrow$ for assigning a value on an object. We use bold letters to represent a set of elements such as $\mathbf{a}$ and each element is shown with an index of the same letter, e.g., a_i. $\mathbf{a} := \{a_i\}_{i=1}^{N}$ denotes a set of items a_i's for $i = 1$ to N and $|\mathbf{a}|$ refers to the size of the set, i.e., $|\mathbf{a}| = N$. Now, we explain the details of the initialization and claim-and-refund phases of our protocol.

A. Initialization Phase

In this phase, $\mathcal{S}$ and $\mathcal{R}$ agree on the exchange conditions and generate the description $\mathsf{desc}_{\mathbf{x}}$. First, parties agree on the payment price of P, sample amount s, timelock condition T_{lock} and objection time T_{obj}. T_{lock} and T_{obj}, as explained later, are crucial for the timeliness property of the protocol. Second, $\mathcal{S}$ shares the encryption of goods $\mathbf{y}$ with $\mathcal{R}$. Finally, parties jointly generate the description $\mathsf{desc}_{\mathbf{x}}$ using the encryption keys.

The subkey generation and encryption algorithms are given in Algorithm 1. $\mathcal{S}$ chooses a random master key K regarding the security parameter k and generates $|\mathbf{x}|$ subkeys using $\mathsf{SUBKGEN}(K, |\mathbf{x}|)$. Each subkey k_i is computed by querying hash function H on input K concatenated with index i. Then, $\mathcal{S}$ encrypts the goods $\mathbf{x}$ with the encryption algorithm $\mathsf{ENCRYPT}$ where each sub-good x_i encrypted with Enc function using the corresponding subkey k_i. Then, $\mathcal{S}$ shares the encryption of goods $\mathbf{y}$ with $\mathcal{R}$.

After $\mathcal{S}$ shares the encryption of goods $\mathbf{y}$ with $\mathcal{R}$, parties generate the description $\mathsf{desc}_{\mathbf{x}}$. The description is used as a witness to ensure that $\mathcal{S}$ will provide the expected goods $\mathbf{x}$. More precisely, parties, first, agree on a description $\mathsf{desc}_{\mathbf{x}}$, then they make the exchange of $\mathbf{x}$ and payment conditioned on that. Then, in the case of a dispute, $\mathsf{desc}_{\mathbf{x}}$

Algorithm 1 Subkey generation and encryption.

```
SUBKGEN$(K, |\mathbf{x}|)$
    $\mathbf{K} \leftarrow \emptyset$
    for $i \in (1, |\mathbf{x}|)$ do
        $k_i \leftarrow \mathsf{H}(K \| i)$
        $\mathbf{K} \leftarrow \mathbf{K} \cup \{k_i\}$
    return $\mathbf{K}$

ENCRYPT$(\mathbf{K}, \mathbf{x})$
    $\mathbf{y} \leftarrow \emptyset$
    for $i \in (1, |\mathbf{x}|)$ do
        $y_i \leftarrow \mathsf{Enc}_{k_i}(x_i)$
        $\mathbf{y} \leftarrow \mathbf{y} \cup \{y_i\}$
    return $\mathbf{y}$
```

allows parties to claim whether the received goods are the same as the expected goods or not. This is checked by running the description algorithm on the received goods. Here, we define the description algorithm.

Our description algorithm can be seen as a random sampling of subkeys. Initially, using Rand function, parties jointly generate s distinct indexes, i.e., $\mathbf{r} := \{r_i\}_{i=1}^{s}$ where $r_i \in (1, |\mathbf{x}|)$ and $r_i \neq r_j$ for $i \neq j$. These indexes define which subkeys are part of $\mathsf{desc}_\mathbf{x}$. $\mathcal{S}$ shares the corresponding subkeys $\mathbf{K}_{samp} := \{k_{r_i}\}_{i=1}^{s}$ with $\mathcal{R}$. Then, $\mathcal{R}$ validates the correctness of the subkeys $\mathbf{K}_{samp}$ by decrypting the corresponding indexes of $\mathbf{y}$, presented by the Verify function. Here, $\mathcal{R}$ checks each sub-good $x'_{r_i} := \mathsf{Dec}_{k_{r_i}}(y_{r_i})$ for all $r_i \in \mathbf{r}$, if the validation fails for any of the sub-goods, $\mathcal{R}$ aborts the protocol. Otherwise, $\mathcal{R}$ generates $\mathsf{desc}_\mathbf{x}$ using the Merkle tree construction [37] where $\mathsf{desc}_\mathbf{x}$ is the root of the tree and $\mathbf{K}_{samp}$ are the leaves. Moreover, $\mathcal{R}$ stores $\mathbf{K}_{samp}$ and $\mathbf{r}$ off-chain, which would be necessary in the case of dispute. The description algorithm, DESC, is presented in Algorithm 2 and the details of Rand and Verify functions are given in Appendix A.

Algorithm 2 DESC$(\mathbf{K}, \mathbf{y}, s)$

```
$\mathbf{r} \leftarrow \mathsf{Rand}_{\mathcal{S},\mathcal{R}}(|\mathbf{y}|, s)$
$\mathbf{K}_{samp} := \{k_{r_i} | r_i \in \mathbf{r}\} \leftarrow \mathsf{Sample}_{\mathcal{S}}(\mathbf{K}, \mathbf{r})$
if $\mathsf{Verify}_{\mathcal{R}}(\mathbf{K}_{samp}, \mathbf{y})$
    $\mathsf{desc}_\mathbf{x} \leftarrow \mathsf{MTree}(k_{r_1}, r_1, \ldots, k_{r_s}, r_s)$
return $(\mathsf{desc}_\mathbf{x}, \mathbf{K}_{samp}, \mathbf{r})$
```

B. Claim-and-refund Phase

In the claim-and-refund phase, the exchange of goods and payment is executed with on-chain transactions. According to the agreed conditions in the previous phase, the smart contract is published on the blockchain by the sender. The pseudo-code of the contract is given in Fig. 2.

$\mathcal{R}$ first publishes the payment of price P conditioned to description $\mathsf{desc}_\mathbf{x}$ by using the function PayWithDescription. When the function is invoked, the timeout value T_{lock} is assigned on the payment, which prevents $\mathcal{S}$ to claim the payment after time T_{lock}. More precisely, if the key is not published within T_{lock} rounds, then $\mathcal{R}$ can claim a refund of the payment by calling the RefundToReceiver function. Once the payment is published, $\mathcal{S}$ checks the correctness of the description. If the description condition on the payment is not correct or the current time t is passed the time condition, i.e., $t > T_{lock}$, then $\mathcal{S}$ aborts the protocol. Otherwise, $\mathcal{S}$ publishes the master key $\hat{K}$ on-chain using PublishMasterKey. After the publication of $\hat{K}$, $\mathcal{R}$ is allowed to object on the received key in T_{obj} rounds.

```
contract FairDEx {
  // Init. parameters: price, T_lock, T_obj, Sender and Receiver addresses
  function PayWithDescription( _description)
  only(receiver, value == price) {  // called by receiver with price
    description = _description; // Assign description
    timeout_lock = now + T_lock;// Time condition to publish master key
  }

  function RefundToReceiver()
  only(receiver, now > timeout_lock) {// called by receiver after timeout_lock
    selfdestruct(buyer);  // Transfer payment to receiver
  }

  function PublishMasterKey(_masterKey)
  only(sender, now < timeout_lock) {  // called by sender before timeout_lock
    masterKey = _masterKey;  // Assign the master key
    timeout_obj = now + T_obj;  // Time condition to object on master key
  }

  function RaiseObjection(r_i, k_i, mPath)
  only(receiver, now < timeout_obj) {  // called by receiver before timeout_obj
    if (H(masterKey, r_i) != k_i) {  // Check 1: the subkey is NOT correct
      root = MProof(r_i, k_i, mPath) // Build Merkle Tree proof for description
      if (root == description) // Check 2: the root equals to the description
      selfdestruct(buyer); // Transfer payment to receiver
    }
  }

  function TransferToSender()
  only(sender, now > timeout_obj) {  // called by sender after timeout_obj
    selfdestruct(seller);  // Transfer payment to sender
  }
}
```

Figure 2: Pseudo-code of FAIRDEX smart contract.

If no party objects, after T_{obj} rounds, $\mathcal{S}$ receives the payment by calling TransferToSender function. In the case of an objection where $\mathcal{R}$ claims that the published key $\hat{K}$ does not match with $\mathsf{desc}_\mathbf{x}$, $\mathcal{R}$ creates a proof using EVIDENCE algorithm given in Algorithm 3. The proof simply shows that $\hat{K}$ does not generate the sampled subkeys $\mathbf{K}_{samp}$. Specifically, EVIDENCE returns a proof π which consists of (i) a pair of r_i and k_{r_i} for which the obtained key $\hat{K}$ is not generating the same subkey value (ii) the Merkle tree path π_{path} that shows the pair is part of the description $\mathsf{desc}_\mathbf{x}$. With a valid proof π, $\mathcal{R}$ can re-claim the payment by invoking RaiseObjection function.

Algorithm 3 EVIDENCE$(\hat{K}, \mathsf{desc}_\mathbf{x}, \mathbf{K}_{samp}, \mathbf{r})$

```
$\pi \leftarrow \perp$
Parse: $\{r_i\}_{i=1}^{s} \leftarrow \mathbf{r}, \quad \{k_{r_i}\}_{i=1}^{s} \leftarrow \mathbf{K}_{samp}$
$\hat{\mathbf{K}} \leftarrow \mathsf{SUBKGEN}(\hat{K}, |\mathbf{x}|)$
for $i \in (1, s)$
    if $k_{r_i} \neq \hat{k}_{r_i}$
        $\pi_{\mathsf{path}} \leftarrow \mathsf{MTreePath}(k_{r_i}, r_i, \mathsf{desc}_\mathbf{x})$
        $\pi \leftarrow (r_i, k_{r_i}, \pi_{\mathsf{path}})$
return $\pi$
```

C. Discussion on Design Rationale

Here, we discuss the design choices of the description and subkey generation algorithm regarding the on-chain cost-efficiency concerns.

The on-chain cost of our protocol heavily depends on the description algorithm. A straightforward way of generating the description would be a sampling of goods regarding the chosen index set $\mathbf{r}$, i.e., $\{x_{r_i}\}_{i=1}^{s}$. Then, in the case of a dispute, the proof size would be at least in the size of x_i. In other words, the storage and computational on-chain cost would be proportional to the size of x_i. To minimize the on-chain cost for a possibly large set of goods $\mathbf{x}$, FAIRDEX uses the encryption subkeys $\mathbf{K}$ to define the description $\mathsf{desc}_{\mathbf{x}}$. Here, the receiver $\mathcal{R}$ validates $\mathsf{desc}_{\mathbf{x}}$ by decrypting the corresponding sub-good x_i for each sampled subkey k_i. Thereby, in case of a dispute, the blockchain does not have to operate on the goods, instead, $\mathsf{desc}_{\mathbf{x}}$ and SUBKGEN algorithms are sufficient. Thus, the on-chain computation cost is also independent of the size of the goods.

We utilize Merkle tree construction to create the description from $\mathbf{K}_{samp}$. The storage cost of $\mathsf{desc}_{\mathbf{x}}$ would be only the root of the tree. In case of a dispute, the computational cost of the proof is a logarithmic number of hash operations w.r.t. the size of $\mathbf{K}_{samp}$. To minimize the cost, $\mathcal{R}$ uses only one incorrect subkey generated from the SUBKGEN as proof. In other words, to prove the misbehavior, instead of showing all subkeys of indices of $\mathsf{desc}_{\mathbf{x}}$, $\mathcal{R}$ can use a single subkey k_{r_i} where $r_i \in \mathbf{r}$. This is because, for the correct master key, all of the subkeys are supposed to be correctly computed with the SUBKGEN algorithm.

Another crucial component of the on-chain cost of our protocol is the SUBKGEN algorithm. Firstly, it is important to note that the protocol does not require a subkey generation algorithm. A naive solution would be to encrypt each sub-good with a randomly generated subkey. However, then, the sender needs to reveal all of the subkeys on-chain to receive the payment. In our protocol, the receiver only publishes the master key because all the subkeys can be generated from the master key. Secondly, to minimize the storage and computational on-chain cost, SUBKGEN is supposed to have a small code size and it should require low computation cost concerning the on-chain computational cost of each arithmetic operation. This is because, in the case of a disagreement, the proof-of-misbehavior generated by EVIDENCE algorithm is checked by running the subkey generation algorithm for some specific subkeys. We achieve cost efficiency by using a simple subkey generation algorithm, SUBKGEN, requiring one concatenation with an index and one hash operation.

D. Security Analysis

In this section, we show that our protocol satisfies the security properties of a fair exchange protocol given in Section II: correctness, timeliness, receiver fairness and sender fairness. Because of the limited space, we briefly argue the correctness and timeliness properties. The *correctness* property implies that if all parties honestly follow the protocol, then they will obtain their expected output. It can be seen that if $\mathcal{S}$ honestly generates the goods and description, honest $\mathcal{R}$ would pay to the description which can be claimed by $\mathcal{S}$ once the master key is revealed.

Lemma 1. FAIRDEX *satisfies the correctness property.*

The second property is *timeliness* meaning that for an honest user the protocol eventually terminates. The time conditions in the smart contract of FAIRDEX ensures the timeliness property. Specifically, if $\mathcal{R}$ publishes the payment at time t, then at time $t + T_{lock}$, either the master key is published or the payment is sent back to her account. Also, if the published key is faulty, then $\mathcal{R}$ can claim the payment back in T_{obj} rounds. Therefore, at the latest, on time $t + T_{lock} + T_{obj}$, the protocol is terminated for $\mathcal{R}$. For $\mathcal{S}$, since the key is not revealed until the payment is published, he can abort at any time before that. Once the payment is on-chain and he publishes the key at time t, then he will receive the payment at most in time $t + T_{obj}$.

Lemma 2. FAIRDEX *satisfies the timeliness property.*

Now, we show that FAIRDEX satisfies the fairness properties: *receiver fairness* in Lemma 3 and *sender fairness* in Lemma 4. Here, it is important to note that the sample of goods does not violate the sender fairness given in Section II because it is part of the description. However, using a sample of goods as a description has two drawbacks: (i) sender is not paid for the sample and therefore she might be susceptible to attack of repetitive execution of sampling; and (ii) the fairness of the receiver is probabilistic and depends on the size of the sample set. The former repetitive attack is not possible for our exchange scenario since we assume that there is only one (unique) receiver of the exclusive goods. For the latter, we show that the probability of violating fairness can be made negligibly small by increasing the number of samples.

Lemma 3. FAIRDEX *satisfies the probabilistic receiver fairness property. No adversarial sender $\mathcal{A}$ can violate the receiver fairness with probability more than $(1-p)^s$ where $\mathcal{A}$ does not follow the protocol for $p \cdot |\mathbf{x}|$ of the sub-goods.*

Proof Sketch. Receiver fairness implies that $\mathcal{S}$ does not receive the payment of price P unless $\mathcal{S}$ provides $\mathbf{x}$ matching with the description $\mathsf{desc}_{\mathbf{x}}$. In our case, $\mathsf{desc}_{\mathbf{x}}$ is generated over the master key K, and $\mathcal{S}$ shares the encrypted goods $\mathbf{y}$ with $\mathcal{R}$. Thus, the receiver fairness relies on $\mathcal{S}$ sharing the correct $\mathbf{y}$ and publishing the correct K such that $\mathcal{R}$ can decrypt and obtain $\mathbf{x}$. In other words, an adversary $\mathcal{A}$ violating receiver fairness implies that $\mathcal{A}$ obtains the payment and $\mathcal{R}$ does not obtain (at least some of parts of) $\mathbf{x}$. Because of the conditional payment in FAIRDEX, violation of fairness can only occur if $\mathcal{A}$ publish a master key that satisfies the description yet it does not provide all the correct subkeys for $\mathcal{R}$. Particularly, the maliciously generated sub-goods or their encryption keys are not part of the description. Otherwise, $\mathcal{R}$ would abort the protocol before sending the payment.

We now investigate the attack scenario where $\mathcal{A}$ does not follow the protocol for $p \cdot |\mathbf{x}|$ of the sub-goods. Here, p is the probability of malicious execution for each sub-good. Note that since the sample set is randomly chosen, the selection of the $p \cdot |\mathbf{x}|$ sub-goods does not affect the success probability of the attack. The success probability of the adversary can be seen as the probability of randomly selecting s non-faulty items from a set size of $|\mathbf{x}|$ where $p \cdot |\mathbf{x}|$ items are faulty. This is because $\mathcal{A}$ needs to share $\mathbf{y}$ before the description is created, and, in the description algorithm, parties jointly select a subset of $\mathbf{y}$ to be checked by Verify function. Then, success probability can be formulated as

$$\Pr[\mathcal{A}(p) = 1] = \prod_{i=0}^{s-1} \left(\frac{|\mathbf{x}| - p \cdot |\mathbf{x}| - i}{|\mathbf{x}| - i} \right) \ll (1-p)^s. \quad (1)$$

□

Lemma 4. FairDEx *satisfies the sender fairness property.*

Proof Sketch. Sender fairness implies that $\mathcal{R}$ cannot obtain any useful information on $\mathbf{x}$ (other than $\mathsf{desc}_\mathbf{x}$) unless $\mathcal{S}$ receives the payment. In other words, an adversarial receiver $\mathcal{A}$ should not obtain $\mathbf{x}$ without the payment.

There are two cases we need to investigate: before and after the payment is published on-chain. Firstly, before the payment is published, $\mathcal{A}$ should not be able to obtain more information on $\mathbf{x}$ other than $\mathsf{desc}_\mathbf{x}$. $\mathcal{A}$ has the encrypted goods $\mathbf{y}$ and sampled subkeys $\mathbf{K}_{samp}$. Note that since the encryption algorithm Enc satisfies (IND-CPA) security, $\mathcal{A}$ cannot learn any information from $\mathbf{y}$ unless he obtains the subkeys. Thus, we need to ensure that $\mathcal{A}$ cannot obtain the rest of the subkeys. This relies on the one-wayness of the $\mathsf{SUBKGEN}$ algorithm. In other words, given a subset of subkeys k_i's, it should be infeasible to obtain the master key K. This is satisfied by using the hash function H in the random oracle model (ROM) for the subkey generation algorithm: $k_i := \mathsf{H}(K \| i)$. More precisely, given $\mathsf{H}(K \| i)$ and i it is infeasible to extract the master key K.

Secondly, after the payment is published on-chain, $\mathcal{A}$ should not claim the payment. This is satisfied by the smart contract on the blockchain because the $\mathsf{EVIDENCE}$ algorithm will return $\perp$ for the correct key K and the payment is sent to $\mathcal{S}$. Here, it is important to remember that if the on-chain payment consists of a faulty description, $\mathcal{S}$ would not provide the master key K. As explained previously, without K, $\mathcal{A}$ cannot obtain the goods. □

Note that our protocol satisfies probabilistic fairness for the receiver, which is characteristic in sampling-based protocols. Now, we discuss the rational behavior of the sender and show that it is beneficial for $\mathcal{S}$ to follow honest behavior. Let us investigate the rational behavior over the example scenario of data annotation. Here, we formulate the expected gain of a rational $\mathcal{S}$ with respect to the honest workload, i.e., the amount of correctly generated sub-goods. Since the payment amount is fixed and executed if $\mathcal{R}$ agrees on the description, we compute the gain by multiplying the saved workload and the chances of not getting caught. Assume $\mathcal{S}$ generates n-out-of-$|\mathbf{x}|$ of the annotations incorrectly. The workload saved by $\mathcal{S}$ from not doing the annotations is n and the proportional gain is $\frac{|\mathbf{x}|}{|\mathbf{x}|-n}$, whereas the chance of not getting caught by $\mathcal{R}$ is $\prod_{i=0}^{s-1} \left(\frac{|\mathbf{x}|-n-i}{|\mathbf{x}|-i} \right)$, which is less than $(\frac{|\mathbf{x}|-n}{|\mathbf{x}|})^s$ for any $n > 0$. The expected gain is less than $(\frac{|\mathbf{x}|-n}{|\mathbf{x}|})^{s-1}$, whereas for the honest $\mathcal{S}$ it is equal to 1. Thus, it is not rational to violate the protocol for any number of sub-goods.

IV. Performance Evaluation

In this section, we evaluate the efficiency of FairDEx. We begin by analyzing the complexity of the *initialization* phase of our protocol, in which messages are exchanged off-chain. Next, the on-chain cost of our protocol is measured by running experiments on the blockchain environment. We analyze the on-chain cost in two scenarios, namely optimistic and pessimistic. In the optimistic scenario, both parties act honestly, whereas, in the pessimistic scenario, one of the parties deviates from the protocol.

a) Test scenario and parameters: We investigate a scenario where $\mathbf{x}$ is digital goods of size N bytes and contains $|\mathbf{x}|$ sub-goods where each sub-good x_i can be individually verified by the receiver. Our protocol can be used whenever $|\mathbf{x}|$ and s fits the security margin declared by the parties. In our experiments, the minimum number of sample size is $s = 64$ and the probability of a sub-good being corrupted, p, is set to 0.05. With these parameters, Eq. 1 yields that the probability of fooling the receiver is lower than 0.3% in our experiments. It is worth noting that the upper bound of the probability given in Eq. 1 is independent of N and $|\mathbf{x}|$.

The timeliness is checked by calculating the difference between block timestamps and comparing to the T_{lock} and T_{obj}. In our experiments, both T_{lock} and T_{obj} are set to 10 minutes. In real-world applications, parties can determine these values according to their convenience.

b) Implementation: Our prototype is implemented in Solidity language and runs on Ethereum blockchain [38], which is the most commonly used smart contract platform. Ethereum Virtual Machine (EVM) provides an interface to `keccak256` hash function, of which our prototype takes benefit. For concatenation, we utilized encoding functions available in Solidity.

Client applications are implemented in Python 3 and use Web3.py [39] library to interact with the blockchain. Sender and receiver programs derive keys using SHA3 and encrypt/decrypt goods with AES.

c) Communication Complexity: In the off-chain part of our protocol, $\mathbf{r}$ can be generated in 3 rounds (see Appendix A) of which, the last round can be used to send $\mathbf{K}_{samp}$. Then, one more round is required to transfer $\mathsf{desc}_\mathbf{x}$, so that it makes 4 rounds in total. The protocol continues with the on-chain part, which can be executed in 3 rounds, where messages are exchanged with the blockchain.

d) Off-chain Computation: In a naive Python implementation of Algorithm 4 given in Appendix A, generating $\mathbf{r}$ and key derivation complete in nanoseconds. Clients achieve

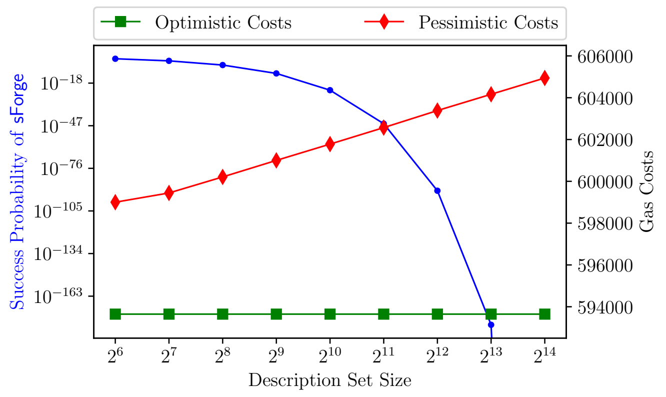


Figure 3: Execution costs of FAIRDEX with different description sizes.

323.41 MB/sec encryption/decryption throughput on a computer with 8-core Intel i7 CPU clocked at 3.6 GHz. On a significantly less powerful ARM Cortex-A53 CPU clocked at 1.4 GHz, clients can throughput 4.47 MB of data per second. The measurements demonstrate that FAIRDEX can be used on various devices, including personal computers, smartphones, and smart TVs.

A. Benchmarks

In the domain of Ethereum smart contracts, one of the most important performance criteria is the cost of execution, which is measured in *gas*. Each operation, such as the addition of two variables, taking the hash of a string or storing some data on the chain, requires a predefined gas cost, which is determined by the miners.

To measure the cost of FAIRDEX, we deployed the smart contract and run on Ropsten Network, the primary testnet of Ethereum. Deployment constitutes the main cost of running our protocol which takes about 427043 gas. For the execution, our protocol follows one of the two paths depending on the behavior of the parties. We use the term *optimistic execution* for the case where both sender and receiver follow the protocol steps honestly. In contrast, *pessimistic execution* is used for the case where any of the parties might behave maliciously. It is reasonable to expect that the pessimistic case would cost more because FAIRDEX makes a ruling after the receiver complains about the received goods. We analyze the cost of each case separately. Execution cost is calculated by summing the fees of calling the functions in the deployed contract.

a) Optimistic Case: In this case, FAIRDEX does not perform any verification operations. PayWithDescription, PublishMasterKey, and TransferToSender functions are executed and they perform merely data transfer and memory assignments. The size of written storage is always constant and takes up 96 bytes, therefore, the total cost of execution is independent of the size of the goods when parties honestly follow the protocol. In our measurements for the optimistic case, we found that deploying and running FAIRDEX cost in total 593647 gas. For a gas price of 40 Gwei, it is approximately 0.0237 ETH, which equals to \$58.18 for an exchange rate of 1 ETH = \$2455. Average prices for gas and ETH are both taken from [40] for April 26, 2021.

b) Pessimistic Case: In the pessimistic execution where one of the parties is malicious, the dispute mechanism is triggered. If the receiver is malicious and does not pay or pay to a faulty description by invoking PayWithDescription, the sender would not continue to the protocol and only the deployment cost is paid. In the case where the sender provides faulty goods by publishing a faulty master key, the receiver invokes the RaiseObjection function. As the latter case covers the cost of the first one, we calculate the pessimistic cost where the sender is being malicious.

The function RaiseObjection takes the evidence from the receiver, computes the description, and compares the published description. This is the only place where FAIRDEX performs computation, which is merely $\log s + 1$ number of executions of keccak256 in addition to a single equality comparison. The extra cost can be reduced to formula as

$$(\log s + 1) \cdot C_H + C_{EQ}, \tag{2}$$

where C_H is the cost of single call to keccak256 and C_{EQ} is the cost of equality comparison. Eq. 2 states that the cost of pessimistic execution increases linearly as the size of description exponentially grows. Fig. 3 shows the total cost of optimistic and pessimistic runs of FAIRDEX. The cost of pessimistic execution is around \$59.41 for a large description set of 16K samples. Fig. 3 confirms our observation about the gas cost estimation. Also, as in the optimistic case, the number of hash computations, therefore the cost of execution of RaiseObjection is independent of the size of the goods, N, and $|\mathbf{x}|$.

B. Performance Comparison with Closely Related Work

Here, we compare the performance of our protocol with closely related works given in Section II-B. Note that [13] is not included in the comparison since it is based on third-party validators, which is not a sampling-based protocol. Nonetheless, its on-chain cost on Ethereum is greater than 3.4M gas for the minimal scenario, which is 5x higher than ours. The comparison of the rest is given in Table I that consists of both on-chain and off-chain performances.

a) On-chain: We use the metrics of transaction size for Bitcoin and gas cost for Ethereum blockchain. In Bitcoin-based solutions, [14] and [15] have constant size transactions as they only require to publish the conditional payment on the secret key. Whereas, [16] poorly performs because of the transaction conditioned to each symmetric key of sub-goods (other than the sampled ones). In Ethereum-based solutions, our protocol outperforms because of the simple structure and usage of only cost-friendly operations like hashing. In addition, our protocol does not require the validation of the master key on-chain unless there is a dispute, and a dispute can be solved by only a couple of hashing operations.

Table I: On-chain and off-chain performance comparison of sampling-based exchange protocols

	On-chain			Off-chain		
Work	**Blockchain**	**On-chain cost**[1]	**Based on**	**Encryption**	**Computational Cost**	**Storage Cost**
[14]	Bitcoin	2 txs, Constant size	DAPS	Hybrid	$\mathcal{O}(\lvert\mathbf{x}\rvert)$(PKE+SKE)	$\mathcal{O}(\mathbf{x}) + \mathcal{O}(\lvert\mathbf{x}\rvert\kappa)$
[15]	Bitcoin	3 txs, Constant size	DAPS	Hybrid	$\mathcal{O}(\lvert\mathbf{x}\rvert)$(PKE+SKE)	$\mathcal{O}(\mathbf{x}) + \mathcal{O}(\lvert\mathbf{x}\rvert\kappa)$
[16]	Bitcoin	1 tx, $\mathcal{O}(\lvert\mathbf{x}\rvert)$ size	Hash	Symmetric	$\mathcal{O}(\lvert\mathbf{x}\rvert)$(SKE)	$\mathcal{O}(\mathbf{x})$
[17]	Ethereum	Not available[2]	Re-encryption	Hybrid	$\mathcal{O}(\lvert\mathbf{x}\rvert)$(PKE+SKE)	$\mathcal{O}(\mathbf{x}) + \mathcal{O}(\lvert\mathbf{x}\rvert\kappa)$
[18]	Ethereum	$\sim$ 930 K gas	DAPS	Hybrid	$\mathcal{O}(\lvert\mathbf{x}\rvert)$(PKE+SKE)	$\mathcal{O}(\mathbf{x}) + \mathcal{O}(\lvert\mathbf{x}\rvert\kappa)$
Ours	Ethereum	$594 + O(log(s))$ K gas[4]	Merkle Tree	Symmetric	$\mathcal{O}(\lvert\mathbf{x}\rvert)$(SKE)	$\mathcal{O}(\mathbf{x})$

[1] The cost metric is transaction size in Bitcoin, and gas cost for Ethereum.
[2] The construction is similar to [18] with an additional on-chain encryption. Thus, the cost is expected to be greater than 930K gas.
[3] The minimum possible on-chain cost, which excludes the validation cost paid to the validators.
[4] For pessimistic case, our protocol has additional MT check which is logaritmic wrt sample set size. As shown in Figure 3, the total gas cost is less than 606K even for a quite large sample set of 10K samples.

We compare our solution with Bitcoin-based ones with constant transaction size. The cost of executing our protocol is less than \$60 even in the pessimistic execution. For Bitcoin transactions mined within the next 30 mins on April 26, 2021, the cost can be estimated with 122 satoshis per byte, and an average transaction of size 250 bytes would cost \$16.17 [41]. In [14], there are two transactions with sizes of 197 and 397 bytes, which would cost around \$38.4. In [15], the authors added an additional spending condition to the transaction in [14] to fix a security attack. Also, they have an additional transaction to improve the latency of the protocol, which would cost around \$50 in total. Regarding the prices on April 26, 2021, our protocol costs 20% more than the secure Bitcoin-based solution given in [15]. However, it should be noted that the fiat comparison suffers from high fluctuations in Bitcoin and Ethereum currencies. For example, the transaction prices of April 21, 2021 are almost double of April 26, 2021 [41].

b) Off-chain: The solutions using a hybrid encryption [14], [15], [17], [18] mechanism require encryption of the goods with a symmetric key (depicted by $\mathcal{O}(|\mathbf{x}|)$ SKE) and additional encryption of these keys with a public key (depicted by $\mathcal{O}(|\mathbf{x}|)$ PKE). This additional encryption cost is also reflected in the storage cost, which is depicted by $\mathcal{O}(|\mathbf{x}|\kappa)$ where κ is the size of a symmetric key. In this manner, our protocol, together with [16], requires only symmetric key encryption of the goods and achieves the best results regarding computational and storage costs.

V. Conclusion

Almost a decade ago, fair exchange protocols were not considered feasible because of the lack of the availability of a third party. The developments in blockchain, in particular smart contracts, removed this shortcoming. If the description of goods is known in advance, then smart contracts can be used to create a conditional payment regarding the description. Otherwise, sampling-based exchange protocols can be used to generate the description. The existing sampling-based proposals require either high on-chain cost or off-chain computational and storage costs.

In this paper, we presented FairDEx that achieves low on-chain and off-chain costs. Thanks to our description procedure, it requires low cost to execute FairDEx on the blockchain and small computational power to perform off-chain operations. More specifically, the on-chain cost of FairDEx is very low and independent of the size of the goods, for both optimistic and pessimistic executions. Ethereum implementation of our protocol gives that the cost of running FairDEx is around 0.6M gas even for the pessimistic case and a sampling set of size 10K. Moreover, the off-chain part utilizes merely hashing and symmetric-key encryption algorithms, which enables high-speed computations with low resource consumption. The results show that FairDEx can be executed on resource-constrained devices like smartphones and smart TVs.

References

[1] H. Pagnia and F. C. Gärtner, "On the impossibility of fair exchange without a trusted third party," Darmstadt University of Technology, Tech. Rep., 1999.

[2] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Secure multiparty computations on bitcoin," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2014, pp. 443–458.

[3] ——, "Fair two-party computations via bitcoin deposits," in *Financial Cryptography Workshops*, ser. Lecture Notes in Computer Science, vol. 8438. Springer, 2014, pp. 105–121.

[4] S. Barber, X. Boyen, E. Shi, and E. Uzun, "Bitter to better — how to make bitcoin a better currency," in *Financial Cryptography and Data Security*, A. D. Keromytis, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 399–414.

[5] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," in *Advances in Cryptology – CRYPTO 2014*, J. A. Garay and R. Gennaro, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 421–439.

[6] S. Dziembowski, L. Eckey, and S. Faust, "Fairswap: How to fairly exchange digital goods," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, 2018, pp. 967–984.

[7] A. Kiayias, H. Zhou, and V. Zikas, "Fair and robust multi-party computation using a global transaction ledger," in *EUROCRYPT (2)*, ser. Lecture Notes in Computer Science, vol. 9666. Springer, 2016, pp. 705–734.

[8] R. Kumaresan, T. Moran, and I. Bentov, "How to use bitcoin to play decentralized poker," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: ACM, 2015, pp. 195–206.

[9] H. Pagnia, H. Vogt, and F. C. Gärtner, "Fair exchange," *Comput. J.*, vol. 46, no. 1, pp. 55–75, 2003.

[10] M. J. Crump, J. V. McDonnell, and T. M. Gureckis, "Evaluating amazon's mechanical turk as a tool for experimental behavioral research," *PloS one*, vol. 8, no. 3, p. e57410, 2013.

[11] J. Heer and M. Bostock, "Crowdsourcing graphical perception: using mechanical turk to assess visualization design," in *CHI*. ACM, 2010, pp. 203–212.

[12] A. Burmania, S. Parthasarathy, and C. Busso, "Increasing the reliability of crowdsourcing evaluations using online quality assessment," *IEEE Trans. Affect. Comput.*, vol. 7, no. 4, pp. 374–388, 2016.

[13] M. Müller, S. R. Garzon, and A. Küpper, "COST: A consensus-based oracle protocol for the secure trade of digital goods," in *DAPPS*. IEEE, 2020, pp. 72–81.

[14] S. Delgado-Segura, C. Pérez-Solà, G. Navarro-Arribas, and J. Herrera-Joancomartí, "A fair protocol for data trading based on bitcoin transactions," *Future Gener. Comput. Syst.*, vol. 107, pp. 832–840, 2020.

[15] J. Gao, T. Wu, and X. Li, "Secure, fair and instant data trading scheme based on bitcoin," *J. Inf. Secur. Appl.*, vol. 53, p. 102511, 2020.

[16] Y. Chen, J. Guo, C. Li, and W. Ren, "Fade: A blockchain-based fair data exchange scheme for big data sharing," *Future Internet*, vol. 11, no. 11, p. 225, 2019.

[17] H. Yu, J. Gao, T. Wu, and X. Li, "A novel fair and verifiable data trading scheme," in *International Conference on Frontiers in Cyber Security*. Springer, 2019, pp. 308–326.

[18] Y. Zhao, Y. Yu, Y. Li, G. Han, and X. Du, "Machine learning based privacy-preserving fair data trading in big data market," *Inf. Sci.*, vol. 478, pp. 449–460, 2019.

[19] N. Asokan, V. Shoup, and M. Waidner, "Asynchronous protocols for optimistic fair exchange," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 1998, pp. 86–99.

[20] G. Avoine, F. C. Gärtner, R. Guerraoui, and M. Vukolic, "Gracefully degrading fair exchange with security modules," in *EDCC*, ser. Lecture Notes in Computer Science, vol. 3463. Springer, 2005, pp. 55–71.

[21] M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest, "A fair protocol for signing contracts," *IEEE Trans. Inf. Theory*, vol. 36, no. 1, pp. 40–46, 1990.

[22] S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts," in *CRYPTO*. Plenum Press, New York, 1982, pp. 205–210.

[23] M. O. Rabin, "Transaction protection by beacons," *J. Comput. Syst. Sci.*, vol. 27, no. 2, pp. 256–267, 1983.

[24] N. Asokan, M. Schunter, and M. Waidner, "Optimistic protocols for fair exchange," in *CCS*. ACM, 1997, pp. 7–17.

[25] W. Banasik, S. Dziembowski, and D. Malinowski, "Efficient zero-knowledge contingent payments in cryptocurrencies without scripts," in *ESORICS (2)*, ser. Lecture Notes in Computer Science, vol. 9879. Springer, 2016, pp. 261–280.

[26] M. Campanelli, R. Gennaro, S. Goldfeder, and L. Nizzardo, "Zero-knowledge contingent payments revisited: Attacks and payments for services," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*. ACM, 2017, pp. 229–243.

[27] G. Fuchsbauer, "WI is not enough: Zero-knowledge contingent (service) payments revisited," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019*. ACM, 2019, pp. 49–62.

[28] L. Eckey, S. Faust, and B. Schlosser, "Optiswap: Fast optimistic fair exchange," in *AsiaCCS*. ACM, 2020, pp. 543–557.

[29] E. Wagner, A. Völker, F. Fuhrmann, R. Matzutt, and K. Wehrle, "Dispute resolution for smart contract-based two-party protocols," in *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2019*. IEEE, 2019, pp. 422–430.

[30] S. Janin, K. Qin, A. Mamageishvili, and A. Gervais, "Filebounty: Fair data exchange," *CoRR*, vol. abs/2008.11362, 2020.

[31] M. Blum, "How to exchange (secret) keys," *ACM Trans. Comput. Syst.*, vol. 1, no. 2, pp. 175–193, 1983.

[32] I. Damgård, "Practical and provably secure release of a secret and exchange of signatures," *J. Cryptology*, vol. 8, no. 4, pp. 201–222, 1995.

[33] O. Goldreich, "A simple protocol for signing contracts," in *CRYPTO*. Plenum Press, New York, 1983, pp. 133–136.

[34] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, no. 1, pp. 36–63, 2001.

[35] B. Poettering and D. Stebila, "Double-authentication-preventing signatures," in *ESORICS (1)*, ser. Lecture Notes in Computer Science, vol. 8712. Springer, 2014, pp. 436–453.

[36] N. P. Smart, *Cryptography Made Simple*, ser. Information Security and Cryptography. Springer, 2016.

[37] R. C. Merkle, "A digital signature based on a conventional encryption function," in *CRYPTO*, ser. Lecture Notes in Computer Science, vol. 293. Springer, 1987, pp. 369–378.

[38] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014.

[39] n/a, "web3.py," https://github.com/ethereum/web3.py, 2021.

[40] EtherScan, "The ethereum blockchain explorer," 2021, available at: https://etherscan.io/.

[41] PrivacyPros, "Bitcoin transaction fee estimator & calculator," 2021, available at: https://privacypros.io/tools/bitcoin-fee-estimator.

Appendix

A. Rand *and* Verify *Functions*

Rand function starts with sender choosing a random value $\mathsf{r}_{\mathcal{S}}$ and computing the commit $h_{\mathcal{S}} := \mathsf{H}(\mathsf{r}_{\mathcal{S}})$ where H is a secure hash function. Sender sends $h_{\mathcal{S}}$ to the receiver. Next, receiver generates $\mathsf{r}_{\mathcal{R}}$ and sends it to sender. Then sender opens $\mathsf{r}_{\mathcal{S}}$, *i.e.,* sends $\mathsf{r}_{\mathcal{S}}$ to receiver. Having the random value of the other party, parties obtain joint random number $\mathsf{r} = \mathsf{r}_{\mathcal{S}} \oplus \mathsf{r}_{\mathcal{R}}$. At this point, if one of the parties misbehaves such as by not opening the commitment, the other party aborts and stops the exchange. Otherwise, the randomness r is used to generate s distinct indexes $\mathbf{r} := \{r_i\}_{i=1}^{s}$ where $r_i \in (1, |\mathbf{x}|)$ using Algorithm 4.

Algorithm 4 Generate random indexes.

```
1:  function GenIndex($\mathsf{r}$, $|\mathbf{x}|$, $s$)
2:      $\mathbf{r} = \emptyset$
3:      $R \leftarrow \{\}$                          ▷ Initialized as empty array.
4:      for $i \in (1, |\mathbf{x}|)$ do
5:          $h_i \leftarrow \mathsf{H}(\mathsf{r} \,||\, i)$
6:          $\mathsf{Append}(R, h_i)$           ▷ $h_i$ is the last element of $R$.
7:      $R_{\mathcal{S}} \leftarrow \mathsf{SortIncreasing}(R)$       ▷ Sort elements of $R$.
8:      for $i \in (1, s)$ do
9:          $h \leftarrow \mathsf{ElementAt}(R_{\mathcal{S}}, i)$  ▷ Get $i^{th}$ element of $R_{\mathcal{S}}$.
10:         $r_i \leftarrow \mathsf{IndexOf}(R, h)$       ▷ Find index of $h$ in $R$.
11:         $\mathbf{r} = \mathbf{r} \cup r_i$
12:     return $\mathbf{r}$
```

In Verify function, $\mathcal{R}$ decrypts the goods for corresponding indexes of $\mathbf{r}$. Then, check if the sub-good is correct or not. If all of the sub-goods are correct, then it returns true. Otherwise, it returns false. For the example scenario of manual data annotation, $\mathcal{R}$ would check whether each data point of $\mathbf{r}$ has the correct annotation or not.