Sepideh Avizheh*, Preston Haffey, and Reihaneh Safavi-Naini

Privacy-preserving FairSwap: Fairness and privacy interplay

Abstract: Fair exchange protocols are among the most important cryptographic primitives in electronic commerce. A basic fair exchange protocol requires that two parties who want to exchange their digital items either receive what they have been promised, or lose nothing. Privacy of fair exchange requires that no one else (other than the two parties) learns anything about the items. Fairness and privacy have been considered as two distinct properties of an exchange protocol. In this paper, we show that subtle ways of leaking the exchange item to the third parties affect fairness in fair exchange protocols when the item is confidential. Our focus is on FairSwap, a recently proposed fair exchange protocol that uses a smart contract for dispute resolution, has proven security in UC (Universal Composability) framework, and provides privacy when both parties are honest. We demonstrate, however, that FairSwap's dispute resolution protocol leaks information to the public and this leakage provides opportunities for the dishonest parties to influence the protocol's fairness guarantee. We then propose an efficient privacy-enhanced version of FairSwap, prove its security and give an implementation and performance evaluation of our proposed system. Our privacy enhancement uses circuit randomization, and we prove its security and privacy in an extension of universal composability model for non-monolithic adversaries that would be of independent interest.

Keywords: Fair exchange, Smart contract, Privacy, Probing attack

DOI 10.2478/popets-2022-0021
Received 2021-05-31; revised 2021-09-15; accepted 2021-09-16.

1 Introduction

A growing number of interactions in cyber space are commercial transactions that involve exchange of electronic "goods" that are in the form of signatures (e.g. contracts and receipts), confidential data (i.e. digital items), or payments.

Transaction fairness provides confidence for a party that they will not be worse off compared to other parties in the protocol. There are a number of definitions for fairness with slight differences that are motivated by the exchange scenario. An exchange scenario includes *exchanging parties* that have items to be exchanged, and *non-exchanging* parties that assist with the correct execution of the exchange.

In this paper, we focus on *two-party exchange protocols*, where the two exchanging parties P_1 and P_2 want to exchange their items x_1 and x_2, respectively. In general, the items can be (i) *confidential data*, "which is not known to the recipient beforehand and will be released to it during the protocol run", (ii) public data such as the text of a contract "which may be released even if the protocol execution has not been successful" and requires exchange of non-repudiable evidences, such as signature of the agreeing parties, and (iii) *payments* that transfer "money" from the payer to the payee's account [5]. We consider confidential data in the form of digital items with defined properties that can be efficiently checked (e.g. by evaluating a predicate $\phi()$) by the holder of the item.

The exchange of items is called *fair* if it guarantees that when one party behaves honestly, then the exchange terminates, and either P_1 gets x_2 and P_2 gets x_1, and both items satisfy the claimed properties, or P_1 gets no information about x_2 and P_2 gets no information about x_1. A number of impossibility results [20, 29, 42] have shown that two-party fair exchange without a trusted entity is impossible. The structure of the trusted party (e.g. a single entity or a collection of entities), their involvement in the protocol (e.g. always online, or offline), and the level of trust on them (fully trusted, or semi-trusted) varies among different protocols.

In *optimistic fair exchange (OFE)* the trusted party, referred to as *judge*, is only invoked for resolving disputes, and as long as the two parties behave honestly, the judge will be offline. This makes OFE protocols optimal in terms of using non-exchanging entities.

***Corresponding Author: Sepideh Avizheh:** University of Calgary, AB, Canada
Preston Haffey: University of Calgary, AB, Canada
Reihaneh Safavi-Naini: University of Calgary, AB, Canada

Privacy in fair exchange protocols requires no leakage of the exchanged items to the non-exchanging parties, and captures various aspects of the exchange that is relevant to the type of the exchanged items. Privacy in the exchange of confidential items can protect aspects such as the value of the item (e.g. proprietary digital items) or linkability of the item content to the exchanging parties. In the exchange of public data (contract signing) privacy protects intention and signatures of exchanging parties from non-exchanging parties, and privacy of payment in an exchange protects one or more aspects of a payment such as the paid amount, the payer or the payee.

We consider optimistic fair exchange of confidential items when the judge (non-exchanging third party) is implemented by a smart contract (SC), that may have additional tasks such as transfer of coins (see [7, 10]). We focus on FairSwap [24], a protocol for exchange of a confidential item with payment where the judge is implemented by a smart contract. We take the view of the author's: "One may view smart-contract based solutions as a variant of optimistic protocols, where the smart contract takes the role of the TTP" (Section 1.2, paragraph 2 in FairSwap).

Smart contracts are transparent programs that enjoy trusted execution on a blockchain consensus-based computation network. Smart contracts cannot hold secrets, and their execution is visible to the public. Using SC to implement the judge in an OFE (referred to as *SC-aided OFE*) is attractive because the SC provides trust and transparency for, and automates the execution of, the judge algorithm. It also seamlessly integrates the exchange with the blockchain cryptocurrency system. Transparency of the SC, however, means any leakage to the judge is effectively leakage to the public.

In this paper, we show that the dispute resolution protocol in FairSwap leaks information about the digital item, and this provides an opportunity for the buyer or seller to influence the fairness of the exchange.

1.1 Our work

In FairSwap a buyer wants to buy a "digital commodity x", that we consider to be a confidential item, from a seller and is willing to pay p coins if x satisfies a predicate $\phi(x) = 1$ that is represented by a circuit. If parties are honest, buyer receives x and SC transfers the p coins to the seller. If x does not satisfy $\phi(x) = 1$, the buyer invokes a dispute resolution (DR) protocol that allows them to construct a *proof of misbehavior (PoM)* that is sent to the SC. The PoM enables the SC to correctly decide if it should transfer the coins to the seller, or refund the buyer. Fairness of the exchange is proven in the UC (Universal Composability) framework of Canetti [16, 17].

Our main observation is that the dispute resolution protocol in FairSwap leaks information about the confidential item to the SC, and hence to the public and this leakage provides opportunity for a malicious party (buyer or seller) to influence execution of the protocol in such a way that the honest party would be willing to forego the guaranteed fairness of the protocol. This motivates us to require privacy, in the sense that no information about the item is leaked to non-exchanging parties, for the DR process in addition to the execution of the protocol by two honest parties. We propose an approach to represent the computation and revise the protocol accordingly to provide privacy while maintaining efficiency of FairSwap. We model security of the protocol and privacy against smart contract using an extension of UC model, and prove security and privacy of our protocol. We also provide implementation and experimental results on the efficiency of our design. These are elaborated below.

The argument for preventing information leakage during dispute resolution in SC-aided fair exchange protocols is general, and the approach for achieving privacy in circuit based property checking protocol's like FairSwap can be of independent interest.

Fairness and privacy in FairSwap. FairSwap consists of a main protocol π_1 that is executed when the two exchanging parties are honest, and a dispute resolution protocol π_2 that is called if the seller has not fulfilled their promise. Thus, the execution of FairSwap can involve execution of, (i) π_1 alone, or (ii) $\pi_1 \circ \pi_2$ which is the composition of two protocols. The exchange protocol must protect confidentiality of x with respect to SC (and hence the public) when both parties are honest, and so the protocol π_1 must be privacy preserving. We argue that π_2 must not leak any information about the item. This may appear an unusual requirement because the DR protocol is designed to be called by the buyer to complain about a dishonest seller, for whom one need not be concerned about confidentiality of the item. However, as we show below the information leakage of DR can be abused by a dishonest buyer by making unnecessary call to the protocol, or harm an honest buyer when the seller is malicious and the protocol must be called. In the following, we give hypothetical, yet realistic scenarios, where leakage during DR puts the dishonest party in a position to create credible threats that affect the fairness guarantee of the protocol. Note that we take the view that leakage during the protocol execution is

distinct from leakage outside the protocol and only consider the former here. Once the protocol is completed, the protocol does not provide any guarantee for the information that participants hold. For example, a buyer can always publish or leak the item that they have received after the protocol ends. Protection of information outside the protocol will rely on additional mechanisms, technological and legal, and will be scenario dependent and not our consideration.

Malicious buyer. Consider a contract that requires payment of p_1 when the item satisfies ϕ_1, and payment of $p_2 > p_1$, when the item satisfies ϕ_2. The predicate will be $\Phi = (\phi_1 \wedge \phi_2)$. The SC will consider the protocol execution as honest if the two parties agree on one of the two ϕ_1 or ϕ_2, and the associated payment and execute π_1 only (no call to the DR, π_2). The dishonest buyer commits to the payment p_2 and receives the item that correctly satisfies ϕ_2. The buyer will use an out-of-band channel to request lower payment, else they would call the DR protocol that will partially leak the item. If the honest seller does not accept the lower payment p_1, the dishonest buyer could call the DR protocol during which part of the item will be leaked to the public. The buyer will ultimately lose the dispute resolution protocol and will pay p_2 for the item, as they were supposed to. If the honest seller accepts the lower pay offer, the buyer sends a transaction to the seller which claims that the item satisfies p_1, and the seller responds with a refund transaction with the value $p_2 - p_1$. These are additional public transactions that will be exchanged before the SC declares the protocol complete. In this scenario the leakage of the dispute resolution has put the dishonest buyer in an advantageous position.

Malicious seller controls their confidential item, and can publish it as part of the protocol messages at their will (e.g. use unencrypted channel to deliver the item to the buyer). However, there can also be scenarios where a malicious seller can benefit from the leakage of the DR and create a privacy dilemma for the buyer. For example, consider a malicious seller who claims to offer an item with property ϕ_1 for payment p. After an honest buyer commits to the payment, the seller will provide an item that does not satisfy ϕ and is personalised with the buyer's personal information (e.g. identity). If the buyer uses DR, part of the item content will be leaked through transaction traces of the protocol, and this leakage can be used to link the item content to the buyer. Here the seller does not even need to communicate their malicious intent; the buyer knows that if they complain, their purchasing preferences will be made public.

Note that although the buyer may use a pseudonym to provide some level of anonymity and untraceability for their exchange, this is not a requirement of FairSwap or a general SC-based OFE protocol. Here the leakage of the DR puts the dishonest seller in an advantageous position.

The above scenarios show that leakage from the DR protocol can put one party in an advantageous position. We note that through DR, a malicious party can leak (part of) the content without fear of being prosecuted because DR is part of the protocol specification. In contrast, leaking outside the protocol may have legal consequences that would dissuade a malicious party or require careful planning by them.

In both cases, the leakage of DR creates a credible threat strategy for the malicious party in the sense that the threat serves their interest, and they are willing to carry out the threat without becoming disadvantaged. Furthermore, both parties can use the DR protocol in a way that maximizes the leakage and thus the effectiveness of their threat. For example, the buyer is free to choose the parts of x that is most sensitive and must remain confidential.

One can design smart contracts to include incentive mechanisms to remove these credible threats; for example, add penalties for a false claim by the buyer. This however requires additional mechanism design and game theoretic analysis.

The above discussion highlights the peril of DR leakage and its potential effect on the protocol fairness, and motivates the need for privacy (preventing leakage to non-exchanging parties) in *all executions of SC-aided OFE protocols*.

We define fairness and privacy requirements of a two-party SC-aided OFE for the exchange of a confidential digital item for coins, as follows.

(FE1): Both exchanging parties are honest: Seller gets p coins, buyer gets the item x.

(FE2): One party is honest:

(a) Buyer fairness: if buyer is honest, either buyer receives correct x, or does not lose any coins.

(b) Seller fairness: if the seller is honest, either the seller receives p coins, or does not reveal any information about x to the buyer.

(PR): SC learns negligible information about the confidential item, even if one exchanging party does not follow the protocol.

This is the basic privacy requirement that is necessary for the protection of the item's confidentiality, and can

be extended to include payment privacy which we leave as a future work.

Privacy preserving FairSwap. In FairSwap ϕ is expressed by a circuit that receives a digital item x as input and outputs a binary value true or false. The seller commits to a sequence that is the concatenation of the encryption of x and the intermediate computation results of the gates in ϕ by constructing a Merkle tree on the sequence, and sending its root to the SC. In the case of a dispute, the buyer constructs a PoM using the decrypted sequence and sends it to the SC. PoM leaks at least 2γ where γ is the input bit length of the "gates" (with fan-in 2) in ϕ. A gate in general can perform a complex operation, and the leakage can even be much larger than the security parameter. In FairSwap's implementation, for example, ϕ is constructed using "hash gates" (keccak256 hash), each compressing its input.

To provide privacy for the computation, an immediate solution is to express ϕ as a Boolean circuit in which case the leakage is reduced to 2 bits and one can argue that this is "negligible" in most cases. This, however, will have significant computation cost for the seller (expressing the predicate as a Boolean circuit, and evaluating it on the binary sequence corresponding to the input), making the protocol impractical for predicates of interest (e.g. hash of a file). A second solution is to express ϕ as an arithmetic circuit and use fully homomorphic encryption to perform the computation of the circuit in encrypted domain. This again will have unacceptably high computation cost for the seller and the SC.

Our approach is to use a circuit randomization technique that has been used for side channel protection [34] in this new context, to hide the leakage of information in DR to the SC. Our observation is that when ϕ is represented as an arithmetic circuit, the PoM includes the values of wires in the circuit description of $\phi(x)$, and the leakage can be modeled as a wire probing attack on the circuit.

We then extend the circuit randomization approach of [34] to arithmetic circuits over $\mathbb{F}_p$, and construct *gadgets* for addition and multiplication using additive $(m+1,\ m+1)$ secret sharing, where $m = 2\ell$ and ℓ is the number of wires that are observable by the adversary, that effectively replaces each wire in the circuit with $m+1$ wires. This ensures that the exposed wire to the SC do not leak any information about the input.

Circuit randomization will be used by the buyer and the seller. Both parties use a secure key agreement protocol to obtain a common random seed that will be used for generating the required randomness with a cryptographic pseudorandom generator. This seed must be kept private from SC. Using this randomization requires modifying FairSwap to include commitments to the random values that are used for circuit expansion (to make it verifiable by the buyer and the SC if seller uses different randomness), commitments to the encrypted computation results of the expanded circuit, and the description of the expanded circuit. The complete description of privacy preserving FairSwap, referred to as pFairSwap is in Section 4.

Modeling and proving security and privacy. To model fairness and privacy of pFairSwap, we need to use an extension of UC model for non-monolithic adversaries that was proposed in [35]. Standard UC framework considers monolithic adversaries and uses a single simulator to represent adversarial behaviours of the parties in the ideal world. For security against dishonest exchanging parties (i.e. buyer and seller) and privacy against the SC (non-exchanging semi-honest party), we use *non-monolithic adversaries* that model multiple (in our case two) adversaries that do not collude. We extend the framework of [35] to include global functionalities (e.g. distributed ledger functionality), and modify the ideal world in FairSwap to include SC as an entity. The new FairSwap ideal functionality has the same fairness description of [24], and additionally provides a description of privacy against SC. The security proof in Section 5 describes the interaction between the two types of adversaries.

Proof of concept implementation: We demonstrate the feasibility of pFairSwap in Section 6 with a proof of concept implementation. To evaluate the cost of circuit randomization we implement relevant subroutines from the original FairSwap protocol to work with an arithmetic circuit description of $\phi()$. We implement pFairSwap with subroutines that randomizes the circuit and its input. We use our implementation to compare the performance of FairSwap and pFairSwap for the buyer and seller (off-chain computation) and for the smart contract.

To measure the on-chain computation we implement an Ethereum smart contract that verifies a PoM. We compare the computation cost to verify a PoM from an original circuit and expanded private circuit. For the off-chain computation, we measure the execution time of the buyer and the seller. We use the Pinocchio Verifiable Computing suite [43] available on GitHub [31] to build and execute arithmetic circuits. We implement the scenario where a buyer would like to purchase a digital file x with a published hash value that is computed using a universal hash function.

Discussion and extensions. Our work points out the subtle relation between information leakage in executions of SC-aided OFE for confidential items with the fairness guarantee of the protocol because of the transparency of SC and leakage to the public. Requiring no leakage to the SC *in all executions of the protocol* restores fairness guarantee of the protocol in FairSwap. An SC-aided FE is an example of a semi-honest third party based FE with the additional restriction that the SC cannot hold any secret. More fine-grained definitions of privacy against a semi-honest third party is an interesting research question. Our protection mechanism requires the predicate to be described as an arithmetic circuit. Design of specialized randomized transformation for more general gates and with respect to an associated PoM is also an interesting research question.

Multi-party computation (MPC) and fair exchange (FE). MPC and FE problems have similarities but as noted in [30], they originate in the two fields of cryptography and distributed systems respectively, and so it is not straightforward to compare and relate the results. In [30], authors propose a common framework called *fair general computation* to elucidate the relation between the two. Using this framework, FE is modeled as a deterministic function $F : (y_1, \ ..., \ y_n) = F(x_1, \ ..., \ x_n)$, where x_i and y_i are input and output of party P_i respectively, and the function F, when all parties are correct, defines a permutation with no fixed points. Privacy and fairness are two properties (in addition to other properties validity, uniqueness, nontriviality and termination) that a FE protocol must satisfy. In particular, privacy means that no party P_j outputs "the input value x_i or the output value y_i of any correct process P_i, apart from what is possibly given away by inputs and outputs of Byzantine processes." Fairness requires that "if a party P_i outputs a value y_i, with $y_i \in Y_i$, then every correct process P_j outputs a value y_j, with $y_j \in Y_j$, unless P_i is Byzantine and y_i is computable from the inputs of Byzantine processes".

A two party OFE with a single trusted party (trusted party can be a set of participants) such as FairSwap, can be seen as a three party protocol in the above setting, where two exchanging parties have inputs x_1 and x_2, and a third non-exchanging party has input $\perp$ (empty). The functional description of OFE is given by $F_X(x_1, \ x_2, \ \perp) = (out_1, \ out_2, \ out_3) = (x_2, \ x_1, \ \perp)$. Fairness is the security of the exchange and effectively requires that either both exchanging parties receive the output of FE or none of them does. Privacy of the exchange means that the output of SC be empty. Our work shows that because the output of SC is a public value, even when one of the parties is dishonest, the output of SC must be empty. A number of works have modeled fair exchange protocols as an MPC problem [2, 3]. These protocols use traditional TTPs and not an SC.

1.2 Related work

Impossibility of fair exchange without a trusted party is proven in both synchronous model [29] and asynchronous model [42]. A trusted party need not be centralized and can be implemented as a set of distributed tamper-proof devices [8, 9, 28]. Fairness without TTP can be achieved through gradual release or probabilistic fairness [9, 11, 13, 21]. Asokan et al. introduced optimistic fair exchange [5] which generated a large number of follow up works [6, 7, 10, 14, 44] (and many more). Privacy of two-party fair exchange is motivated differently depending on the type of exchanged data. For confidential items privacy is required because of the value of the items, and requires that non-exchanging parties do not learn any information about the exchanged items and this requirement can be extended to the case that one of the parties deviates from the protocol also [9, 10].

In contract signing however the data (contract) can be made public but the information about the signing parties may need different types of protection. For example in [32] the requirement is that all information about the signatures of the signing parties, or "any evidence about an exchange between them even after the resolution of a dispute" must be hidden. Authors introduce Privacy-Preserving OFE (P^2OFE) that requires that the arbitrator not learn the signer's full signature. In Ambiguous Optimistic Fair Exchange (AOFE) [27, 33] however, the requirement is that a partial signature (initial commitment) not leak information about the signer, as such partial signatures can reveal Alice's "will/intention to do exchange with Bob, from which Bob may take advantage of and could be unfair to Alice" [33].

Blockchain and smart contracts as a TTP has been used in [4, 12, 19, 24, 25, 36, 38–40] (and many more). Smart-contract based solutions can be seen as variants of OFE where the smart contract takes the role of the TTP. A recent line of work that achieves fairness is by using cryptocurrencies such as Bitcoin to penalize dishonest behaviour and achieve fairness [4, 12, 36, 38, 39]. We study FairSwap [24] that is an efficient realization of the claim-or-refund functionality that was introduced in [4, 12]. OptiSwap optimizes FairSwap for the optimistic case by reducing the initial computation and communication complexity at the cost of an interactive dispute resolution protocol.

Privacy of smart contract is a growing research topic. Approaches such as [37] use specific private functions whose executions are off-chain using trusted entities (that can be realized through trusted hardware) and protect confidential data. Very recently, there has been a large body of work on algorithm fairness and the relation between privacy and fairness [22, 23, 26].

2 Preliminaries

In this section, we give notations and cryptographic primitives that we have used in this paper.

2.1 Notation

We use $\mathcal{S}$ with Sender/Seller interchangeably and $\mathcal{R}$ with Receiver/Buyer interchangeably. We stick with many notation conventions presented in FairSwap, but outline them here thoroughly for completeness. $[n]$ denotes the set of natural numbers $\{1, ..., n\}$. $x = (x_1, ..., x_n)$ is a digital good that $\mathcal{R}$ would like to buy and $\mathcal{S}$ would like to sell. Concretely, x is a bit string constructed by concatenating n chunks of x_1, ..., x_n. We say that x is of length n. Each chunk x_i where $i \in [n]$ is of length λ bits.

Proof of Misbehaviour (PoM) is the construction developed in FairSwap that allows the smart contract to adjudicate a dispute between $\mathcal{R}$ and $\mathcal{S}$. The proof of misbehaviour π is a tuple of Merkle proofs that are sent to and verified by the smart contract, we refer to this tuple as $\pi = (\pi_{\phi_i}, \pi_{out_i}, \pi_{I_i[i]}, \pi_{I_i[j]})$; π_{ϕ_i} represents the Merkle proof that gate ϕ_i is indeed in the circuit ϕ (its description including its position, operation, and inputs and outputs indices are all correct), π_{out_i} represents the Merkle proof that the output of the gate has been committed by the sender, $\pi_{I_i[i]}$ (and $\pi_{I_i[j]}$) is the Merkle proof that shows the input $I_i[i]$ (and $I_i[j]$) has been committed by the sender. This tuple includes information that is generated and committed to through out the protocol. When we use PoM we refer to this process of generating the information and commitments.

2.2 Cryptographic primitives

In this section, we describe some of the cryptographic primitives that are used in our work.

Cryptographic hash functions: A hash function $\mathcal{H}$ is a function that maps an arbitrarily sized input to a fixed size output $\{1,0\}^* \rightarrow \{1,0\}^\mu$, for sufficiently large μ. We consider a secure hash function $\mathcal{H}$ which is collision resistant (such as SHA3). We model $\mathcal{H}$ as a Global (restricted programmable and observable) random oracle function described in section 3.

Pseudorandom number generator (PRG): PRG is used to generate a sequence of psuedorandom numbers. A PRG is an efficient, deterministic algorithm that expands a short uniformly random seed s_0 to a longer pseudorandom output y, such that no efficient algorithm can distinguish the output of the PRG from a truly random string y.

Symmetric encryption scheme: A symmetric encryption scheme is defined by three algorithms $\{Gen,\ Enc,\ Dec\}$. Key generation accepts a security parameter λ and outputs a key k of size λ bits, $Gen(\lambda) = k$. The encryption takes a plaintext message m and encryption key k and outputs a ciphertext z, $Enc(k, m) = z$. The decryption algorithm decrypts a ciphertext z using the key k to recover the plaintext m, $Dec(k, z) = m$. An encryption scheme is IND-CPA secure if the success probability of an adversary, with access to an encryption oracle, distinguishing a ciphertext constructed on a randomly chosen message from two-element message space, is only negligibly higher than the success probability of random guessing.

Commitment scheme: A commitment scheme consists of two algorithms $\{commit,\ open\}$ and it is used to commit to a specific value without revealing what that value is. To commit to a value k, we run the probabilistic algorithm $(c,\ d) = commit(k)$ that outputs commitment c, and opening d. To validate the commitment, we run $open(k,\ d,\ c)$ which outputs either 1 (valid), or 0 (invalid). The commitment scheme needs to be hiding, the commitment $(c,\ d)$ reveals no information about k and binding, only $(k,\ d)$ will open the commitment c.

Merkle tree: A Merkle tree is used to commit to a sequence of data elements $x = (x_1, ..., x_n)$. A Merkle tree is a tree where non-leaf nodes are the hash of their children nodes concatenated together. We consider a randomized Merkle tree $Mtree$ that is used in FairSwap (Appendix B.1 [24]) in which each leaf x_i node is concatenated with a distinct uniformly random value $d_i \in \{0,1\}^\kappa$, where κ is the security parameter. We modify this Merkle tree by replacing the leaf nodes by their corresponding hash values of $x_i||d_i$. $Mtree$ accepts a sequence of elements $x_i||d_i$ and returns a Merkle tree M. The pair (i) root of the Merkle tree $r_x = root(Mtree(x||d))$ and (ii) the vector of the chosen random values $d = (d_1\ , ...,\ d_n)$, is a commitment to the sequence of elements $x_1,\ x_2,\ ...,\ x_n$ and can be used to verify that an element x_i belongs to x at position i through verifying a Merkle proof. A Merkle proof is a path β in M, $\beta = MProof(i, M)$. The function $Mvrfy$ can use the Merkle proof β_i to verify membership of item x_i. This randomized Merkle root commitment is com-

putationally hiding in the global random oracle model, and it is computationally binding assuming one party (either the committer or receiver) is honest.

Arithmetic circuit: Our protocol uses arithmetic circuits (ACs) which are directed acyclic graphs (DAGs) whose nodes are gates and edges are wires. Gates perform operations from an instruction set $\Gamma = \{\times, +\}$ where $\times$ is multiplication and $+$ is addition over a finite field $\mathbb{F}_p$. Wires directed into a gate are called input wires and wires directed from a gate are called output wires. The number of input wires in a gate is called its fan-in. Addition and multiplication gates have a fan-in of two. We provide an *ℓ-private* circuit transformer that transforms arithmetic circuits into an *ℓ-private* arithmetic circuit. These circuits use negation gates which is the combination of multiplication gate by -1 and an addition gate. The instruction set for the private circuits is $\Gamma = \{-\} \cup \{\times, +\}$. Input to the circuit is denoted by $x = (x_1, ..., x_n)$. Each chunk of the input x_i for $i \in [n]$ is modeled as a special input gate that has a fan-in zero and outputs x_i on it's output wire(s). Each gate in a circuit ϕ is described as a tuple $\phi_i = (i, op_i, I_i)$ (aka gate description) where i is the index of the gate, $op_i \in \Gamma$ is the gate operation, and I_i is an array of indices that identify gates in ϕ whose output wires are directed into gate ϕ_i. The output of ϕ_i is out_i and can be expressed as $out_i = op_i(out_{I_i[1]}, ..., out_{I_i[t]})$. To denote the execution of the circuit ϕ on an input x, we write $\phi(x)$.

Universal hash function: Universal hash functions are keyed hash functions calculated over a prime field $\mathbb{F}_p$. They take an input $x = (x_1, ..., x_v)$, where $x_i \in \mathbb{F}_p$. This function computes the linear combination of the input blocks with v random elements from $\mathbb{F}_p$. This family of functions has the property that the probability of $h(x) = h(y)$ when $x \neq y$ is $\frac{1}{p}$ when the key is chosen independent of the input [18].

2.3 FairSwap protocol description

Fairswap [24] is an optimistic fair exchange protocol that allows a receiver, $\mathcal{R}$, to receive an item x that satisfies a property ϕ from a sender, $\mathcal{S}$, in exchange for a price of p coins. We say x satisfies ϕ when $\phi(x) = 1$. $\mathcal{S}$ and $\mathcal{R}$ communicate through a secure channel. A smart contract, called a Judge contract $\mathcal{J}$ transfers p coins and resolves disputes between $\mathcal{R}$ and $\mathcal{S}$.

The predicate ϕ is modeled as a circuit. $\mathcal{S}$ runs $\phi(x)$ and encrypts the output out_i of each gate ϕ_i in the circuit. The encrypted wires $z = (z_1, \cdots, z_m)$ is then sent to $\mathcal{R}$ who decrypts and verifies each gate in ϕ. If $\mathcal{R}$ finds that $\phi(x) \neq 1$ they can prove this to $\mathcal{J}$ with a single gate in $\phi(x)$, where this gate is corrupted and either it has one input which is different from the value $\mathcal{S}$ has already committed, or the computation of the gate is not correct. $\mathcal{J}$ can verify the validity of the complaint by checking the commitments of $\mathcal{S}$ or performing the computation of the single gate. FairSwap has three stages:

Initialization: $\mathcal{S}$ encodes x with a subroutine *Encode*. Its inputs are x, ϕ, and a key k for a secure symmetric key encryption algorithm *Enc*. It outputs the encryption of all the output wires in $\phi(x)$ as an array $z = (z_1, ..., z_m)$ where $z[i] = Enc(k, out_i)$ and out_i is the output of each gate ϕ_i, and m is the number of gates in ϕ. $\mathcal{S}$ sends the following commitments to $\mathcal{J}$: the Merkle root, $r_z = root(Mtree(z))$, the Merkle root, $r_\phi = root(Mtree(\phi))$, and a commitment c to the key $(c, d) = commit(k)$. $\mathcal{S}$ then sends z to $\mathcal{R}$ through a secure channel. $\mathcal{R}$ calculates its own Merkle roots of z and ϕ and compares them with the root received from $\mathcal{J}$. If they are equal, $\mathcal{R}$ sends p coins to $\mathcal{J}$ where they are locked until the last stage.

Reveal: $\mathcal{S}$ either reveals (k, d) to $\mathcal{J}$ and $\mathcal{R}$, or aborts.

Payout: If $\mathcal{S}$ reveals k and the commitment opening is satisfied, $open(k, c, d) = 1$, then $\mathcal{R}$ runs the *Extract* subroutine with inputs (z, ϕ, k). *Extract* decrypts and verifies every gate $\phi_i \in \phi$, $\forall i \in m$ by checking that $op_i(\{in_i\}) = Dec(k, z_i)$ where *Dec* is a secure symmetric key decryption, and in_i are the inputs to gate ϕ_i. If all checks pass then *Extract* outputs $(False, \perp)$ and $\mathcal{R}$ finds that $\phi(x) = 1$. If any checks fail *Extract* output $(True, \pi)$ and $\mathcal{R}$ learns that $\phi(x) \neq 1$. The variable π denotes a proof of misbehavior and contains a single gate needed to prove to $\mathcal{J}$ that $\phi(x) \neq 1$. If $Extract()$ outputs $(False, \perp)$, $\mathcal{R}$ finalizes the round and p coins are paid to $\mathcal{S}$. Otherwise $\mathcal{J}$ verifies π. If the proof is valid $\mathcal{J}$ learns that $\phi(x) \neq 1$ and returns p coins to $\mathcal{R}$.

2.4 Arithmetic circuit randomization

Protection of circuit computation against an adversary who has access to a bounded number of wires (ℓ wires) was considered in [34], and circuit randomization technique was proposed to protect the system from probing. This work considers Boolean circuits consisting of NOT and AND gates and gives a randomized transformation that replaces each gate in the circuit with an $(m + 1, m + 1)$ additive secret sharing scheme (all the shares are required for reconstruction) where $m = 2\ell$.

In FairSwap the seller represents ϕ as a circuit using one or more types of gates, that can be thought of as basic function, such as a hash function (in proof of concept implementation of FairSwap). The leakage of PoM is twice the input size of these gates, and for

Boolean circuits can reduce the leakage to 2 bits. Not all predicates can be efficiently implemented in Boolean circuits such as ones that require computations over a finite field. We extend the approach of [34] to define ℓ-private circuits, and design randomized arithmetic gates and ℓ-private arithmetic circuits. Designing randomization for specialized gates is an interesting direction for future work.

3 Security model

In this section, we formalize security and privacy of SC-aided fair exchange. We first give an overview of the model and the new definitions, and then describe the ideal functionality of the SC-aided fair exchange that captures the fairness and privacy.

3.1 UC model of the SC-aided OFE

In Canetti's UC framework, security is defined by comparing the execution of the protocol in the real-world, with the execution of its idealized version in the ideal-world. In the real-world, each party is modeled by an interactive probabilistic polynomial time Turing machine (PPT) that interacts with other parties. An adversary $\mathcal{A}$ can corrupt a set of parties. We consider a static adversary in which the parties are corrupted at the start of the protocol run.

In the ideal world, parties interact with an ideal functionality that gives an abstract specification of the required security properties. An ideal adversary, also called simulator Sim, "attacks" the ideal functionality through its interfaces.

In a UC-secure protocol there is an additional special party called the environment Z that orchestrates both worlds by providing the inputs for all parties, and receives their outputs. A protocol is said to be UC-secure if the environment Z cannot distinguish whether it is interacting with the ideal or real world. This implies that the real world is at least as secure as the ideal functionality. This is formalized (UC theorem) by showing that for every adversary $\mathcal{A}$ that attacks the real world protocol, there exists an ideal world adversary (or simulator) Sim that interacts with the ideal functionality, and outputs a view that is indistinguishable by $\mathcal{Z}$ from the one produced by $\mathcal{A}$. The UC theorem allows a modular design and construction of a *hybrid world* where a protocol makes calls to the ideal functionalities $\mathcal{F}_1, ..., \mathcal{F}_{n'}$ as its subroutines. Each ideal functionality can be replaced by a protocol which securely realizes it, while maintaining security of the protocol.

In *Generalized UC (GUC)* [17] protocols can share state through global functionalities. The important assumption in GUC is that the global ideal functionalities exist in both hybrid and ideal world and unlike UC, environment $\mathcal{Z}$ has access to them and can use them to distinguish the two worlds.

In standard (G)UC framework the adversary is monolithic and can corrupt multiple parties and coordinate their attacks. To model security against a dishonest buyer and seller, and privacy against SC, we need to model non-monolithic adversaries that have independent goals and are not willing to collude. Kamara et al. [35] motivated the need for modeling non-monolithic adversaries with different intentions in the server-aided computation setting where the computation is outsourced to a semi-honest server. Our SC-aided OFE is similar to this setting with the major difference that SC has the role of the judge and its computation must be minimized. We however need to extend this framework to model a hybrid world with global functionalities of ledger and random oracle that had been used in FairSwap, and are slightly revised for our framework.

Global ledger functionality $\mathcal{L}$ [24]. To handle coin transfer between parties and support contracts that lock coins, we need a global ledger functionality that manages the transfer of funds. We use the ledger model used by FairSwap that provides the basic properties of a cryptocurrency. The internal state of $\mathcal{L}$ is public and consists of the balances p_1, ..., p_n. $\mathcal{L}$ has three interfaces: *update* that is used by the environment to update the balance of parties $\mathcal{P}_i$; *freeze* is used to transfer p coins from one party to a contract where they are locked; *unfreeze* is used to transfer the coins from a contract to the balance of a party (in FairSwap this is accessible to ideal functionalities only). Please see Appendix A for details.

Global (restricted programmable and observable) random oracle functionality $\mathcal{H}$ [15]. The random oracle functionality $\mathcal{H}$ responds to all queries with uniformly random sampled values $r \leftarrow \{0,1\}^k$, and outputs the same value for the same query. All query-response pairs are stored in the set Q. If the query has been answered before, such that $(q,\ r) \in Q$ is stored, $\mathcal{H}$ responds with the same value r. We consider a programmable random oracle which lets the simulator Sim control the random oracle and program its hash values to specific responses using $Program$ interface. Additionally, the simulator can see all queries made by honest or corrupted parties to $\mathcal{H}$ through $Observe$ interface. Please see Appendix A for details.

Communication model. We assume a synchronous communication model where the protocol is executed in rounds and all parties are always aware of the current round through interacting with the smart contract and blockchain [36]. Smart contract keeps track of the state of the protocol and parties can go to the next round accordingly. If a party sends a message at round i it will be received by the next party at the beginning of the next round, $i+1$. Parties initially decide on the duration of each round. For simplicity, we consider instantaneous channels, although in reality communication to blockchain and smart contracts takes more time compared to interactions between the sender and receiver.

3.2 Adapting and extending the Kamara et al.'s framework

Using the framework of Kamara et al. [35], we model SC, the buyer and the seller) as *non-colluding*, *non-cooperative independent* adversaries, where SC is semi-honest, and the buyer and the seller can deviate arbitrarily from the protocol (one of the two). Independent adversaries do not share information through non-protocol messages. Non-cooperative adversaries do not share information through protocol messages. The independence of the adversaries refers to their distinct security goals.

Definition. (Non-cooperative adversary [35].) Let $\mathcal{F}$ be a deterministic n-party functionality and Π be an n- party protocol. Furthermore, let H, I_{nc}, and I_c be pairwise disjoint subsets of $[n+1]$ and let $\mathcal{A} = (\mathcal{A}_1, ..., \mathcal{A}_m)$, where $m = |I_{nc}|$, be a set of independent PPT adversaries. For any $i,j \in [m+1]$ such that $i \neq j$, we say that adversary $\mathcal{A}_j$ is non-cooperative with respect to $\mathcal{A}_i$ if there exists a PPT simulator $\nu_{i,j}$ such that for all $x \in (\{0,1\}^*)^n$ and $z \in (\{0,1\}^*)^{n+1}$, and all $y \in Ran(f_i) \cup \{\perp\}$,

$\{\nu_{i,j}(y, z_i)\}_{k\in N} \approx \{view_{i,j} | output_i = y : \{OUT_\ell\}_\ell \leftarrow Real^{(i)}_{\Pi,\mathcal{A},I_{nc},I_c,z}(k,x)\}_{k\in N}$

$view_{i,j}$ denotes the messages between $\mathcal{A}_i$ and $\mathcal{A}_j$ in the real world execution and $output_i = y$ is the event that party P_i receives output value y. $Ran(f_i)$ is the range of f_i, where f_i is the output of $\mathcal{F}$ for party i.

An adversary $\mathcal{A}_i$ is isolated if all adversaries $\mathcal{A}_{|_{j\neq i}}$ are non-cooperative with respect to $\mathcal{A}_i$. In our model, the smart contract is an isolated adversary.

To formulate security with respect to non-colluding adversaries, Kamara et al. introduced the notion of *partial emulation* for non-monolithic adversaries, where at most one party is corrupted by an adversary. Partial emulation states that "for each independent adversary $\mathcal{A}_i$, the joint distribution composed of the honest parties' output and a single adversary $\mathcal{A}_i$'s view in the real world, is indistinguishable from the joint distribution of the honest parties' outputs and the Simulator Sim_i's output in the ideal world." To specify the set of adversaries and their behaviours the notion of *adversary structure ADV* is used. For example $ADV = \{\mathcal{A}_1[sh], \mathcal{A}_2[sh], \mathcal{A}_3[nc_1, nc_2]\}$ states that $\mathcal{A}_1$ and $\mathcal{A}_2$ are semi-honest, and $\mathcal{A}_3$ is non-cooperative with respect to $\mathcal{A}_1$ and $\mathcal{A}_2$.

Extensions of the framework. We build on the framework of Kamara et al. and develop it in the following two ways. Firstly, partial emulation in Kamara et al. proof considers a single adversary. In FairSwap the protocol execution can follow two distinct path, first when buyer and seller are honest, and the second when one of the two is dishonest. In this latter case, the partial emulation must consider two adversaries and so we need to consider two simulators in the ideal world.

A second consideration is inclusion of global ideal functionalities that allow sharing of state between adversaries. This requires careful definition of interfaces to the ideal functionalities. In Appendix B, we give details of our approach to extend partial emulation to include multiple adversaries, and the formal definition of security and privacy for the SC-aided computation.

3.3 Security of SC-aided OFE

In the real world, the smart contract is an entity with access to a Ledger functionality $\mathcal{L}$ as a subroutine, and interacts with protocol parties to assist them in their computation. In the ideal world, SC is a dummy entity with no input. Each party in the protocol, except the SC, receives an input from environment and hands it to the functionality $\mathcal{F}$. $\mathcal{F}$ proceeds according to its specification and returns the result to each party, and also updates the Ledger functionality $\mathcal{L}$, if necessary. We define two simulators Sim and Sim_{sc}. Sim captures the effect of $\mathcal{A}$ attacking the protocol Π against its basic properties, and Sim_{sc} captures the effect of $\mathcal{A}_{sc}$ attacking protocol Π against its privacy. In Appendix B, we give the detailed description of the ideal/real world.

3.4 Ideal functionality for pFairSwap

Figure 1 shows the new ideal functionality $\mathcal{F}^{\mathcal{L}}_{cfe}$ following the above framework (the changes with respect to FairSwap are highlighted in red). $\mathcal{F}^{\mathcal{L}}_{cfe}$ describes a setting where the sender $\mathcal{S}$ sells a witness x to the receiver $\mathcal{R}$ and obtains p coins if the witness is correct (i.e. $\phi(x) = 1$). Transfer of coins is performed by $\mathcal{F}^{\mathcal{L}}_{cfe}$ through interacting with the Ledger ideal functionality $\mathcal{L}$. The functionality proceeds in three phases (*Initial-*

ize, *Reveal*, *Payout*). In the *Initialize* phase, $\mathcal{S}$ sends the witness x and the description of the predicate circuit ϕ to $\mathcal{F}_{cfe}^{\mathcal{L}}$. We let $\mathcal{S}$ also sends the leakage of ϕ, denoted by $le(\phi)$, to the functionality. $le(\phi)$ models the real-world leakage of pFairSwap (which is equal to the description of the corrupted gate consisting of its index, type and indexes of its inputs and outputs) in the ideal world and is $\perp$ when $\mathcal{S}$ is honest. For a malicious $\mathcal{S}$, $le(\phi)$ is non-empty and includes the description of the corrupted gate in ϕ that can be seen by everyone and is provided to $\mathcal{F}_{cfe}^{\mathcal{L}}$ by the environment.

We also let $\mathcal{F}_{cfe}^{\mathcal{L}}$ to leak the public information about the circuit and its leakage $le(\phi)$ to both simulators, Sim and Sim_{sc} that are present in the ideal world. If $\mathcal{R}$ approves then $\mathcal{F}_{cfe}^{\mathcal{L}}$ freezes p coins in the Ledger $\mathcal{L}$. In the *Reveal* phase, if a corrupted $\mathcal{S}$ aborts, $\mathcal{F}_{cfe}^{\mathcal{L}}$ unfreezes the p coins to $\mathcal{R}$ and sends a message to Sim_{sc} to inform that $\mathcal{S}$ has aborted. This was not required in FairSwap since Sim is able to see the messages of a corrupted $\mathcal{S}$. In the ideal world, Sim_{sc} is informed about the state (flow) of the protocol which are public in the real (hybrid) world. If $\mathcal{S}$ does not abort then $\mathcal{F}_{cfe}^{\mathcal{L}}$ goes to the next phase. In *Reveal* phase, if a corrupted $\mathcal{R}$ aborts then $\mathcal{F}_{cfe}^{\mathcal{L}}$ unfreezes the p coins in favor of $\mathcal{S}$ and informs Sim_{sc} that $\mathcal{R}$ has aborted. Again, note that $\mathcal{F}_{cfe}^{\mathcal{L}}$ does not need to inform Sim about $\mathcal{R}$ aborting since it sees and controls all the messages of the corrupted parties. Else if $\mathcal{R}$ does not abort, $\mathcal{F}_{cfe}^{\mathcal{L}}$ checks $\phi(x)$ and unfreezes the coins to $\mathcal{S}$ if $\phi(x) = 1$. This case happens if all parties behave honestly. Otherwise, $\mathcal{F}_{cfe}^{\mathcal{L}}$ unfreezes the coins in favor of $\mathcal{R}$ which indicates the corrupted sender has revealed an incorrect witness.

Security properties. This ideal functionality guarantees the following properties:

– Sender fairness: An honest sender $\mathcal{S}$ is guaranteed that the receiver $\mathcal{R}$ learns the witness iff they pay p coins.

– Receiver fairness: An honest receiver $\mathcal{R}$ is guaranteed that they pay p coins iff the sender delivers the correct witness in exchange.

– Smart contract privacy: SC does not learn the witness.

– Termination: If at least one party is honest, the fair exchange protocol terminates in at most 5 rounds and unlocks all coins from the contract (In round 4, if $\mathcal{F}_{cfe}^{\mathcal{L}}$ receives $(abort, id)$ from the corrupted receiver, it waits for one more round and terminates in round 5).

We now describe how $\mathcal{F}_{cfe}^{\mathcal{L}}$ captures the above properties. *Sender fairness* is ensured because if $\phi(x) = 1$, irrespective of the receiver's early abort, $\mathcal{F}_{cfe}^{\mathcal{L}}$ unfreezes the coins to the sender. *Receiver fairness* is ensured since an honest receiver will receive their coins back if the sender aborts before revealing the witness x, or when $\phi(x) = 0$. *Smart contract privacy* is obtained since Sim_{sc} is only allowed to see the public information. *Termination* is guaranteed since $\mathcal{F}_{cfe}^{\mathcal{L}}$ proceeds in 5 rounds in the worst case. Thus, any protocol that realizes this ideal functionality can achieve all these properties that we motivated our paper accordingly.

Privacy preserving coin-aided fair exchange ideal functionality $\mathcal{F}_{cfe}^{\mathcal{L}}$

The ideal functionality $\mathcal{F}_{cfe}^{\mathcal{L}}$ (in session id) interacts with a receiver $\mathcal{R}$, a sender $\mathcal{S}$, a smart contract SC (does not provide input but receives output from functionality), the ideal adversaries Sim and Sim_{sc}, and the global ledger $\mathcal{L}$. Sim can corrupt either $\mathcal{S}$ or $\mathcal{R}$. Sim_{sc} corrupts SC.

Initialize

(Round 1) Upon receiving $(sell, id, \phi, le(\phi), p, x)$ with $p \in N$ from $\mathcal{S}$, leak $(sell, id, \phi, le(\phi), p, \mathcal{S})$ to Sim and Sim_{sc}, store witness x, circuit ϕ, leakage about ϕ and price p.

(Round 2) Upon receiving (buy, id, ϕ, p) from receiver $\mathcal{R}$ in the next round, leak $(buy, id, \mathcal{R})$ to Sim and Sim_{sc}, send $(freeze, id, \mathcal{R}, p)$ and go to Reveal phase.

Reveal

(Round 3) Upon receiving $(abort, id)$ from the corrupted sender $\mathcal{S}^*$ in round 3, send $(abort, id, \mathcal{S})$ to Sim_{sc} and $(unfreeze, id, p, \mathcal{R})$ to $\mathcal{L}$ in the next round and terminate. Otherwise if do not receive such message in round 3, then send $(bought, id, x)$ to $\mathcal{R}$ and go to Payout phase.

Payout

(Round 4) Upon receiving $(abort, id)$ from the corrupted receiver $\mathcal{R}^*$, send $(abort, id, \mathcal{R})$ to Sim_{sc}, wait until round 5 to send $(sold, id)$ to $\mathcal{S}$, $(unfreeze, id, p, \mathcal{S})$ to $\mathcal{L}$ and terminate. Otherwise if no such message was received:

1. If $\phi(x) = 1$, send message $(unfreeze, id, p, \mathcal{S})$ to $\mathcal{L}$ and $(sold, id)$ to $\mathcal{S}$,
2. If $\phi(x) \neq 1$, send message $(unfreeze, id, p, \mathcal{R})$ to $\mathcal{L}$ and $(not\ sold, id)$ to $\mathcal{S}$.

Fig. 1. Ideal functionality $\mathcal{F}_{cfe}^{\mathcal{L}}$ for privacy preserving coin-aided FE

4 Enhancing privacy of FairSwap

In FairSwap, if the two parties behave honestly, the SC will only receive commitments to the key, the root of the Merkle tree constructed on the encrypted inputs and the result of the computing circuit ϕ, as well as the root of the Merkle tree constructed on the circuit ϕ; so the leakage of the protocol can be estimated from its static specification. We show that the leakage to the SC

when one of the parties is dishonest can be estimated and propose pFairSwap that realizes $\mathcal{F}_{cfe}^{\mathcal{L}}$.

4.1 Information leakage of PoM in FairSwap

Let the corrupted gate be represented as $\phi_g = (g,\ op_g,\ I_i,\ I_j,\ I_k)$, where I_i and I_j are the indices of the inputs, I_k is the index of the output, and op_g is the operation of gate ϕ_g. Note we make explicit the indices of the input gates to ϕ_g. PoM consists of the gate description ϕ_g, encrypted input and output values denoted, z_i, z_j and z_k, corresponding to the out_{I_i}, out_{I_j}, and out_{I_k}. The Judge contract $\mathcal{J}$ knows the encryption key k and hence revealing the encrypted values reveals the corresponding plaintexts. In Appendix G we give a detailed estimate of the leaked information in PoM.

We note that the leakage can be reduced to the input and output values of a single gate, i.e. to at most three wires, if the Merkle tree is constructed over the hash values of the elements of z and the circuit ϕ. The information leakage however will depend on the location of the gate and a gate right above the input layer will leak at least 2γ of the input data, where γ is the bit length of each input chunk, $\gamma \geq \lambda$, and λ is a security parameter.

4.2 ℓ-Private arithmetic circuits

Knowing the bound on the number of leaked wires allows us to use the ℓ-probing security model and use the randomized transformation on gates that was proposed in [34] to protect against the leakage. In Appendix C we extend the transformation to addition and multiplication gates and prove its security for $\mathbb{F}_p$ where p is a prime that is chosen using the security parameter. The gates use $(m+1\ , m+1)$ secret sharing of each input wires, and show how the $m+1$ wires of the output can be calculated to provide privacy of the circuit output against ℓ probed wire, $m = 2\ell$. Given the estimate of the number of leaked wires in the modified (extra hash layer) PoM, we will use $\ell = 1$ in our work. Briefly, we can use $\ell = 1$ since the revealed wires are not independent and they are all related to one gate.

Circuit randomization effectively replaces a wire in the circuit with $m+1$ wires, and divides the value $w \in \mathbb{F}_p$ that is sent over the original wire into $m+1$ shares that are sent over the $m+1$ wires such that access to up to m wires does not give any information about w.

4.3 Our scheme

We assume the predicate is described as an arithmetic circuit over $\mathbb{F}_p$. The sender uses the ℓ-private circuit randomization for $\ell = 1$, that needs randomness for each gate to expand circuit ϕ to ϕ'. The seed for the PRG must be generated by a secure two-party computation involving $\mathcal{S}$ and $\mathcal{R}$. The seed will be private to the SC and can be generated by the two parties using a secure key agreement protocol. The sender must include this randomness as part of their committed values. Therefore, using circuit expansion requires us to change the initialization phase (i.e. circuit expansion, commitments and encoding the inputs and circuit computation), and the payout phase (i.e. extracting the input). We divide our scheme into three phases (see Appendix D).

Initialization phase: The sender $\mathcal{S}$ first expands ϕ to ϕ' with subroutine *Expand* (the psuedocode is given in Section 6). The subroutine replaces gates in the original circuit with gadgets that perform the operation of the gate, while replacing each input and output wire of the gate with a corresponding set of $m+1$ wires such that the sum of the inputs to the $m+1$ wires is the same as the value of the corresponding wire. In addition to multiplication and addition gates, a multiplication gadget includes single-input identity gates with randomly chosen inputs. Then, $\mathcal{S}$ encodes the input $x = \{x_1,\ ..., x_n\}$ with subroutine *InputEncode* to generate the corresponding secret shares for each x_i. We denote the encoded input with x' (which is used as the input of the expanded circuit ϕ'). Afterwards, $\mathcal{S}$ samples a key k and commits to it with $(c,\ d) = commit(k)$. Then, $\mathcal{S}$ generates n_{rand} number of random values $\rho_1,\ ...,\ \rho_{n_{rand}}$ by applying a PRG on the shared seed for the identity gates with random inputs (within each multiplication gadget) in ϕ' (the required number of randomness are given in Appendix C) and stores $(i,\ \rho_1,\ ...,\ \rho_{n_{rand}})$ locally where i is the index of the corresponding multiplication gate in ϕ (obtained by linearizing the circuit ϕ).

$\mathcal{S}$ runs the modified *Encode* subroutine on the expanded circuit ϕ', the encoded inputs x', random values ρ', and key k. *Encode* generates the encoded inputs. It also computes each gate (within the gadgets) in the circuit ϕ' and encrypts their outputs. *Encode* outputs all the encrypted values denoted by z. $\mathcal{S}$ sends z to $\mathcal{R}$ through the secure channel. $\mathcal{S}$ then sends the following commitments to SC: the commitment c for key k, the root r_z of the Merkle tree constructed on z, the root $r_{\phi'}$ of the Merkle tree constructed on the circuit ϕ', and the root $r_{\rho'}$ of the Merkle tree constructed on the random values ρ'.

$\mathcal{R}$ expands the circuit ϕ using *Expand* and constructs a Merkle tree on ϕ' to get its Merkle root $r_{\phi'}$. $\mathcal{R}$ generates n_{rand} random values $\rho_1,\ ...,\ \rho_{n_{rand}}$ us-

ing a PRG on the shared seed for the identity gates which take random inputs (within the multiplication gadgets) in ϕ', and computes the Merkle root on $\rho' = \{(i,\ \rho_1,\ ...,\ \rho_{n_{rand}})\}$. $\mathcal{R}$ then constructs a Merkle tree on the z they received from $\mathcal{S}$ to get Merkle root r_z. $\mathcal{R}$ checks that these Merkle roots are the same as the ones received by SC. If the Merkle roots are the same, $\mathcal{R}$ sends a *buy* message to SC, which in turn locks p coins from $\mathcal{R}$ in the ledger, and protocol proceeds to *reveal* phase. Otherwise, $\mathcal{R}$ aborts.

Reveal phase: In this phase, $\mathcal{S}$ reveals the key k and opening value d. $\mathcal{R}$ and SC checks $open(k,\ c,\ d)$. If the opening is not correct or $\mathcal{S}$ aborts, then SC transfers the p coins to $\mathcal{R}$ through the ledger and terminates. Otherwise, they proceed to the **Payout** phase.

Payout phase: In this phase, $\mathcal{R}$ extracts the original input x using $Extract$ on the expanded circuit ϕ', the encrypted values z and the key k. If $Extract$ outputs $(False,\ \perp)$, $\mathcal{R}$ finalizes the exchange and SC transfers p coins to $\mathcal{S}$. In this case the exchange is successful. Otherwise, if $Extract$ returns $(True,\ \pi)$, then $\mathcal{R}$ complains about $\mathcal{S}$ by sending π to SC. SC verifies the complaint using the $Judge$ subroutine (given in Appendix C and the same as $Judge$ in FairSwap). If the computation of a single gate is incorrect, $Judge$ checks the operation of the gate on the inputs received from receiver (that has already been committed to by sender) and compares the obtained result with the output claimed by the sender. Note that as we use circuit randomization, the input wires contain two random values so they will not reveal any information about input even if the corrupted gate is near input layer. If π is a valid proof of misbehavior, then SC refunds p coins to $\mathcal{R}$ and terminates the exchange with *not sold* state. If π is not valid, SC transfers p coins to $\mathcal{S}$ and terminates the exchange in *sold* state. Additionally, if $\mathcal{R}$ aborts and does not send any message at this phase $\mathcal{S}$ can finalize the exchange. In Appendix C we give details of modified Encode and Extract algorithms of pFairSwap.

5 Security analysis

We give proof sketches for the required security properties in section 3 (please see Appendix F for details).

– *Sender fairness:* To guarantee this property, the sender should receive p coins when they reveal the item x to the receiver. In our scheme, item x is encrypted using key k and the key is committed in SC. From the IND-CPA security of the encryption scheme and hiding property of the commitment scheme, receiver cannot learn anything about the item x with non-negligible probability before SC locks p coins. As SC is semi-honest, if sender reveals the key and receiver cannot make a valid complaint, SC will pay p coins to the sender. Because of the ℓ-privacy of the circuit, i.e. a single gate can be revealed without compromising the privacy, even if receiver makes a fraudulent complaint to the SC, only random values are leaked to the SC.

– *Receiver fairness:* We need to show that sender cannot provide an incorrect item $x' \neq x$ such that a valid PoM cannot be generated. Sender can only succeed if they can generate an encoding such that $Extract(\phi',\ z,\ k) = (x', \pi')$, where ϕ' is the expanded circuit, and π' will not be accepted by SC. This can only happen if the sender breaks the binding property of the commitment scheme or the collision resistant property of the hash function.

– *Smart contract privacy:* Regardless of the behavior of the malicious sender and receiver, SC only observes three random values since the expanded circuit satisfies ℓ-privacy.

– *Termination:* In the worst case if SC receives the $finalize$ message from receiver in round 5, it transfers the coins and terminates (takes at most 5 rounds).

Theorem 1. *For any PPT adversaries $\mathcal{A}_S$, $\mathcal{A}_R$ and $\mathcal{A}_{sc}$, corresponding to the sender, receiver and smart contract, pFairSwap (ADV)-securely and privately realizes $\mathcal{F}_{cfe}^{\mathcal{L}}$ where security and privacy are according to Definition 1, where $ADV = \{\{\mathcal{A}_S[m]\},\ \{\mathcal{A}_R[m]\},\ \{\mathcal{A}_{sc}[sh]\},$ $\{\mathcal{A}_S[nc],\ \mathcal{A}_{sc}[sh]\},\ \{\mathcal{A}_R[nc],\ \mathcal{A}_{sc}[sh]\}\}$*

Proof sketch. We prove two properties, security (through defining the simulator Sim) and privacy against public (SC) (through defining the simulator Sim_{sc}). FairSwap only proves the security.

We assume Sim chooses a random value that will be used as the seed by the sender and the receiver. This simulates the seed generation at the start of the protocol. To prove privacy we show that the view of SC that consists of all messages that are sent by $\mathcal{S}$ and $\mathcal{R}$ to the SC can be simulated by Sim_{sc} using public information that are available in the ideal world. The main difference between the two simulators, Sim_{sc} and Sim, is that Sim knows the item. For example when Sim simulates a corrupted $\mathcal{S}$, it can use private information of the seller including the secret key k (by extracting k from the key commitment). This will allow Sim to decrypt the ciphertexts of the corrupted seller and obtains the item and all intermediate computation results. However Sim_{sc}, for this case, must simulate the PoM using

The smart contract SC acts as a judge contract for session id id and interacts with the global $\mathcal{L}$ functionality and the parties $\mathcal{S}$ and $\mathcal{R}$. It locally stores addresses $Pk_{\mathcal{S}}$ and $Pk_{\mathcal{R}}$, price p, commitment c, decryption key k, Merkle tree root hashes r_z, $r_{\phi'}$, $r_{\rho'}$, and state s.

Initialize

(Round 1) Upon receiving $(init, id, p, c, r_{\phi'}, r_{\rho'}, r_z)$ from $\mathcal{S}$, with $p \in N$, store $r_{\phi'}$, r_z, p, c, output $(initialized, id, p, r_{\phi'}, r_z, c)$, set $s = initialized$ and proceed to the *reveal* phase.

(Round 2) Upon receiving $(accept, id)$ from $\mathcal{R}$ when $s = initialized$, send $(freeze, id, \mathcal{R}, p)$ to $\mathcal{L}$. If it responds with $(frozen, id, \mathcal{R}, p)$, set $s = active$, and output $(active, id)$.

Reveal

(Round 3) Upon receiving $(reveal, id, d, k)$ from sender $\mathcal{S}$ when $s = active$ and $Open(c, d, k) = 1$, send $(revealed, id, d, k)$ to all parties and set $s = revealed$. Then proceed to *payout* phase. Otherwise, if no such message from $\mathcal{S}$ was received, send message $(unfreeze, id, p, \mathcal{R})$ to $\mathcal{L}$ and abort.

Payout

(Round 4) Upon receiving a message m from the receiver $\mathcal{R}$ when $s = revealed$, set $s = finalized$ and do the following:

- If $m = (complain, id, \pi)$ s.t. $Judge(k, r_z, r_{\phi'}, \pi) = 1$ send $(unfreeze, id, p, \mathcal{R})$ to $\mathcal{L}$, $(not\ sold, id)$ to $\mathcal{S}$ and terminate.
- Otherwise, send $(unfreeze, id, p, \mathcal{S})$ to $\mathcal{L}$,$(sold, id)$ to $\mathcal{S}$ and terminate.

(Round 5) Upon receiving message $(finalized, id)$ from the sender $\mathcal{S}$, when $s = revealed$, send message $(unfreeze, id, p, \mathcal{S})$ to $\mathcal{L}$. Then output $(Sold, id)$ to $\mathcal{S}$ and terminate.

Fig. 2. Description of the protocol for smart contract SC

only the public gate description provided in $le(\phi)$ and without knowing the inputs.

The analysis of the scheme for security with respect to adversary $\mathcal{A} = \{\mathcal{A}_{\mathcal{S}}, \mathcal{A}_{\mathcal{R}}\}$ is similar to FairSwap. In the ideal world, $\mathcal{F}_{cfe}^{\mathcal{L}}$, with respect to Sim, is the same as $\mathcal{F}_{cfe}^{\mathcal{L}}$ in FairSwap, and in the hybrid world, our initialization stage is mainly different from FairSwap where the circuit is expanded and the details of algorithms $Encode$ and $Extract$. These modifications will not affect the simulation. Please see Appendix F for details.

Simulation with respect to $\mathcal{A}_{sc}$. Sim_{sc} chooses random values for the key k and ciphertexts z', and ρ'. Sim_{sc} sends the Merkle roots r_z, $r_{\rho'}$ and $r_{\phi'}$ to SC on behalf of the sender. In the second round, Sim_{sc} opens the commitment to the key k and when $\mathcal{F}_{cfe}^{\mathcal{L}}$ updates

The protocol consists of descriptions of the behavior of the honest sender $\mathcal{S}$ and receiver $\mathcal{R}$ in exchanging item x. $\mathcal{S}$ and $\mathcal{R}$ generate a shared seed s_0.

Initialize

$\mathcal{S}$: Upon receiving input $(sell, id, \phi, le(\phi), p, x)$ in round 1, assuming that property ϕ has n_{Mult} number of multiplication gates, $\mathcal{S}$ applies PRG on the seed s_0 and generates n_{rand} number of random values $\rho = \{\rho_1, ..., \rho_{n_{rand}}\}$ for each multiplication gate, where $n_{rand} = \frac{m(m+1)}{2}$, $m = 2\ell$, and ℓ is the number of wires that can be revealed (this information is provided in $le(\phi)$). Lets consider that $\rho' = \{(i, \rho_1, ..., \rho_{n_{rand}})\}$ incorporates the list of all random values for each multiplication gate i. Then $\mathcal{S}$ samples $k \leftarrow Gen(1^k)$, computes $(c, d) \leftarrow Commit(k)$, $\phi' = Expand(\phi, \ell)$, $x' = InputEncode(x, \ell, \lambda)$, λ is the security parameter, and $z = Encode(\phi', x', \rho', k)$. Then he sends $(sell, id, z, \phi, c)$ to $\mathcal{R}$ and $(init, id, p, c, r_{\phi'}, r_{\rho'}, r_z)$ to SC, where $r_{\phi'} = root(Mtree(\phi'))$, $r_{\rho'} = root(Mtree(\rho'))$ and $r_z = root(Mtree(z))$. Then he continues to the *reveal* phase.

$\mathcal{R}$: Upon receiving input (buy, id, ϕ), $\mathcal{R}$ checks if he received message $(sell, id, z, \phi, c)$ from $\mathcal{S}$ in round 1 and computes $\phi' = Expand(\phi, \ell)$, generates the random values rho', computes $r_{\rho'} = root(Mtree(\rho'))$, $r_z = root(Mtree(z))$ and $r_{\phi'} = root(Mtree(\phi'))$. Upon receiving $(init, id, p, c, r_{\phi'}, r_{\rho'}, r_z)$ from SC, $\mathcal{R}$ responds with $(accept, id)$ and proceeds to the *reveal* stage.

Reveal

$\mathcal{S}$: Upon receiving $(active, id)$ from SC, $\mathcal{S}$ responds with $(reveal, id, d, k)$ and proceeds to the *payout* phase. If no $(active, id)$ message was received from SC in the third round, he instead terminates the protocol.

$\mathcal{R}$: Upon receiving $(revealed, id, d, k)$ from SC, $\mathcal{R}$ proceeds to *payout* phase. Otherwise, if no $(revealed, id, d, k)$ message was received from SC in round 4, $\mathcal{R}$ terminates the protocol.

Payout

$\mathcal{R}$: The receiver runs $(x, \pi) = Extract(\phi', z, k)$. If $\pi = \perp$, he sends message $(finalize, id)$ to SC, otherwise he sends $(complain, id, \pi)$ instead. Then he outputs $(bought, id, x)$ and terminates the protocol execution.

$\mathcal{S}$: Upon receiving $(Sold, id)$ or $(not\ sold, id)$ from SC, $\mathcal{S}$ outputs this message and terminates the protocol. If no message has been received in round 4, he sends $(finalize, id)$ to SC.

Fig. 3. Formal protocol description for honest $\mathcal{S}$ and $\mathcal{R}$

the ledger $\mathcal{L}$, terminates. $\mathcal{Z}$ cannot distinguish the simulation from the real world execution since the only way to distinguish is to learn z'. The randomized Merkle tree commitment scheme is information hiding, so the probability to learn z' is negligible.

Simulation with respect to $\mathcal{A}_S$ and $\mathcal{A}_{sc}$. Sim_{sc} samples a random key k and chooses random ciphertexts z', and randomness ρ', constructs a commitment on key k, and Merkle roots r_z, $r_{\rho'}$ and $r_{\phi'}$. Then in the next round if sender does not abort (in which case Sim_{sc} will be notified through $\mathcal{F}_{cfe}^{\mathcal{L}}$), Sim_{sc} opens the commitment to key k. As $\mathcal{Z}$ cannot observe z' and the randomized Merkle tree commitment is hiding, it cannot distinguish the simulated Merkle roots from the real world values. In the last round, Sim_{sc} checks the $\mathcal{L}$ to learn the transfer of funds and infer whether the sender has provided an incorrect x' such that $\phi(x') \neq 1$ or not. If so, Sim_{sc} simulates a valid complaint about one of the gates which has been specified by $\mathcal{Z}$ in $le(\phi)$ and has been shared with Sim_{sc} through $\mathcal{F}_{cfe}^{\mathcal{L}}$. Sim_{sc} uses random values for the input and output of the specified gates and program $\mathcal{H}$ to reveal specific values in the set z' when queried by $\mathcal{Z}$. Since the expanded circuit provides ℓ-privacy (please see the Lemma in Appendix C), $\mathcal{Z}$ cannot distinguish the real values from random ones. Additionally, Sim_{sc} can successfully program $\mathcal{H}$, except with negligible probability, so $\mathcal{Z}$ can distinguish the real and the ideal world execution only with a negligible probability.

Simulation with respect to $\mathcal{A}_R$ and $\mathcal{A}_{sc}$. Sim_{sc} samples a random key k, computes a commitment to k, and outputs the commitment and three random values as r_z, $r_{\rho'}$ and $r_{\phi'}$. If receiver aborts, Sim_{sc} will be notified by $\mathcal{F}_{cfe}^{\mathcal{L}}$ and terminates the simulation. Otherwise, Sim_{sc} checks $le(\phi)$; $le(\phi)$ is provided by environment in the initialization stage and given to Sim_{sc} by $\mathcal{F}_{cfe}^{\mathcal{L}}$. If $le(\phi) = \perp$ then Sim_{sc} makes an invalid complaint for the specified gate in $le(\phi)$. For that Sim_{sc} has to choose random values as the inputs and outputs of the gates, and a random proof π. The view of $\mathcal{Z}$ is indistinguishable from the real world execution since the complaint does not reveal anything about the item x due to the ℓ-privacy of the circuit expansion.

6 Implementation

We evaluate the overhead of circuit expansion by implementing FairSwap and pFairSwap where ϕ is a universal hash function (defined in Section 2.2 and implemented as shown in Algorithm 1) and x is a text file. Our goal is to compare the computation cost of our privacy enhancement with original FairSwap. We implement FairSwap as a set of subroutines {*Encode*, *Extract*, *Judge*}, and pFairSwap as a set of {*Encode*, *Extract*, *Judge*,*InputEncode*, *Expand*} subroutines. All subroutines are written in Python except for *Judge* which is implemented as a smart contract written in Solidity. Our measurements are divided into off-chain and on-chain measurements.

Off-chain (OFF) computation measurements: As in FairSwap, we only consider $\mathcal{S}$ for our off-chain measurements because they do similar work to $\mathcal{R}$ ($\mathcal{R}$ does some extra equality checks). For FairSwap, we measure how long it takes Pinocchio to execute ϕ with input x and *Encode* to encrypt the wires. For pFairSwap, we do the same but first measure the time it takes to randomize the circuit and inputs. Measurements are taken with the Python time module and are in seconds (s). We use the Pinocchio compiler [43] available on GitHub to generate and run arithmetic circuits. We expect the size of a circuit to have the greatest effect on the runtime of $\mathcal{S}$.

On-chain (ON) computation measurements: In the on-chain environment we run *Judge* (given in Appendix C.2.1) with the browser-based Remix IDE and measure the Gas cost to verify the PoM from FairSwap and pFairSwap. We only measure the computation cost of the subroutine and not the communication cost. Ethereum smart contracts execute in the Ethereum Virtual Machine (EVM) where each machine operation has a Gas cost. In practice, smart contracts are executed on diverse machines in a distributed network. Gas cost provides a standardized unit of computational work done by the EVM on each machine. The Gas cost is paid for in Ether, Ethereum's currency, and this means computational work on-chain has a direct financial cost. Miners in the network set an Ether price for a unit of Gas and so this cost fluctuates overtime. We expect that the size of circuits and the work to perform finite field operations will have a large impact on these measurements.

Arithmetic circuits: Our circuits compute a universal hash function that is efficient to implement as an arithmetic circuit while provably providing collision freeness if the key is chosen independent of the input (see Section 2.2). To construct this hash function, we follow Merkle's construction for hashing arbitrarily large files [41] which is shown in Algorithm 1. We construct a subroutine that hashes a fixed number of message blocks. We loop this function using the result from the previous iteration as the first block in the next iteration.

Circuit parameters: Larger λ means we can hash the file in fewer loops and reduce the circuit size, but also means larger modular reductions to compute each

Algorithm 1 Hash: Universal hash function

Input x
Pad x so that its length is multiple of $v-1$ & set $result = 0$
Set $n_t = \frac{len(x)}{v-1}$
while $n_t > 0$ **do**
    $result = result * r_1 + x_i * r_2 + ... + x_{i+v-1} * r_v \mod P$
    $n_t = n_t - 1$
Output $result$

gate. We expect the size of the circuit to have the greatest influence of the off- and on-chain measurements. For our experiment, we create 3 different circuits with prime fields $p \in \{2^{127} - 1,\ 2^{107} - 1,\ 2^{89} - 1\}$. The λ for each circuit is $\{15,\ 13,\ 11\}$ bytes respectively computed as $\frac{\log_2 p}{8}$.

6.1 Details and algorithms

Here we outline our implementation's flow. We present simplified algorithms that highlight the computation steps relevant to the measurements we took. For more details see Appendix C.2 and our GitHub repository [1].

Let x denote a text file divided into blocks (chunks), with each block represented as an index and value pair where index is the integer i of x_i for $i \in [n]$, and value is the hexadecimal representation of x_i. Circuits. ϕ, are generated and executed on input x with Pinocchio. Pinocchio outputs a file, *wires*, with the index and output value of every gate in ϕ.

EncodeInputs (Algorithm 2 used in pFairSwap only) creates an encoding of the input where each x_i is mapped to a tuple of three secret shares using pseudo-random values from the operating system to generate the shares. It outputs a file x'.

Expand (Algorithm 3 used in pFairSwap only) takes a file description of ϕ, generates a gadget for each gate and builds an expanded circuit in a new file called ϕ'. Input gadgets replace each input gate in the original circuit with m input gates.

$Encode_{simp}$ (Algorithm 4 used in FairSwap and pFairSwap) takes as input a key k and text file *wires*. We use hash-based encryption/decryption algorithms given in FairSwap implemented with the *solidity*Sha3 function from the web3 Python package. $Encode_{simp}$ outputs a text file called z. The Merkle roots r_z and $r_{\phi'}$, and merkle proofs are created with a straightforward implementation of a *MerkleTree* object in Python.

Extract (pseudo-code ommited for space, see *Extract* in Appendix C.2.1, used in FairSwap and pFairSwap) generates PoMs that can be validated by *Judge*. In our implementation *Extract* takes two files, the encrypted wire file z from $\mathcal{S}$ and plaintext wire file $wires_{\mathcal{R}}$ from $\mathcal{R}$. It decrypts z and compares the result with the corresponding values in $wires_{\mathcal{R}}$. We edit a wire value in z to corrupt the gate output. *Extract* finds the corrupted value and creates the PoM.

To run $Encode_{simp}$ and *Extract* in pFairSwap we just give circuits and inputs that have been randomized with *InputEncode* and *Expand* as inputs.

Judge is an Ethereum Smart contract (See *Judge* in Appendix C.2.1, used in FairSwap and pFairSwap) It validates the Merkle proofs, decrypts the encrypted input and output wire values, and executes the gate operation on the plaintext input wires. It finally compares its computed gate operation with the plaintext output wire value. If they are equal, it outputs true, otherwise false. we implement *Judge* in a straightforward way so omit the pseudo-code for brevity.

Algorithm 2 *EncodeInputs*: Used to encode the input file x for pFairSwap

Input (x,p) ▷ *x is input file, p is finite field*
for $x_i \in x$ **do**
    $share1 \leftarrow pseudo - rand(p)$
    $share2 \leftarrow pseudo - rand(p)$
    $share3 = x_i - (share1 + share2) \mod p$
    $x_i = (share1,\ share2,\ share3)$
Output $x' = (x'_0, ..., x'_n)$

Algorithm 3 *Expand*: Expands and randomizes circuit generated by Pinocchio for pFairSwap

Input (ϕ)
for $\phi_i \in \phi$ **do**
    Parse $\phi_i = (i,\ op_i,\ I_i)$
    if $op_i =$ `input` **then**
        $op'_i = Generate($`input gadget`$)$
    else if $op_i =$ `add` **then**
        $op'_i = Generate($`addition gadget`$)$
    else if $op_i =$ `mul` **then**
        $op'_i = Generate($`multiplication gadget`$)$
Output $\phi' = (\phi'_0, ..., \phi'_m)$ ▷ *Gadgets connected by wires according to ϕ*

1 The source code for our implementation can be found at https://github.com/Prezzy/Private-FairSwap

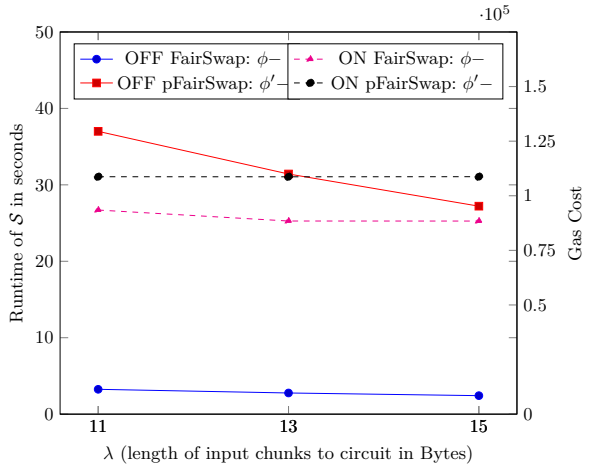

Fig. 4. Left Axis: Off-chain (OFF) measurements for runtime in seconds of $\mathcal{S}$ to execute a circuit and encrypt the intermediate values with $Encode$ algorithm. pFairSwap includes $EncodeInput$ and $Expand$ runtimes as well. **Right Axis:** On-Chain (ON) measurements of Gas Cost for $Judge$ to verify PoM.

Algorithm 4 $Encode_{simp}$: Encrypts the intermediary gate values.

1: **Input** $wires$ ▷*wires generated by Pinocchio*
2: **for** $w_i \in wires$ **do**
3: $z_i = Enc(k,\ i,\ w_i)$
4: **Output** $z = (z_1, ..., z_m)$

6.2 Benchmarks

We use a 66KB UTF-8 encoding of "The Raven" by Edgar Allan Poe [1] as our x. Our off-chain environment is a 6 Core 3.6 GHz AMD Ryzen 5 3600 Processor with 16 GB of RAM. Our 3 circuits have different values of λ in FairSwap and pFairSwap. The increase in number of gates and execution time from FairSwap to pFairSwap is linear by a factor of ≈ 10. These results indicate that larger values of λ at the cost of larger modular reductions is preferable for both FairSwap and pFairSwap, but is especially significant for pFairSwap.

For on-chain measurements, we expect the size of circuits and the size of the modular reduction increase the Gas cost of $Judge$ to verify the PoM. The results are shown in figure 4. Hashing operations in the decryption and Merkle proof verification algorithms account for a significant amount of Gas cost. For this reason, the growth in gas cost is logarithmic based on the length of Merkle proofs which are logarithmic to the circuit size. The gas cost to verify PoM's from pFairSwap is high but grows slowly as the circuit size increases.

7 Concluding remarks

OFE is a key cryptographic primitive, and implementation of the TTP by a SC appears a natural and attractive application of SCs, in particular by integrating automated coin transfer. Interaction with smart contract and its computation however will be non-private. The relation between fairness and privacy suggests that implementing TTP of an OFE protocol by an SC needs careful evaluation. In the following we outline a number of direct extensions of our work:

Optimizing Private FairSwap: pFairSwap can be made more efficient for the buyer by avoiding decryption and verification of every value in z. Please see Appendix H.

Making OptiSwap Private: Dispute resolution phase in OptiSwap [25] is an interactive challenge and response protocol and leaks a chain of gates that can be as long as the depth of the circuit. Our circuit randomization approach can be directly applied to OptiSwap. Capturing security of the interactive dispute resolution requires careful extension of our security model.

Optimizing circuit description: FairSwap's implementation of file hashing uses a circuit whose gate operation is the keccak256 hash. Our implementation is general and uses arithmetic circuit to describe any predicate. This, although attractive because of its generality and possibility of applying circuit randomization, requires significant effort for circuit construction of complex predicates such as one-way hash functions like keccak256.

8 Acknowledgements

This work is in part supported by Natural Sciences and Engineering Research Council of Canada Discovery Grant Program, and Alberta Innovates Strategic Chair in Information Security.

References

[1] Allan Poe, E.: The raven. http://www.gutenberg.org/cache/epub/17192/pg17192.txt (1845)
[2] Alper, H.K., Küpçü, A.: Optimally efficient multi-party fair exchange and fair secure multi-party computation. In: Cryptographers Track at the RSA Conference. pp. 330–349. Springer (2015)
[3] Alper, H.K., Küpçü, A.: Coin-based multi-party fair exchange. In: International Conference on Applied Cryptography and Network Security. pp. 130–160. Springer (2021)
[4] Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 443–458. IEEE (2014)

[5] Asokan, N., Schunter, M., Waidner, M.: Optimistic protocols for multi-party fair exchange (1996)

[6] Asokan, N., Shoup, V., Waidner, M.: Asynchronous protocols for optimistic fair exchange. In: Proceedings. 1998 IEEE Symposium on Security and Privacy (Cat. No. 98CB36186). pp. 86–99. IEEE (1998)

[7] Asokan, N., Shoup, V., Waidner, M.: Optimistic fair exchange of digital signatures. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 591–606. Springer (1998)

[8] Avoine, G., Gärtner, F., Guerraoui, R., Vukolić, M.: Gracefully degrading fair exchange with security modules. In: European Dependable Computing Conference. pp. 55–71. Springer (2005)

[9] Avoine, G., Vaudenay, S.: Fair exchange with guardian angels. In: International Workshop on Information Security Applications. pp. 188–202. Springer (2003)

[10] Avoine, G., Vaudenay, S.: Optimistic fair exchange based on publicly verifiable secret sharing. In: Australasian Conference on Information Security and Privacy. pp. 74–85. Springer (2004)

[11] Ben-Or, M., Goldreich, O., Micali, S., Rivest, R.L.: A fair protocol for signing contracts. IEEE Transactions on Information Theory **36**(1), 40–46 (1990)

[12] Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: Annual Cryptology Conference. pp. 421–439. Springer (2014)

[13] Brickell, E.F., Chaum, D., Damg$\mathcal{R}$ard, I.B., van de Graaf, J.: Gradual and verifiable release of a secret. In: Conference on the Theory and Application of Cryptographic Techniques. pp. 156–166. Springer (1987)

[14] Cachin, C., Camenisch, J.: Optimistic fair secure computation. In: Annual International Cryptology Conference. pp. 93–111. Springer (2000)

[15] Camenisch, J., Drijvers, M., Gagliardoni, T., Lehmann, A., Neven, G.: The wonderful world of global random oracles. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 280–312. Springer (2018)

[16] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science. pp. 136–145. IEEE (2001)

[17] Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: Theory of Cryptography Conference. pp. 61–85. Springer (2007)

[18] Carter, J.L., Wegman, M.N.: Universal classes of hash functions. Journal of computer and system sciences **18**(2), 143–154 (1979)

[19] Choudhuri, A.R., Green, M., Jain, A., Kaptchuk, G., Miers, I.: Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 719–728 (2017)

[20] Cleve, R.: Limits on the security of coin flips when half the processors are faulty. In: Proceedings of the eighteenth annual ACM symposium on Theory of computing. pp. 364–369 (1986)

[21] Cleve, R.: Controlled gradual disclosure schemes for random bits and their applications. In: Conference on the Theory and Application of Cryptology. pp. 573–588. Springer (1989)

[22] Cummings, R., Gupta, V., Kimpara, D., Morgenstern, J.: On the compatibility of privacy and fairness. In: Adjunct Publication of the 27th Conference on User Modeling, Adaptation and Personalization. pp. 309–315 (2019)

[23] Dwork, C., Hardt, M., Pitassi, T., Reingold, O., Zemel, R.: Fairness through awareness. In: Proceedings of the 3rd innovations in theoretical computer science conference. pp. 214–226 (2012)

[24] Dziembowski, S., Eckey, L., Faust, S.: Fairswap: How to fairly exchange digital goods. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 967–984. ACM (2018)

[25] Eckey, L., Faust, S., Schlosser, B.: Optiswap: Fast optimistic fair exchange. In: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security. pp. 543–557 (2020)

[26] Ekstrand, M.D., Joshaghani, R., Mehrpouyan, H.: Privacy for all: Ensuring fair and equitable privacy protections. In: Conference on Fairness, Accountability and Transparency. pp. 35–47 (2018)

[27] Garay, J.A., Jakobsson, M., MacKenzie, P.: Abuse-free optimistic contract signing. In: Annual International Cryptology Conference. pp. 449–466. Springer (1999)

[28] Garbinato, B., Rickebusch, I.: A modular solution to fair exchange for peer-to-peer middleware. In: workshop in Sintra. p. 51 (2006)

[29] Garbinato, B., Rickebusch, I.: Impossibility results on fair exchange. 10th International Conferenceon Innovative Internet Community Systems (I2CS)–Jubilee Edition 2010– (2010)

[30] Garbinato, B., Rickebusch, I.: Secure multiparty computation vs. fair exchange-bridging the gap. In: SAFECOMP 2013-Workshop ASCoMS (Architecting Safety in Collaborative Mobile Systems) of the 32nd International Conference on Computer Safety, Reliability and Security. Matthieu Roy (2013)

[31] Hearn, M.: msr-vc/pinocchio. https://github.com/corda/msr-vc/tree/master/pinocchio (2017)

[32] Huang, Q., Wong, D.S., Susilo, W.: P 2 ofe: Privacy-preserving optimistic fair exchange of digital signatures. In: Cryptographers' Track at the RSA Conference. pp. 367–384. Springer (2014)

[33] Huang, Q., Yang, G., Wong, D.S., Susilo, W.: Ambiguous optimistic fair exchange. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 74–89. Springer (2008)

[34] Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Annual International Cryptology Conference. pp. 463–481. Springer (2003)

[35] Kamara, S., Mohassel, P., Raykova, M.: Outsourcing multiparty computation. IACR Cryptol. Eprint Arch. **2011**, 272 (2011)

[36] Kiayias, A., Zhou, H.S., Zikas, V.: Fair and robust multiparty computation using a global transaction ledger. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 705–734. Springer (2016)

[37] Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE symposium on

security and privacy (SP). pp. 839–858. IEEE (2016)
[38] Kumaresan, R., Bentov, I.: Amortizing secure computation with penalties. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security. pp. 418–429 (2016)
[39] Kumaresan, R., Vaikuntanathan, V., Vasudevan, P.N.: Improvements to secure computation with penalties. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security. pp. 406–417 (2016)
[40] Liu, J., Li, W., Karame, G.O., Asokan, N.: Toward fairness of cryptocurrency payments. IEEE Security & Privacy **16**(3), 81–89 (2018)
[41] Merkle, R.C.: One way hash functions and des. In: Brassard, G. (ed.) Advances in Cryptology — CRYPTO' 89 Proceedings. pp. 428–446. Springer New York, New York, NY (1990)
[42] Pagnia, H., Gärtner, F.C.: On the impossibility of fair exchange without a trusted third party. Tech. rep., Technical Report TUD-BS-1999-02, Darmstadt University of Technology (1999)
[43] Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy. pp. 238–252. IEEE (2013)
[44] Zhao, Y., Qin, Z.g.: An optimistic protocol for distributed fair exchange. In: 2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. pp. 395–399. IEEE (2012)

A Global ideal functionalities

Figure 5 shows the global ledger functionality $\mathcal{L}$ and Figure 6 shows the global random oracle functionality $\mathcal{H}$.

Functionality $\mathcal{L}$

Shared functionality $\mathcal{L}$ is globally available to all participants P_1, ..., P_n, SC, and the ideal functionalities. It stores balances p_1, ..., p_n where $p_i \in N$ for every party $P_i \in [n]$. It also stores a partial function L for frozen cash.

Update funds

– Upon receiving $(update,\ P_i,\ p)$ with $p \geq 0$ from $\mathcal{Z}$ set $p_i = p$ and send $(updated,\ P_i,\ p)$ to every entity.

Freeze funds

– Upon receiving message $(freeze,\ Id,\ P_i,\ p)$ from party SC or an ideal functionality of session Id check if $p_i > p$. If this is not the case, reply with $(nofunds,\ P_i,\ p)$. Otherwise, set $p_i = p_i - p$, store $(id,\ p)$ in L and send $(frozen,\ Id,\ P_i,\ p)$ to every entity.

Unfreeze funds

– Upon receiving message $(unfreeze,\ Id,\ P_j,\ p)$ from party SC or an ideal functionality of session Id, check if $(Id,\ p') \in L$ with $p' \geq p$. If this check holds update $(Id,\ p')$ to $(Id,\ p'-p)$, set $p_j = p_j + p$ and send $(unfrozen,\ Id,\ P_j,\ p)$ to every entity.

Fig. 5. Global Ledger functionality $\mathcal{L}$, provides three interfaces *update funds*, *Freeze funds*, and Unfreeze funds to lock and manage transfer of coins among parties.

Functionality $\mathcal{H}$

Shared functionality $\mathcal{H}$ is globally available to all participants. It takes as input queries $q \in \{0,1\}^*$ and outputs values $r \in \{0,1\}^k$. Internally, it stores initially empty sets Q, P, and a set Q_{Id} for all sessions Id.

Query

– Upon receiving $(query,\ Id,\ q)$ from a party of session Id' proceed as follows:
– If $(Id,\ q,\ r) \in Q$ respond with $(query,\ q,\ r)$.
– If $(Id,\ q,\ r) \notin Q$, samples $r \in \{0,1\}^k$, store $(Id,\ q,\ r)$ in Q and respond with $(query,\ q,\ r)$.
– If the query is made from a wrong session $(Id \neq Id')$, store $(q,\ r)$ in Q_{Id}.

Program

– Upon receiving message $(program,\ Id,\ q,\ r)$ by the adversary $\mathcal{A}$ check if $(Id,\ q,\ r')$ is defined in Q. If this is the case, abort. Otherwise, if $r \in \{0,1\}^k$ store $(Id,\ q,\ r)$ in Q and $(Id,\ q)$ in P.
– Upon receiving $(isPrgrmd,\ q)$ from a party of session Id, check if $(Id,\ q)$ in P. If this is the case respond with $(isPrgmd,\ 1)$.

Observe

– Upon receiving message $(observe)$ from the adversary $\mathcal{A}$ of session id Id respond with $(observe,\ Q_{Id})$.

Fig. 6. Global restricted programmable and observable random oracle functionality $\mathcal{H}$ [15], provides three interfaces *Query*, *Program*, and *Observe* to participants.

B Our UC framework

B.1 Extensions of the framework

We first extend the security definition given in [35] by separating the adversaries to the sets of deviating and semi-honest adversaries. We analyze the protocol in two steps: (i) *Ensuring security*: the analysis of the malicious adversaries will proceed as before (we can consider a monolithic model of adversaries if they are allowed to collude and achieve the standard UC security for them or consider a non-monolithic model of adversary if they are non-colluding). (ii)*Ensuring privacy*: when considering the semi-honest adversaries (like smart contract in our protocol) we assume the deviating parties are also present but they are non-cooperative with respect to semi-honest adversaries. Note that this definition is weaker than UC since the environment does not have the same power and abilities that it has in UC definitions. We need to clarify how the deviating adversaries interact with the semi-honest adversaries in the ideal world. This is important as we want to ensure they are non-colluding. We require that the public information that is needed to be seen by the semi-honest entities are leaked by the ideal functionally to the semi-honest simulators in the ideal world. This is a critical task since we

need to make sure we are not revealing more than it is needed. We have shown how this is done for pFairSwap ideal functionality.

To extend Kamara et al. framework to be used with global ideal functionalities, we need to restrict the access of isolated adversaries to interfaces that leaks information about the private inputs of other parties. Additionally, whenever $\mathcal{A}_i$ is non-cooperative with respect to $\mathcal{A}_j$ we have to modify the global ideal functionalities such that they do not leak any private information about $\mathcal{A}_i$ to $\mathcal{A}_j$.

In general, when dealing with global ideal functionalities, we need to consider the existence of non-monolithic adversaries and let the ideal functionality to respond to each of the adversaries based on their type. For example, the global random oracle functionality $\mathcal{H}$ [15] gives *Observe* interface to adversaries $\mathcal{A}$, which can be used to observe a list of illegitimate queries for session Id; illegitimate queries are made by parties that are not part of session Id. So, this observability will not give the adversary any new information, as it contains only queries made by the adversary (honest parties only send legitimate queries). In the ideal world, however, the simulator Sim can observe all queries made by honest and corrupted parties, as it is the ideal-world attacker, which means it will see all legitimate queries, irrespective of the fact that they have been issued by honest or corrupted parties. Therefore, this observability can leak useful information about the private input of corrupted parties to smart contract adversary $\mathcal{A}_{sc}$ and in the ideal world, it can leak the useful information of both honest and corrupted parties to Sim_{sc}. To prevent this leakage we restrict the access of $\mathcal{A}_{sc}$ (and hence Sim_{sc}) to *Observe* interface of $\mathcal{H}$. With this modification still the corrupted protocol parties $\mathcal{A}$ have access to illegitimate queries and Sim have access to all queries, and we can preserve the functionality of $\mathcal{H}$ in our modeling. Note that in reality smart contract adversary is independent of the protocol parties and this separation of access is meaningful and necessary.

B.2 Security of SC-aided OFE

Real world. In the real world, smart contract is an entity with access to a Ledger functionality $\mathcal{L}$ as a subroutine, and interacts with protocol parties to assist them in their computation. We consider two types of adversaries, (i) a set of deviating adversaries adversary $\mathcal{A}_i$ which can corrupt protocol parties p_i and (ii) a semi-honest adversary $\mathcal{A}_{sc}$ that models leakage to the SC. $\mathcal{A}$ is non-cooperative with respect to $\mathcal{A}_{sc}$. Environment $\mathcal{Z}$ provide inputs x_i and auxiliary information z_i to all parties except the smart contract which only receives auxiliary information. Smart contract interacts with protocol parties and at the end all parties except the smart contract sends an output to $\mathcal{Z}$. $\mathcal{Z}$ has also access to the ledger functionality $\mathcal{L}$. Lets denote the output of the protocol running with adversary $\mathcal{A}$ and the honest parties by $Real_{\pi,\mathcal{A},z}$ and the output of the protocol running with adversary $\mathcal{A}_{sc}$ (which is run alongside $\mathcal{A}$) and the honest parties by $Real_{\pi,\mathcal{A},\mathcal{A}_{sc},z}$. These two ensembles depends on the random coin of all parties.

Ideal world. In the ideal world, SC is a dummy entity with no input. Each party in the protocol, except the SC, receives an input from environment and hands it to the functionality $\mathcal{F}$. $\mathcal{F}$ will compute $f()$ and returns the result to each party, and also updates the Ledger functionality $\mathcal{L}$, if necessary. We consider two simulators Sim and Sim_{sc}. Sim captures the effect of $\mathcal{A}$ attacking the protocol Π against its basic properties, and Sim_{sc} captures the effect of $\mathcal{A}_{sc}$ attacking protocol Π against its privacy. Sim_{sc} has access to the Ledger functionality and can observe the transfer of funds for any party that it wants. Sim corrupts one of the parties p_i and has oracle access to $\mathcal{A}$. Lets denote the output of the protocol running with adversary Sim and the honest parties by $Ideal_{\mathcal{F},Sim,z}$ and the output of the protocol running with adversary Sim_{sc} (which is run alongside Sim) and the honest parties is denoted by $Ideal_{\mathcal{F},Sim,Sim_{sc},z}$.

Definition 1. *Let π be a protocol that realizes a smart contract aided n-party functionality $\mathcal{F}$. Furthermore, let $H \subseteq [n]$, let $I_{nc} \subseteq [n]$ denote the set of non-colluding parties, $I_c \subset [n]$ denote the set of colluding parties, SC denote the semi-honest smart contract, such that all subsets are pairwise disjoint. Let ADV be the adversary structure. We say that π $(I_{nc},\ I_c,\ ADV)$-securely realizes $\mathcal{F}$, if for any PPT adversary $\mathcal{A}_i \in \{I_{nc},\ I_c\}$ and smart contract adversary $\mathcal{A}_{sc}$ there exist PPT transformations Sim_i and Sim_{sc} respectively such that the following are negligible:*

Security:

$$|Pr[Real^{(i)}_{\Pi,\mathcal{A},Z}(k,x)=1] - Pr[Ideal^{(i)}_{\mathcal{F},Sim,Z}(k,x)=1]|.$$

Where $Real^{(i)}_{\Pi,\mathcal{A},z}(k,x)$ denotes the view of the adversary $\mathcal{A}_i$ and output of honest parties when running protocol Π. $Ideal^{(i)}_{\mathcal{F},Sim,z}$ is the view of malicious parties and output of honest parties when running the ideal process computing $\mathcal{F}$. $\mathcal{A} = \{\mathcal{A}_1, ..., \mathcal{A}_m\}$ and $Sim = \{Sim_1, ..., Sim_m\}$.

Smart contract privacy:

$$|Pr[Real^{(j)}_{\Pi,\mathcal{A},\mathcal{A}_{sc},Z}(k,x)=1]-$$
$$Pr[Ideal^{(j)}_{\mathcal{F},Sim,Sim_{sc},Z}(k,x)=1]|$$

Where $Real^{(j)}_{\Pi,\mathcal{A},\mathcal{A}_{sc},z}$ denotes the view of the adversary $\mathcal{A}_{sc}$ and output of all other parties when running protocol Π in the presence of adversary $\mathcal{A}_j$. $Ideal^{(j)}_{\mathcal{F},Sim,Sim_{sc},z}$ is the view of the semi-honest parties and output of all other parties when running the ideal process computing $\mathcal{F}$ in the presence of adversary Sim_j. k is the security parameter, and x is the set of outputs provided to all parties. z is the set of auxiliary inputs provided to all parties.

Note that in our definition the global ledger functionality $\mathcal{L}$ only manages the transfer of coins which are public information and would not make $\mathcal{A}_{sc}$ stronger than it is.

C ℓ-Private arithmetic circuits

Input encoder I. Let $\mathbb{F}_p$ denote the finite field of order p, where p is a large prime (chosen based on security parameter λ) with two operations + and multiplication $\times$. Each input value x in $\mathbb{F}_p$ is blown up to $m+1$ random values using $(m+1,\ m+1)$- secret sharing. For that m random values $r_1,\ ...,\ r_m \in \mathbb{F}_p$ are chosen and r_{m+1} is computed as $r_{m+1} = x-(r_1+r_2+...+r_m)$. Each input value is determined independent of other input values in the circuit.

Output decoder O. Each output value y in C is expanded to $m+1$ values in $T(C)$ such that $y = y_1+y_2+...+y_{m+1}$.

Circuit transformer T. An arithmetic circuit consists of multiplication and addition gates. We construct a transformer which takes the original circuit C and transforms it to circuit C' such that any wire in C is $m+1$ wires in C' and they are additive $(m+1,\ m+1)$-secret sharing in $\mathbb{F}_p$ where p is a large prime. C' is obtained by transforming each gate as below:

Addition gates. Transforming addition gates is straightforward. Each addition gate in C' takes two $m+1$ values as input. Lets denote such inputs as $a_1,\ ...,\ a_{m+1}$ and $b_1,\ ..,\ b_{m+1}$ such that $a = \Sigma_{i=1}^{m+1} a_i$ and $b = \Sigma_{i=1}^{m+1} b_i$. To find $c = a+b$ we have $c = \Sigma_{i=1}^{m+1}(a_i+b_i)$. This is equivalent to generate the shares of c as $c_i = (a_i + b_i)$, to get $m+1$ values in the output. Therefore each addition gate is converted to a gadget with $m+1$ addition gates in $\mathbb{F}_p$.

Multiplication gates. Now consider a multiplication gate in $\mathbb{F}_p$ with a and b as input and c as output. In circuit C' we have the corresponding $a_1,\ ...,\ a_{m+1}$ and $b_1,\ ..,\ b_{m+1}$ values. Thus, $c = a \times b = \Sigma_{i=1}^{m+1}(a_i \times b_i)$. To find the shares of c we follow the approach in [34] and extend their AND gadget to a modular multiplication gadget as below:

To transform this gate we first compute intermediate values $z_{i,j}$ for $i \neq j$. For each $1 \leq i < j \leq m+1$ we introduce a random gate which produces $z_{i,j}$. Then we compute $z_{j,i} = (-z_{i,j} + a_i \times b_j) + a_j \times b_i$. Therefore $z_{i,j}$ is distributed randomly but $z_{j,i}$ depends on a_i, b_j, a_j and b_i. Now we compute the output shares of $c_1,\ ...,\ c_{m+1}$ in C' as $c_i = a_i \times b_i + \Sigma_{j \neq i} z_{i,j})$. Thus, each multiplication gate is expanded to a gadget with $O(m^2)$ multiplication gates and $O(m^2)$ addition gates in C'. The gadgets in C' are connected in the same way that the gates in C are connected. C' is the transformed version of C and we let $T(C) = C'$.

Theorem 2. *There exists a perfectly ℓ-private transformer (T, I, O) that maps a circuit C of size n and depth d to a randomized circuit of size $O(n\ell^2)$.*

Proof *Simulation for a single multiplication gate.* This simulation proceeds the same as [34].

Simulation for a single addition gate. We determine the set of indices $I \subset [m+1]$ such that we can perfectly simulate the view of adversary using $a_{|I}$ and $b_{|I}$. I is determined as follows: For each addition wire of the form $a_i + b_i$ add index i to I. When all indexes are determined, use the randomly assigned values to a_i and b_i to compute $a_i + b_i$. Note that the size of I is at most ℓ since each addition wire adds at most one index to I and there are at most ℓ gates that can be queried by the adversary. Since each a_i and b_i are the secret shares with the property that any m values are uniformly distributed, observing $\ell < m$ values will not allow the adversary to distinguish a real computation of circuit C' from the simulated one.

Lemma. *If attacker is restricted to observe t gates in the circuit C', with two input and one output wire, then it is enough to set $\ell = \frac{t}{3}$.*

The reason is that the inputs and output of the gates are not independent which results in an efficiency gain for transformation.

C.1 Complexity of the expanded circuit

Let $m = 2\ell$ where ℓ is the number of wires that are visible by the adversary. Number of multiplication gates per Mult gates is $n_{mult} = (m+1)^2$. Number of addition

gates per Mult gate is $n_{add} = 2[m(m+1)]$. Number of random values per Mult gates is $n_{rand} = \frac{m(m+1)}{2}$. Number of addition gates per Add gates is $m+1$.

D Protocol flow

Figure 7 shows the flow of the SC-aided OFE protocol.

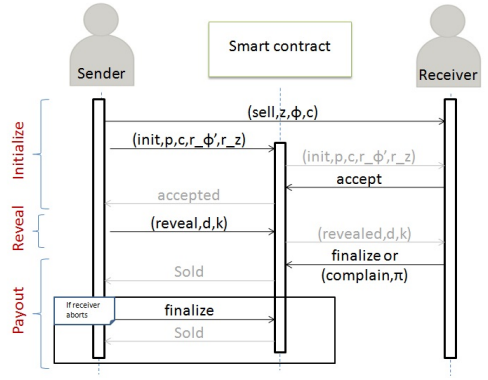

Fig. 7. Outline of the SC-aided fair exchange

E pFairSwap

Our private FairSwap is different than the original FairSwap in that it operates on the expanded circuit ϕ', instead of ϕ itself. Using ϕ' allows us to ensure the privacy of $\phi(x)$ when a PoM is constructed. In the case that *Extract* runs and returns a PoM π, the receiver can send π to SC that can verify the PoM without knowing any of the wire values in the original circuit $\phi(x)$. The circuit expansion requires three subroutines *InputEncode*, *OutputDecode*, and *Expand*.

The Algorithm 5 describes the modified *Encode* and Algorithm 6 describes the modified *Extract* in details. Function *Encode* (see Algorithm 5) is run by the Sender to compute the expanded circuit ϕ' on encoded input x' and random values ρ'. It encrypts all the results with key k using a symmetric encryption scheme. *Encode* takes as input a circuit ϕ', a digital good x', randomness ρ', and symmetric encryption key k, and outputs the set z. The algorithms algorithms 7 and 8 are also shown below. *InputEncode* is the generalization of the *EncodeInputs*, given in Section 6, for expansion factor $m+1$, and it is omitted because of the space. The *Judge* algorithm is the same as *Judge* in FairSwap.

Algorithm 5 Expand

set $n_I = n(m+1)$ ▷*Set number of input blocks*
set $\bar{\phi} : OutputDecoder$ ▷*Represent the output decoder as a circuit with fan-in 2 gates*
set $\hat{\phi} = \phi || \bar{\phi}$ ▷*Concatenate the OutputDecoder circuit with the main circuit*
for $i \in [n_I]$ **do**
  $out_i = x_i$ ▷*Assign witness to input wires*
  $z_i = Enc(k,\ out_i)$ ▷*Encrypt input values*
for $i \in [n_{mult}]$ **do**
  Parse $\rho = (i, \rho_1,\ ...,\ \rho_{n_{rand}})$
  for $j \in [n_{rand}]$ **do**
    $idx \leftarrow Map(i, j)$ ▷*Map the labels of the random wires to the random inputs of the circuit ϕ*
    $out_{idx} = \rho_j$ ▷*Assign random values to wires*
    set $idx_2 = n_I + (i-1)n_{rand} + j$ ▷*Set the index of encryption*
    $z_{idx_2} = Enc(k,\ out_i)$ ▷*Encrypt random values*
for $i \in \{n_I + n_{Mult} \times n_{rand} + 1,\ ..,\ N\}$ **do**
  Parse $\hat{\phi}_i = (i,\ op_i,\ I_i)$
  $out_i = op_i(out_{I_i[1]},\ ...,\ out_{I_i[\ell]})$ ▷*Compute the i-th operation*
  $z_i = Enc(k,\ out_i)$ ▷*Encode output values*
Output: $z = (z_1,\ ...,\ z_N)$

Algorithm 6 Extract

Set $n_I = n(m+1)$
for $i \in [n_I]$ **do**
  $out_i = Dec(k,\ z_i)$ ▷*Decrypt first n outputs*
  $x_i = out_i$ ▷*Extract witness*
for $i \in \{n_I + 1,\ n_I + n_{rand} \times n_{Mult}\}$ **do**
  $idx || \rho_i = Dec(k,\ z_i)$ ▷*Decrypt random values*
  $out_{idx} = \rho_i$ ▷*Extract random values*
$M_z = Mtree(z)$ ▷*Compute Merkle tree over z*
$M_\phi = Mtree(\phi)$ ▷*Compute Merkle tree over ϕ*
for $i \in \{n_I + n_{rand} \times n_{Mult} + 1,\ ...,\ N\}$ **do**
  Parse $\phi_i = (i,\ op_i,\ I_i)$
  $out_i = op_i(out_{I_i[1]},\ ..,\ out_{I_i[\ell]})$ ▷*Compute output of i-th gate*
  if $Dec(k,\ z_i) \neq out_i$ or $(i = N$ *and* $out_i \neq 1)$ **then**
    $\pi_\phi = Mproof(i,\ M_\phi)$ ▷*Proof that $\phi_i \in \phi$*
    $\pi_{out} = Mproof(i,\ M_z)$ ▷*Proof that $z_i \in z$*
    for $k \in [\ell]$ **do**
      set $j = I_i[k]$ ▷*j is the k-th index in the set I_i*
      $\pi^k_{in} = Mproof(j,\ M_z)$ ▷*Proof that $z_j \in z$*
    set $\pi = (\pi_\phi,\ \pi_{out},\ \pi^1_{in},\ ..,\ \pi^\ell_{in})$
  Output: $((x_1,\ ...,\ x_n),\ \pi)$
Output: $((x_1,\ ...,\ x_n),\ \perp)$

Algorithm 7 OutputDecode

1: set $m = 2\ell$
2: **Parse** $y = \{y_1, ..., y_{m+1}\}$ ▷*Obtain all secret shares of the output*
3: $y' = \Sigma_{i=1}^{m} y_i \mod p$ ▷*Reconstruct the output*
4: **Output:** y'

Algorithm 8 Judge

1: **Parse** $\pi = (\pi_\phi, \pi_{out}, \pi_{in}^1, ..., \pi_{in}^\ell)$
2: **Parse** $\pi_\phi = (\phi_i, \rho_\phi)$
3: **Parse** $\phi_i = (i, op_i, I_i)$
4: **if** $Mvrfy(\phi_i, \rho_\phi, r_\phi) \neq 1$ **then output: 0** ▷*Reject of ϕ_i not i-th step of $\phi(x)$*
5: **Parse** $\pi_{out} = (z_i, \rho_{out})$
6: **if** $Mvrfy(z_i, \rho_{out}, r_z) \neq 1$ **then output: 0** ▷*Reject of z_i not i-th step of z*
7: $out_i = Dec(k, z_i)$
8: **if** $i = m$ **and** $out_i \neq 1$ **then output: 1** ▷*accept if $\phi(x) \neq 1$*
9: **for** $j \in [\ell]$ **do** ▷*j is the k-th index in the set I*
10: **Parse** $\pi_{in}^j = (z_j, \rho_j)$
11: **if** $Mvrfy(z_j, \rho_j, r_z) \neq 1$ **then output: 0** ▷*Reject of z_j not $z[j]$*
12: $out_{I_i[j]} = Dec(k, z_j)$
13: **if** $op_i(out_{I_i[1]}, ..., out_{I_i[\ell]}) \neq out_i$ **then output: 1** ▷*Accept*
14: **else Output:0** ▷*Reject complaint if evaluation correct*

F Security analysis

Simulation with respect to $\mathcal{A}_S$. For a malicious sender $\mathcal{S}^*$, Sim^S generates a random value taken from uniform distribution and shares it as a seed s_0 with sender and receiver. Sim^S (with oracle access to the code of $\mathcal{S}^*$) has to use $\mathcal{H}(observe)$ (we use hash as a commitment scheme) to find the encryption key k. Using k, Sim^S can run *Extract* (Algorithm 6) to decrypt all the values received from $\mathcal{S}^*$, find x, and generate a proof of misbehavior π that can be used to simulate the messages sent to $\mathcal{F}_{cfe}^{\mathcal{L}}$, and SC. Sim^S only internally simulates the behavior of SC and the input and output of SC are given to environment. The simulation is indistinguishable from the real world execution if the commitment scheme is binding.

Simulation with respect to $\mathcal{A}_R$. For a malicious receiver $\mathcal{R}^*$, Sim^R generates a random value taken from uniform distribution and shares it as a seed s_0 with sender and receiver. Sim^R chooses a random key k^* and random z^* (encrypted values). When in the second round Sim^R learns the key k, it uses *Extract* and gets x. Then it computes $\phi(x)$, encrypts the results using k, and program $\mathcal{H}$ to output z^* when queried by environment. The simulation is indistinguishable from the real world execution as long as the commitment scheme is information hiding and the encryption scheme ensures IND-CPA security.

Simulation with respect to $\mathcal{A}_{sc}$. – Upon receiving $(sell, id, \phi, le(\phi), p, \mathcal{S})$, Sim_{sc} samples $k \leftarrow Gen(1^\lambda)$, computes $(c, d) \leftarrow Commit(k)$, $\phi' = Expand(\phi, \ell)$. Additionally, chooses z' and ρ' randomly. Then, he sends $(init, id, p, c, r_{\phi'}, r_{\rho'}, r_z)$ to SC where $r_{\phi'} = root(Mtree(\phi'))$, $r_{\rho'} = root(Mtree(\rho'))$ and $r_z = root(Mtree(z'))$ where z' is a set with length equal to z but it consists of random values. Finally, output $(initialized, id, p, r_{\phi'}, r_{\rho'}, r_z, c)$ and set $s = initialized$.
– Upon receiving $(buy, id, \mathcal{R})$, simulate $(accept, id)$ from $\mathcal{R}$ to SC and set $s = active$.
– Upon receiving $(abort, id, \mathcal{S})$, abort. If no message is received responds with $(reveal, id, d, k)$, set $s = revealed$, and send $(revealed, id, d, k)$ to everyone.

Environment $\mathcal{Z}$ cannot distinguish the simulation from the real world execution since the only way to distinguish is to learn z'. As the randomized Merkle tree commitment scheme is information hiding, the probability to learn z' is negligible.

Simulation with respect to $\mathcal{A}_S$ and $\mathcal{A}_{sc}$ – Upon receiving $(sell, id, \phi, le(\phi), p, \mathcal{S})$, samples $k \leftarrow Gen(1^\lambda)$, computes $(c, d) \leftarrow Commit(k)$, $\phi' = Expand(\phi, \ell)$. Additionally, Sim_{sc} chooses z' and ρ' randomly. Then, he sends $(init, id, p, c, r_{\phi'}, r_{\rho'}, r_z)$ to SC where $r_{\phi'} = root(Mtree(\phi'))$, $r_{\rho'} = root(Mtree(\rho'))$ and $r_z = root(Mtree(z'))$ where z' is a set with length equal to z but it consists of random values. Finally, output $(initialized, id, p, r_{\phi'}, r_{\rho'}, r_z, c)$ and set $s = initialized$.
– Upon receiving $(buy, id, \mathcal{R})$, simulate $(accept, id)$ from $\mathcal{R}$ to SC and set $s = active$.
– Upon receiving $(abort, id, \mathcal{S})$, abort. If no message is received responds with $(reveal, id, d, k)$, set $s = revealed$, and send $(revealed, id, d, k)$ to everyone.
– Upon receiving $(abort, id, \mathcal{R})$, wait until round 5 and send $(sold, id)$ to $\mathcal{S}$. Otherwise, check the Ledger $\mathcal{L}$. If there is a message $(unfreeze, id, p, \mathcal{S})$ send $(sold, id)$ to $\mathcal{S}$, Else if there is a message $(unfreeze, id, p, \mathcal{R})$, simulate a complain $(complain, id, \pi)$ from $\mathcal{R}$, where π is generated for a gate that has been specified in $le(\phi)$,

and send (*not sold*, *id*) to $\mathcal{S}$. In order to simulate a valid complain for gate i which is parsed as (i, op_i, I_i), Sim_{sc} chooses random values for I_i and output of gate i, then encrypts the inputs and outputs using key k and then program the random oracle such that $\mathcal{H}$ queries on these values lead to the leaves of the tree constructed on z'.
– If no message (*sold*, *id*) or (*not sold*, *id*) from SC is received send ($finalize$, *id*) to SC and (*sold*, *id*) to $\mathcal{S}$.

The expanded circuit provides ℓ-privacy environment cannot distinguish the random values chosen for the inputs and outputs of the gates from the real inputs and outputs. Additionally, PRG is secure and revealed wires with random values are indistinguishable from values taken from uniform distribution (note that seed of PRG is private to sender and receiver). Finally, Sim_{sc} can successfully program $\mathcal{H}$, except with negligible probability, so the environment is not able to distinguish the real world and the ideal world execution with non-negligible probability.

Simulation with respect to $\mathcal{A}_R$ and $\mathcal{A}_{sc}$.
– Upon receiving (*sell*, *id*, ϕ, $le(\phi)$, p, $\mathcal{S}$), store $le(\phi)$. Then, samples $k \leftarrow Gen(1^k)$, computes (c, d) $\leftarrow Commit(k)$, output (*initialized*, *id*, p, $r_{\phi'}$, $r_{\rho'}$, r_z, c) to $\mathcal{R}$ where $r_{\phi'}$, $r_{\rho'}$ and r_z are chosen randomly, and set $s = initialized$.
– Upon receiving (*buy*, *id*, $\mathcal{R}$), simulate (*accept*, *id*) from $\mathcal{R}$ to SC and set $s = active$.
– Upon receiving (*abort*, *id*, $\mathcal{S}$), abort. If no message is received responds with (*reveal*, *id*, d, k), set $s = revealed$, and send (*revealed*, *id*, d, k) to everyone.
– Upon receiving (*abort*, *id*, $\mathcal{R}$), abort. Otherwise, check the Ledger $\mathcal{L}$; if there is a message ($unfreeze$, *id*, p, $\mathcal{S}$), but $le(\phi) = \perp$ terminate, otherwise simulate an invalid complain (*complain*, *id*, π) from $\mathcal{R}$, where π is generated for a gate which has been specified in $le(\phi)$.

The view of the environment is indistinguishable from the real world execution since the complaint does not reveal anything about the item x due to the ℓ-privacy of the circuit expansion and security of PRG.

G Information leakage in FairSwap

The Merkle proofs in the PoM leaks sibling node of their Merkle tree commitment (e.g. z_{i+1}), and some hash values $H(Z_{i+1}|Z_{i+3})$. The leaked information z_i is: $MProof(z_i, r_z) = \{z_{i+1}, H(z_{i+2}|Z_{i+3}), ...\}$
Each Merkle proof leaks information about two wires. In the best case the information leaked is about four wires and in the worst case information about six wires is revealed. If the corrupted gate takes two chunks of items x as input then in the best case two chunks of x will be revealed (plus some intermediary results of computing ϕ). In the worst case four chunks of item x are leaked (plus some intermediary results of computing ϕ). This is equivalent to revealing the information of three gates partially.

H Optimizations

The *Extract* algorithm does not need to decrypt and verify every gate in z. Instead, it only needs to decrypt the secret shares of the input and output wires of the gates in the original circuit, combine the secret shares, and execute the operation of the original gate. Any result computed by *Extract* that does not match the decrypted secret shares implies a cheating gate inside the gadget. *Extract* can then decrypt the gates inside the suspect gadget to find at least one gate that is corrupted. Then *Extract* can create a PoM for this gate. To know which gates in z are the secret shares of input and output wires in the original circuit, the buyer must create a mapping from the index of the original circuit to the indexes in the expanded circuit.