OptiSwap: Fast Optimistic Fair Exchange *

Lisa Eckey, Sebastian Faust, and Benjamin Schlosser

Technical University of Darmstadt, Germany
{lisa.eckey, sebastian.faust, benjamin.schlosser}@tu-darmstadt.de

Abstract. Selling digital commodities securely over the Internet is a challenging task when Seller and Buyer do not trust each other. With the advent of cryptocurrencies, one prominent solution for digital exchange is to rely on a smart contract as a trusted arbiter that fairly resolves disputes when Seller and Buyer disagree. Such protocols have an *optimistic mode*, where the digital exchange between the parties can be completed with only minimal interaction with the smart contract. In this work we present OptiSwap, a new smart contract based fair exchange protocol that significantly improves the optimistic case of smart contract based fair exchange protocols. In particular, OptiSwap has almost no overhead in communication complexity, and improves on the computational overheads of the parties compared to prior solutions. An additional feature of OptiSwap is a protection mechanism against so-called grieving attacks, where an adversary attempts to violate the financial fairness of the protocol by forcing the honest party to pay fees. We analyze OptiSwap's security in the UC model and provide benchmark results over Ethereum.

1 Introduction

Fair exchange of digital goods for money is a challenging task in the Internet when parties do not trust each other. Consider a two party protocol between a Seller $\mathcal{S}$ and a Buyer $\mathcal{B}$. On the one hand, $\mathcal{S}$ possesses some digital good x, which he is willing to sell for p coins. On the other hand, $\mathcal{B}$ wants to obtain x in exchange for the money. Since the Internet connects millions of users, it is reasonable to assume that both parties do not trust each other. While the Seller wants the guarantee that $\mathcal{B}$ gets to know x only if he pays p coins, the Buyer must be assured that he only needs to pay if $\mathcal{S}$ delivers the correct witness. It was shown by Pagnia and Gärtner in 1999 that such a fair exchange is not possible without a trusted third party (TTP) [25].

A common use case for a fair exchange protocol is the purchase of some digital file x, e.g., a movie, music file or software executable where only the hash $h = H(x)$ is known. The Buyer only wants to pay if the file x that he receives satisfies $h = H(x)$. At the same time, the Seller wants to be sure that if he delivers x which hashes to h, he gets his payment. The above can be generalized to a predicate function ϕ, where the Seller wants to sell a witness x that satisfies $\phi(x) = 1$. For instance, x may be some software, and ϕ defines test cases that the software has to satisfy. There exist different approaches to realize fair exchanges. One approach relies on a trusted middleman, which is often called escrow service, that implements the TTP [24]. A different approach is taken by protocols that rely on a blockchain to implement the TTP in case of dispute. Examples include the zero-knowledge contingent payment (ZKCP) protocols [6], and the FairSwap protocol [14]. In these works the TTP is implemented in a smart contract running on top of the blockchain. At a high-level these protocols work as follows. The Seller $\mathcal{S}$ encrypts the witness and sends it together with some auxiliary information to the Buyer. The Buyer checks the auxiliary information and, if accepted, it deposits money to the smart contract. Once the money is locked the Seller will reveal the secret key to the contract, which allows him to claim the money. Since

* This is the full version of the paper appearing at the ACM ASIA CCS '20, https://doi.org/10.1145/3320269.3384749.

the secret key now becomes publicly recorded on the blockchain, the Buyer can use it to decrypt the encrypted data.

Most relevant for our work is the protocol FairSwap [14]. FairSwap avoids expensive zero-knowledge proofs, and replaces it with a technique called *proof of misbehavior (PoM)*. A PoM allows the Buyer to punish the Seller in case the Seller sent an invalid witness x. This a-posterior approach of punishment has the advantage that FairSwap only needs to rely on simple hash function evaluations, thereby resulting in a more efficient protocol for the users – in particular, it is more efficient if the witness x is large or the predicate function ϕ is complex. On the downside the communication between Seller and Buyer increases with the size of ϕ and x. Since the communication overhead occurs regardless of misbehavior, it is most likely the main bottleneck in practice.

We present an extension to the original FairSwap protocol of Dziembowski et al. [14] in order to improve the optimistic execution of the protocol, and thereby overcome one of the main bottlenecks of FairSwap in practice. This is achieved by incorporating an interactive dispute resolution sub-protocol, which removes the message overhead and thereby also improves the computational complexity of the two honest parties. Because our protocol is optimized for the *optimistic case* (which we believe is the standard case in practice, since parties can be punished when cheating), we call our protocol OptiSwap.

Another shortcoming of [14] is that grieving is not discussed. A corrupt Seller could force any Buyer to submit a large transaction (including the PoM) to the blockchain. This means a Seller can *grieve* the Buyer by forcing him to pay a lot of transaction fees. We will show how we can protect against grieving without relying on any security deposits when there are no disputes.

1.1 Contribution

We extend the FairSwap protocol by Dziembowski et al. [14] by making the dispute resolution interactive. Our contributions are summarized below:

- We present a challenge-response procedure for obtaining information about the evaluation of the predicate function ϕ. This procedure is only used in case of dispute, and allows to pinpoint a single step in the evaluation of ϕ where the Seller cheated. This approach significantly reduces the overheads of the FairSwap protocol from [14] in the optimistic case.
- In order to incentivise both parties to act honestly and prevent grieving, we incorporate fees into the dispute resolution sub-protocol.
- We implemented the OptiSwap protocol and improved it using the technique introduces by SmartJudge [27] in order to estimate the gas costs [1]. The obtained values are compared against the original FairSwap protocol.
- Finally, we provide comprehensive security evaluation of OptiSwap in the UC framework and show that it realizes a fair exchange functionality.

1.2 Related Work

It is well-known that fair exchange is impossible without a TTP [25, 28, 17]. In order to limit the role of a TTP [3, 7] proposed the distinction between the optimistic and pessimistic case of fair exchange. Since the emerge of blockchain technologies, smart contracts are considered as TTP, and there are numerous works that use smart contracts to improve on the fairness properties of cryptographic protocols [2, 5, 22, 23, 21]. Various works use smart contracts for fair exchange [6, 14, 9]. Below, we discuss related work that utilized interactive dispute resolution.

[1] In Ethereum, parties have to pay for executed computation, where computation is measured in units of gas. Hence, the gas costs can be used as efficiency indicator.

Interactive dispute resolution. Recent work incorporated an interactive dispute resolution handling for the case when two parties run into disagreement. The Arbitrum system was introduced by Kalodner et al. in 2018 [19]. They proposed a solution for the scalability issue related to many kinds of blockchain technologies. By performing heavy computation off-chain, the complexity of computation that is feasible to carry out backed-up by the blockchain is increased. Similar to OptiSwap in case of disagreement, Arbitrum runs an interactive dispute resolution protocol. This protocol narrows down the computation under dispute to a single instruction which is recomputed on-chain in order to resolve the whole disagreement. While OptiSwap also contains a challenge-response procedure as part of its dispute resolution protocol, we use a different challenge strategy. Concretely, in Arbitrum, computation is modeled using Turing machines resulting in a linear sequence of computation steps. For this type of data structure, a binary search algorithm is very efficient and, hence, it is used by Arbitrum. Within the OptiSwap protocol, we model computation using circuits that may have a far more complex topology than a linear list. Therefore, we make use of different challenge strategies.

A scalable verification solution for blockchain technologies called TrueBit was proposed by Teutsch and Reitwießner originally in 2017 [26]. It contains a dispute resolution layer which is executed in case of disagreement about the computation of some task. As in the Arbitrum system, TrueBit models computation using Turing machines. Hence, for resolving a dispute, the computation is narrowed down using a binary search procedure. As above the main difference to OptiSwap is that we use a circuit-based approach that allows more complex challenge strategies.

Optimistic mode. The SmartJudge architecture was proposed by Wagner et al. in 2019 [27]. It provides an abstraction of a two-party protocol that relies on the execution of a smart contract in case of dispute. Since our OptiSwap protocol is optimized for the optimistic mode, it is a perfect candidate to be combined with SmartJudge. SmartJudge splits a smart contract into a mediator and a verifier. Only the mediator must be executed in the optimistic mode and the verifier is only called in case of dispute. While SmartJudge provides an improvement for two-party protocol in the optimistic mode, it does not include a dispute resolution mechanism as this will depend on the actual protocol that instantiates the SmartJudge approach. We can combine our protocol with SmartJudge to further improve the optimistic mode and reduce on fee costs.

Fair exchange based on conditional dispute resolution. Concurrently to our work, Hall-Andersen presented an improvement of the FairSwap protocol called FastSwap [18]. FastSwap aims at an improved communication complexity in case of both parties behaving honestly. This protocol incorporates an interactive challenge-response sub-protocol that is only executed in case of misbehavior. The work of Hall-Andersen focuses on modeling the verification computation using a computation model similar to Turing machines. This model results in a linear sequence of computation steps which allows a logarithmic number of rounds in the challenge-response phase. In contrast to FastSwap we use circuits for modeling computation. This allows us to use different challenge strategies depending on the topology of the verification circuit. Moreover, we explicitly add security fees into our protocol design to prevent grieving attacks. We stress that both protocols are compatible to each other which means that incorporated techniques can be adopted to be used in the other protocol as well. Since the FastSwap paper lacks a detailed evaluation of gas costs in the Ethereum system, a concrete comparison of both protocols is not possible. Finally, we provide a detailed security proof of our protocol in the universal composability framework in Appendix E. While the FastSwap protocol is also modeled in this framework, a full security proof is missing.

2 Preliminaries

In this section we introduce the main building blocks and describe the cryptographic primitives we need for designing OptiSwap. We denote integer variables by lower case letters and tuples by bold lower case letters (i.e., $\mathbf{a} = (a_1, \ldots, a_n)$) while sets are signaled by capital letters. The integer set $\{1, \ldots, n\}$ is denoted by $[n]$. For a probabilistic algorithm $\mathcal{A}$, $m' \leftarrow \mathcal{A}(m)$ denotes the output m' of $\mathcal{A}$ on input m. The symbol $\approx_c$ denotes the computational indistinguishability between two random variables. $s \xleftarrow{\$} S$ denotes that s is randomly sampled from the set S using a uniform distribution. Furthermore, we assume direct secure channels and synchronous communication happening in rounds. By using the notion of rounds, we abstract from the underlying process of consensus and confirmation times required when we interact with a blockchain system. The period of Δ rounds ensures that an honest party is able to wait for confirmation and to send a transaction afterwards.

Circuits. A key ingredient to our protocol will be a circuit which we use to model the predicate function that validates correctness of a witness. We will consider a witness $\mathbf{x}$ to be correct if the circuit evaluation of it returns 1, i.e. $\phi(\mathbf{x}) = 1$. Let X be a set of possible inputs and Γ an instruction alphabet. Each instruction $op : X^\ell \to X$ in Γ takes up to ℓ input values (fan-in: ℓ) and returns a single output value. A circuit ϕ over a value set X and an instruction set Γ is modeled as a directed acyclic graph (DAG) as follows. Its vertices $\phi_1, \ldots \phi_m$ are called *gates* and its edges represent the *wires* of ϕ and specify the information flow through the circuit. Any gate with in-degree 0 is called an *input gate*. The circuit contains n input gates, where $1 \leq n < m$, and each input gate ϕ_i outputs v_i to all outgoing edges. Every gate in ϕ with out-degree 0 is called *output gate*. We require that the circuit has a single output gate that evaluates to either 1 or 0. Every gate ϕ_i is labeled by a tuple $\phi_i = (i, op_i \in \Gamma, I_i \in [m]^\ell)$, evaluates the instruction op_i on the input values given by the circuit gates specified by the index set I_i, and outputs the result to all outgoing edges. There exists an ordering over all circuit gates such that for each gate ϕ_i the index i is greater than any index of a gate that provides an input to gate ϕ_i.

We note that the evaluation of a circuit is performed layer-by-layer starting from the input gates. Each gate evaluates its instruction on given inputs and propagates the result to the gates of the next layer. For simplicity we denote the outcome of the overall circuit as $\phi(\mathbf{x}) = \{0, 1\}$ and the result of the ith gate as $\phi_i(\mathbf{x}) = \mathsf{outp}_i$, for all $i \in [m]$. The depth δ of a circuit is the length of the longest path starting from any possible input gate up to its output gate.

Hash functions. We make use of cryptographic hash function $H : \{0,1\}^* \to \{0,1\}^\mu$, where $\mu \in \mathbb{N}$. These functions are used as building blocks for commitment and encryption schemes as well as Merkle trees. For our construction, we assume H to be a restricted programmable and observable random oracle. In Appendix B we give a detailed description of how we realize this primitive in the global random oracle model.

Commitment schemes. A commitment scheme allows to commit to a value m using the Commit-algorithm, which outputs a commitment c and an open value d. The commitment c hides m from others while the d can – at a later point – be used to reveal m. We require a commitment scheme to be *binding*, meaning that nobody can change a committed value m after c is known. Additionally, it must be *hiding*, i.e., nobody should learn the committed value from the public commitment c. Commitment schemes are formally defined in [16].

Symmetric encryption. We make use of a symmetric encryption scheme consisting of three ppt algorithms (KeyGen, Enc, Dec). The secret encryption key k is generated by the KeyGen-algorithm. The Enc-algorithm outputs a ciphertext z on input of the key k and a plaintext m. The Dec-algorithm returns the message m for given ciphertext c using key k. We assume a symmetric encryption scheme to be indistinguishable under chosen-plaintext attacks (IND-CPA) secure.

Merkle trees. Merkle trees are generated through the MTHash algorithm by iteratively hashing two hash values until only one hash – the root of the Merkle tree – remains. The root serves as a commitment on many chunks of data. The algorithm $\mathsf{MTProof}$ takes as input the tree and an index i and produces a logarithmically sized opening statement for the i-th chunk consisting of the neighbors on the path from the element to the root. The algorithm $\mathsf{MTVerify}$ is used to verify that the opening statement is correct in respect to the root. We refer to [14] for instantiation of these algorithms.

2.1 FairSwap

We briefly introduce the FairSwap protocol presented by Dziembowski et al. in 2018 [14] on a high level. The overall goal of the protocol is a fair exchange of a digital good $\mathbf{x}$ against money. The Seller, who knows $\mathbf{x}$, initiates the protocol by sending an offer message to the Buyer. This message contains an encryption $\mathbf{z}$ of every step of the witness evaluation $\phi(\mathbf{x})$ under key k. By confirming this message the Buyer also transfers the payment of p coins to the judge contract. At this point the contract contains the money from the Buyer and a concise commitment on both $\mathbf{z}$ and ϕ. The Seller now reveals the key via the contract which allows the Buyer to decrypt the witness. If this witness is correct, the money will be transferred to the Seller (eventually). The more interesting case occurs when the witness that the Buyer received does not satisfy the agreed upon predicate, i.e., $\phi(\mathbf{x}) \neq 1$.

In this case, which we call the disagreement or pessimistic case, the Buyer can send a *proof of misbehavior (PoM)* to the contract. This proof is small, compared to both the witness and the predicate, but it is sufficient to indubitably prove that the received witness $\mathbf{x} = \mathsf{Dec}(k, \mathbf{z})$ does not satisfy the predicate, i.e. $\phi(\mathbf{x}) \neq 1$. If this can be shown to the judge contract, the coins will be refunded to the Buyer. Besides making sure that no coins can be stolen or locked, the protocol of [14] prevents cheating, namely that the Seller will get paid for a wrong witness or the Buyer can successfully complain about a correct one. Additionally, the designed PoM and all data stored inside the contract are kept very small to reduce the costs of the protocol.

Proof of Misbehavior (PoM). In case of a dispute, Buyer and Seller argue about two different statements. The Seller wants the payment so he will claim that the transferred witness $\mathbf{x}$ is correct, i.e., $\phi(\mathbf{x}) = 1$ while the Buyer claims that it is incorrect $\phi(\mathbf{x}) = 0$. In order to figure out who is wrong the judge would need to re-evaluate the predicate $\phi(\mathbf{x})$. But since ϕ and $\mathbf{x}$ are potentially large, they should not be sent to the contract. Instead in [14] they consider a witness $\mathbf{x}$ that can be split in many elements $x_1, \ldots, x_n$ and a predicate ϕ which can be represented as a circuit with many gates $\phi_1, \ldots, \phi_m$. Now, instead of arguing about the evaluation of the overall predicate, it suffices to prove the dispute about the evaluation of a single gate ϕ_i where Buyer and Seller disagree on the gate output. Since the Buyer constructs and sends the PoM he has to prove that the Seller either sent the wrong witness or cheated during his computation of $\phi(\mathbf{x})$. This is why the Seller has to send the output for every evaluation of every gate of $\phi(\mathbf{x})$ to the Buyer and commit to all of these values (in the first message). These commitments allow the Buyer to prove that (i) either the output of $\phi(\mathbf{x}) = 0$ or (ii) there exists a gate $\phi_i \in \phi$ with inputs $\mathsf{in}_i^1, \ldots, \mathsf{in}_i^\ell$ where the Seller lied about the gate evaluation. The judge contract will then re-compute the gate in question and compare the outcome with the Seller's result, making it impossible to cheat successfully about the evaluation of ϕ_i. If the witness is wrong and the Seller computed the correct predicate, then the output of the last gate will reveal this fact, i.e. $\phi_m(\cdot) = 0$. But if the witness is correct, the Buyer will not be able to find a gate which is wrongly computed and the overall result will be 1, which means the judge will not accept any PoM.

A drawback of the FairSwap protocol is that the PoM is generated by the Buyer only considering information that the Seller sent in the first round. In particular the Seller needs to compute, store, encrypt and transfer the intermediary result of every single gate in the predicate evaluation $\phi(\mathbf{x})$. For predicate circuits with many gates, this can become a large

overhead for the Seller and also for the Buyer, since in most cases when there is no dispute these information are not needed. We propose a different protocol, which does not require this overhead as long as Seller and Buyer agree. Only in case they run into a dispute, they can execute an interactive protocol which allows them to securely prove whether $\phi(\mathbf{x}) = 1$. Additionally, we will analyze how to fairly estimate transaction fees, which is not considered in the work of [14] at all.

3 OptiSwap Protocol Description

Similarly to [14] we consider an interactive protocol between a Buyer $\mathcal{B}$ who is willing to pay a Seller $\mathcal{S}$ p coins in exchange for a witness $\mathbf{x}$, if $\mathbf{x}$ satisfies a predicate function ϕ (i.e. $\phi(\mathbf{x}) = 1$). They use a smart contract as an adjudicator that stores the payment and either transfers the coins to $\mathcal{S}$ if the exchange is completed successfully or back to $\mathcal{B}$ if he does not receive his witness. The witness $\mathbf{x}$ can be split into n elements $x_1, \ldots, x_n$ and the predicate is represented as a circuit with m gates $\phi_1, \ldots, \phi_m$, a fan-in ℓ, a depth δ and a width ω as defined in Section 2.

3.1 OptiSwap Properties.

In order to ensure that neither the Buyer nor the Seller can cheat, we require the fair exchange protocol to have the following security properties:

- S1 Security against malicious Buyers: Honest Sellers will always receive their payment if the Buyer learns the witness $\mathbf{x}$ (cf. Sender fairness of [14]).
- S2 Security against malicious Sellers: Honest Buyers will either learn the correct witness $\mathbf{x}$ or will get their coins back (cf. Receiver fairness of [14]).
- S3 Security against grieving: In case of a dispute, the cheating party must always compensate the cheated party for any transaction fees paid during the dispute process (fee fairness).

Additionally to the security properties, we also need our protocol to be efficient. Therefore, we define the three following efficiency requirements:

- E1 Circuit independent communication: The size of all messages sent by Seller and Buyer in the optimistic case must be $\mathcal{O}(|\mathbf{x}|)$.
- E2 Constant round complexity in the honest case: If the correct witness is transferred without dispute, the protocol must run at most 5 rounds.
- E3 Upper bounded round complexity in the pessimistic case: Even in case of dispute, the protocol terminates after finitely many rounds where the exact number of rounds is upper bounded by $\mathcal{O}(\min(\delta * \ell, \log(n) * \omega))$.[2]

In Section 5 we will argue why our protocol achieves all desired efficiency and security properties. A formal proof in the Universal Composability model can be found in Appendix E.

3.2 OptiSwap Protocol

At the core of the fair exchange protocol, Buyer and Seller use a smart contract, which has the authority over the payment and is used as a judge if the two parties disagree. Similarly to the FairSwap protocol of [14] the smart contract will evaluate a Proof of Misbehavior (PoM) in this case. This short proof is used to show that the Seller sent a wrong witness

[2] There are different dispute resolution procedures which are more or less efficient depending on the circuit parameters. The procedures are described in Section 3.5. In particular, the round complexity for the file sale application is $\mathcal{O}(\log(n))$.

(cf. Section 2.1). In contrast to the FairSwap protocol, the Buyer does not already have the data to generate a PoM. Instead he needs to interactively challenge it from the Seller.

In the following we describe all steps on the protocol informally. A formal description for every party as well as a formal model for ledger and contracts can be found in Appendix C and B, respectively.

Round 1 The Seller starts the protocol by encrypting the witness $\mathbf{x}$ element by element, i.e., $\forall i \in [n] : z_i = \mathsf{Enc}(k, x_i)$. Then he commits to both, the encryption (by computing $r_z \leftarrow \mathsf{MTHash}(z_1, \ldots, z_n)$) and the used encryption key $(c, d) \leftarrow \mathsf{Commit}(k)$. Now, the judge contract can be initialized by the Seller. It is parameterized by the addresses of Seller and Buyer, the price p, public auxiliary information aux of the witness exchange, including fee and timing parameters (cf. Section 5). Additionally, both commitments r_z and c are stored in the contract, while the encrypted data $\mathbf{z}$ is sent to the Buyer directly.

Round 2 In the second round, the Buyer needs to confirm that he received the first message from the Seller and that he agrees with the parameters in the contract. In particular, he recomputes the Merkle tree root r_z of the ciphertext and checks all contract parameters including the price and auxiliary information. If he does not agree with any of them, he aborts the protocol. Otherwise, he sends a transaction to the judge contract which signals his acceptance and transfers the agreed upon payment to the contract. Note, that the Buyer cannot decrypt the witness at this point, because he does not know the key k and neither the encryption $\mathbf{z}$ nor the commitment c reveal any information about $\mathbf{x}$ or k.

Round 3 Now that the coins are locked in the contract, the Seller can reveal the encryption key. In order to make this action publicly verifiable he stores the key in the smart contract, which verifies its correctness ($\mathsf{Open}(c, k, d) = \mathsf{true}$).

Round 4 Since the key is stored publicly, the Buyer can now decrypt the witness: $\forall z_i \in \mathbf{z} : \mathsf{Dec}(k, z_i) = x_i$. At this point we distinguish two cases. In the optimistic case the witness was correct, which the Buyer verifies by running $\phi(\mathbf{x}) = 1$. In this case he confirms the successful transfer to the contract, which triggers the payment to the Seller. Even when the Buyer does not send this message, the Seller can collect the coins after some timeout has passed in the next round. This timeout is necessary for the second case which we call the dispute or pessimistic case, in which the Buyer finds that the received witness is incorrect, i.e., $\phi(\mathbf{x}) = 0$. In this case the Buyer will start the interactive dispute protocol, which will effectively freeze the coins in the contract until a malicious party is found.

3.3 Interactive Dispute Handling

The dispute protocol is based on an interactive challenge-response procedure between Seller $\mathcal{S}$ and Buyer $\mathcal{B}$. The goal is to identify a single gate within the circuit in which the evaluations of both parties differ. Then the Buyer can use this information to prove that he received a false witness using a Proof of Misbehavior (PoM). Recall that for evaluating the PoM about the i-th gate, the judge requires as input the gate ϕ_i, its inputs $\mathsf{in}_i^1, \ldots, \mathsf{in}_i^\ell$ and the gate output computed by $\mathcal{S}$: outp_i. The Buyer will query these information from $\mathcal{S}$ by repeatedly asking for outputs of circuit gates. All queries and responses will be recorded by the judge contract.

The Buyer starts by querying the result for the last gate ϕ_m, which is the overall output of the evaluation of $\phi(\mathbf{x})$. If the Seller replies with $\mathsf{outp}_m = 0$ $\mathcal{B}$ can immediately prove that $\mathcal{S}$ misbehaved. If the Seller cheats, he could claim that $\mathsf{outp}_m = 1$, which means both parties have different results on the output of the gate. Now, $\mathcal{B}$ challenges the outputs of all input gates of ϕ_m and obtains $\mathsf{in}_m = (\mathsf{in}_m^1, \ldots, \mathsf{in}_m^\ell)$ which he can compare to his own computed values. Again the Seller can either send the values which correctly evaluated to his alleged result ($\phi_m(\mathsf{in}_m) = \mathsf{outp}_m$), or he could send mismatched values, such that $\phi_m(\mathsf{in}_m) \neq \mathsf{outp}_m$.

If the second case happens, again the Buyer triggers the PoM evaluation of the contract, which will verify that the Seller is cheating.

During this dispute resolution, the contract verifies the alternating participation of Seller and Buyer. In every repetition j, the Buyer challenges the input gates in_j to one or more of the challenged gate of the previous round $\mathsf{outp}_j \in \mathsf{in}_{(j-1)}$. The Seller has to respond by sending all corresponding output values to the challenged gates. This way the Buyer can evaluate $\phi_i(\mathsf{in}_j) = \mathsf{outp}_j$. Seller and Buyer repeat this challenge-response procedure until one of the following cases happens:

(i) The Seller does not respond to the challenge (in time). In this case the Buyer can request the money back from the contract.

(ii) The Buyer does not send a new challenge or a PoM (in time). In this case the Seller can request the payment.

(iii) The Seller responds incorrectly to a challenge, i.e., $\phi_i(\mathsf{in}_j) \neq \mathsf{outp}_j$. In this case the Buyer reveals the instruction of the gate ϕ_i and proves with a Merkle tree proof that this gate is part of the circuit ϕ. Now the judge contract can verify that the Seller misbehaved and sends the coins back to the Buyer.

(iv) The Seller sends a wrong witness. This case can happen, when one of the requested inputs is an element of the witness but the Seller responds with a different element x_i than the Buyer decrypted x'_i. In this case the Buyer will send the ciphertext value z_i to the contract and includes a Merkle tree proof that z_i is the i-th element of $\mathbf{z}$ (using r_z). Again the contract will be able to verify this cheating and reward the coins back to the Buyer.

(v) The last case occurs when the Buyer exhausts allowed challenge limit a_ϕ. This means the Buyer could not prove cheating of the Seller in time and the contract will send the payment to the Seller. The challenge limit a_ϕ is fixed for every circuit before the protocol start and denotes the maximum number of gates that the Buyer needs to challenge in order to prove cheating.

Challenge limit parameter. As mentioned before, the judge is responsible for limiting the number of possible challenges. To this end, the judge is initialized with a *challenge limit* parameter a_ϕ during the first protocol round as part of the auxiliary information aux. This parameter defines the maximal number of challenged circuit gates and it is important that both parties agree on it. The value must be large enough such that the Buyer is able to prove misbehavior for any possible gate or input value. At the same time it should be as small as possible to reduce the runtime and cost of a dispute.

3.4 Formal Description

In this section, we give a formal definition of the judge smart contract and the two protocol parties named Seller and Buyer. We formally prove the security of our construction within the universal composability framework in Appendix E.

In order to shorten the formal description of the judge smart contract and the protocol parties, parts of the computations are extracted to small algorithms. We denote these algorithms by expressive names such that the reader is able to follow our description. See Appendix D for some explanation of them.

We start the formal protocol description with the definition of the judge given below. It models a smart contract that interacts with both protocol parties, $\mathcal{S}$ and $\mathcal{B}$, as well as the global ledger $\mathcal{L}$ and a global random oracle $\mathcal{H}$. We present details about $\mathcal{L}$ and $\mathcal{H}$ in Appendix B. In comparison to the judge functionality used by FairSwap, $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ as defined below additionally contains functions for handling the challenge-response procedure as well as timeouts.

Judge hybrid functionality $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$

The judge stores two addresses $pk_{\mathcal{S}}$ and $pk_{\mathcal{B}}$, a commitment c, a decryption key k, Merkle tree root hashes r_z, r_e, and r_ϕ, the maximum number of challenged gates $a_\phi \in \mathbb{N}$, a price $p \in \mathbb{N}$, and the fee parameters $f_{\mathcal{S}}, f_{\mathcal{B}} \in \mathbb{N}$. Moreover, it contains a state s which is initially set to $s = start$ and it stores the most recent challenge query Q_r and the most recent response R_r. Additionally, $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ contains two timeout parameters $T_1, T_2 \in \mathbb{N}$ and corresponding variables $t_1, t_2 \in \mathbb{N}$, respectively.
All stored values depend on the session identifier id corresponding to one protocol execution. Hence, the values are functions evaluating on the session identifier, e.g., the state s on input id is initially set to $s(id) = start$. For the sake of simplicity, the input parameter id is omitted in the following description.

Initialization

On receiving $(initialize, id, c, r_z, r_e, r_\phi, a_\phi, p, f_{\mathcal{S}}, f_{\mathcal{B}})$ from $\mathcal{S}$ when $s = start$, store $pk_{\mathcal{S}}, c, r_z, r_e, r_\phi, a_\phi, p, f_{\mathcal{S}}$, and $f_{\mathcal{B}}$. Output $(active, id, c, r_z, r_e, r_\phi, a_\phi, p, f_{\mathcal{S}}, f_{\mathcal{B}})$ and set $s = active$.
On receiving $(accept, id)$ from $\mathcal{B}$ when $s = active$, store $pk_{\mathcal{B}}$ and send $(freeze, id, \mathcal{B}, p)$ to $\mathcal{L}$. If the response is $(frozen, id, \mathcal{B}, p)$, set $s = initialized$ and output $(initialized, id)$.

Abort

On receiving $(abort, id)$ from $\mathcal{S}$ when $s = active$, set $s = finalized$, and output $(aborted, id)$.
On receiving $(abort, id)$ from $\mathcal{B}$ when $s = initialized$, send $(unfreeze, id, \mathcal{B})$ to $\mathcal{L}$, set $s = finalized$, and output $(aborted, id)$.

Revealing

On receiving $(reveal, id, k, d)$ from $\mathcal{S}$ when $s = initialized$ and $\mathsf{Open}(c, k, d) = \mathsf{true}$, output $(revealed, id, k, d)$, set $t_1 = \mathsf{now}$, and $s = revealed$.

Challenge-Response

On receiving $(challenge, id, Q)$ from $\mathcal{B}$ when $s = revealed$ or $s = responded$, $|Q| \leq a_\phi$, and $\forall i \in Q : 1 \leq i \leq m$ send $(freeze, id, \mathcal{B}, f_{\mathcal{B}} * |Q|)$ to $\mathcal{L}$. If it responds with $(frozen, id, \mathcal{B}, f_{\mathcal{B}} * |Q|)$, set $Q_r = Q$, $a_\phi = a_\phi - |Q|$, $t_2 = \mathsf{now}$, output $(challenged, id, Q)$ and set $s = challenged$.
On receiving $(respond, id, R_q)$ from $\mathcal{B}$ when $s = challenged$, send $(freeze, id, \mathcal{S}, f_{\mathcal{S}} * |Q_r|)$ to $\mathcal{L}$. If it responds with $(frozen, id, \mathcal{S}, f_{\mathcal{S}} * |Q_r|)$, set $R_r = R_q$, $t_2 = \mathsf{now}$, output $(responded, id, R_q)$ and set $s = responded$.

Timeouts

On receiving $(challenge\ timeout, id)$ from $\mathcal{S}$ when $s = responded$ and $t_2 + T_2 \leq \mathsf{now}$, set $s = finalized$, send $(unfreeze, id, \mathcal{S})$ to $\mathcal{L}$, and send $(sold, id)$ to $\mathcal{S}$.
On receiving $(response\ timeout, id)$ from $\mathcal{S}$ when $s = challenged$ and $t_2 + T_2 \leq \mathsf{now}$, set $s = finalized$, send $(unfreeze, id, \mathcal{B})$ to $\mathcal{L}$, and send $(cancelled, id)$ to $\mathcal{B}$.

Finalizing

On receiving $(complain, id)$ from $\mathcal{B}$ when $s = responded$, set $s = finalized$ and do one of the following:
- If $\mathsf{ValidateResponse}(Q_r, R_r, r_e) = \mathsf{false}$ send $(unfreeze, id, \mathcal{B})$ to $\mathcal{L}$, send $(valid, id)$ to $\mathcal{B}$ and $(not\ sold, id)$ to $\mathcal{S}$.
- Otherwise, send $(unfreeze, id, \mathcal{S})$ to $\mathcal{L}$, send $(invalid, id)$ to $\mathcal{B}$ and $(sold, id)$ to $\mathcal{S}$.

On receiving $(prove, id, \pi)$ from $\mathcal{B}$ when $s = responded$, set $s = finalized$ and do one of the following:
- If $\mathsf{Judge}(k, r_z, r_e, r_\phi, \pi) = 1$ send $(unfreeze, id, \mathcal{B})$ to $\mathcal{L}$, send $(valid, id)$ to $\mathcal{B}$ and $(not\ sold, id)$ to $\mathcal{S}$.
- Otherwise, send $(unfreeze, id, \mathcal{S})$ to $\mathcal{L}$, send $(invalid, id)$ to $\mathcal{B}$ and $(sold, id)$ to $\mathcal{S}$.

On receiving $(finalize, id)$ from $\mathcal{B}$ when $s = revealed$, send $(unfreeze, id, \mathcal{S})$ to $\mathcal{L}$, set $s = finalized$ and send $(sold, id)$ to $\mathcal{S}$.
On receiving $(finalize, id)$ from $\mathcal{B}$ when $s = revealed$ and $t_1 + T_1 \leq \mathsf{now}$, send $(unfreeze, id, \mathcal{S})$ to $\mathcal{L}$, set $s = finalized$, and send $(sold, id)$ to $\mathcal{S}$.

We continue with the definition of an honest Seller and Buyer in the OptiSwap protocol. Both protocol parties are formally defined below.

Honest Seller and Buyer Description

The protocol description comprises a definition of the behavior of the honest Seller $\mathcal{S}$ and Buyer $\mathcal{B}$. Within the protocol, two timeouts $T_1, T_2 \in \mathbb{N}$ are used. These timeouts are defined and handled by the judge smart contract functionality $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$.

Initialization

$\mathcal{S}$: On receiving $(sell, id, \mathbf{x}, \phi, p, f_\mathcal{S}, f_\mathcal{B})$, where $\mathbf{x} = (x_1, \ldots, x_n) \in (\{0,1\}^\lambda)^n$, ϕ being a circuit with challenge limit property a_ϕ, $\phi(\mathbf{x}) = 1$, and $p, f_\mathcal{S}, f_\mathcal{B} \in \mathbb{N}$, $\mathcal{S}$ samples a key $k \leftarrow \mathsf{Gen}(1^\kappa)$, computes a commitment $(c, d) \leftarrow \mathsf{Commit}(k)$, and executes the presetup algorithm $(\mathbf{z}, r_z, r_e, r_\phi) \leftarrow \mathsf{Presetup}(\phi, \mathbf{x}, k)$. He sends $(initialize, id, c, r_z, r_e, r_\phi, a_\phi, p, f_\mathcal{S}, f_\mathcal{B})$ to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ and $(sell, id, \mathbf{z}, \phi)$ to $\mathcal{B}$. He continues in the *Revealing*-phase.

$\mathcal{B}$: On receiving (buy, id, ϕ), where ϕ being a circuit with challenge limit property a_ϕ, $\mathcal{B}$ checks if he received a message $(sell, id, \mathbf{z}, \phi)$ from $\mathcal{S}$ beforehand. Then, he computes $r_z = \mathsf{MTHash}(\mathbf{z})$ and $r_\phi = \mathsf{MTHash}Mtree(\phi)$. Upon receiving $(active, id, c, r_z, r_e, r_\phi, a_\phi, p, f_\mathcal{S}, f_\mathcal{B})$ from $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$, $\mathcal{B}$ responds with $(accept, id)$ to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ and continues in the *Revealing*-phase.

Revealing

$\mathcal{S}$: On receiving $(initialized, id)$ from $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$, $\mathcal{S}$ reveals his key generated in the *Initialization*-phase by sending $(reveal, id, k, d)$ to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$. He continues in the *Finish*-phase.
On receiving $(abort, id)$, $\mathcal{S}$ sends $(abort, id)$ to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ and continues in the *Finish*-phase.

$\mathcal{B}$: On receiving $(revealed, id, k, d)$ from $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$, $\mathcal{B}$ decrypts the witness $\mathbf{x}' = \mathsf{Dec}(k, \mathbf{z})$, outputs $(revealed, id, \mathbf{x}')$, and evaluates $\phi(\mathbf{x}')$. If the output is 1, he sends a $(finalize, id)$ message to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$, outputs $(bought, id, \mathbf{x}')$, and terminates the protocol. Otherwise, he sets the set of obtained responses $R := \emptyset$, creates a challenge query by executing $(Q, H') \leftarrow \mathsf{NextChallenge}(\phi, R, H)$, updates the most recent query $Q_r = Q$, and sends $(challenge, id, Q)$ to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$. $\mathcal{B}$ continues in the *Challenge-Response*-phase.
On receiving $(abort, id)$, $\mathcal{B}$ sends $(abort, id)$ to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ and continues in the *Finish*-phase.

Challenge-Response

$\mathcal{B}$: On receiving $(responded, id, R_q)$ from $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$, $\mathcal{B}$ computes $check = \mathsf{ValidateResponse}(Q_r, R_q, r_e)$. If $check = \mathsf{false}$, he sends $(complain, id)$ to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ and continues in the *Finish*-Phase. Otherwise, $\mathcal{B}$ adds the received responses to the set of all responses $R = R \cup R_q$ and computes $\pi = \mathsf{GenerateProof}(k, \phi, R)$. If $\pi \neq \mathsf{false}$, he sends $(prove, id, \pi)$ to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ and continues in the *Finish*-phase. Otherwise if $\pi = \mathsf{false}$, $\mathcal{B}$ generates the next challenge $(Q, H') \leftarrow \mathsf{NextChallenge}(\phi, R, H)$ and sends $(challenge, id, Q)$ to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$.
If no $(responded, id, \cdot)$ message was received after timeout T_2, $\mathcal{B}$ sends a $(response\ timeout, id)$ message to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ and continues in the *Finish*-phase.

$\mathcal{S}$: On receiving a $(challenged, id, Q)$ message from $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$, he generates the corresponding response by computing $R_q \leftarrow \mathsf{GenerateResponse}(\phi, \mathbf{x}, k, Q)$ and sends $(respond, id, R_q)$ to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$.
If no $(challenged, id, \cdot)$ message was received after timeout T_2, $\mathcal{S}$ sends a $(challenge\ timeout, id)$ message to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ and continues in the *Finish*-phase.

Finish

$\mathcal{S}$: On receiving $(sold, id)$, $(not\ sold, id)$, or $(aborted, id)$ from $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$, $\mathcal{S}$ outputs this message and terminates the protocol.
On receiving a $(challenged, id, Q)$ message from $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$, he generates the corresponding response by computing $R_q \leftarrow \mathsf{GenerateResponse}(\phi, \mathbf{x}, k, Q)$ and sends $(respond, id, R_q)$ to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$. He continues in the *Challenge-Response*-phase.
If no $(sold, id)$ message and no $(challenged, id, \cdot)$ message from $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ was received after timeout T_1, he sends a $(finalize, id)$ message to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$.

$\mathcal{B}$: On receiving $(invalid, id)$ from $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$, $\mathcal{B}$ outputs $(bought, id, \mathbf{x}')$ and terminates the protocol.
On receiving $(valid, id)$ or $(cancelled, id)$ from $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$, $\mathcal{B}$ outputs $(not\ bought, id, \mathbf{x}')$ and terminates the protocol.
On receiving $(aborted, id)$, $\mathcal{B}$ outputs this message and terminates the protocol.

3.5 Extensions and Optimizations

The challenge-response procedure is very expensive to run compared with the optimistic case execution. We distinguish three different dimensions that influence the cost for running the dispute: (1) The large *data storage* which consists of all values the Seller submits during dispute increases the transaction costs drastically. (2) The runtime of the sub-protocol mainly depends on the *number of challenge rounds* which increases with the depth of the circuit ϕ. (3) Additionally, the *complexity of the judge contract* also depends on ϕ. The main factor here is the size of the instruction set of the circuit, since the judge needs to be able to evaluate any possible gate. Depending on the circuit and the application, it can be smart to optimize the dispute procedure.

Reducing contract storage. Storing data in a contract is costly since it becomes part of the blockchain state and every single miner has to allocate space for this data. Therefore, the costs of running a contract can be reduced if storage is re-used. In the case of our judge contract, the storage requirements of the dispute sub-procedure can be optimized. Instead of storing all challenges and responses, it only stores the latest ones. This means the Seller needs to commit to all computed values using a Merkle tree root r_e computed over the intermediate values of the evaluation $\phi(x)$. Now, for every revealed element the Seller includes a Merkle proof about the item, which is verified and stored by the Buyer. This data allows him to generate a proof including elements which have been revealed in a previous round but are no longer stored in the contract.

Reducing round complexity. The worst case round complexity is fixed for every circuit. In order to reduce the maximal and average number of challenge rounds the parties could agree to challenge and reveal outputs of multiple gates at the same time. Depending on the circuit they could also agree on different challenge strategies which can again help to reduce this parameter. For circuits that are very balanced and have a tree like structure it makes a lot of sense to start with the last gate and always query one predecessor gate, for which the Seller provided a divergent output. This basic strategy results in an upper bound for the overall round complexity in case of dispute of $4 + 2 * ((\delta - 1)\ell + 1)$ rounds. For very deep circuits with a low width ω this strategy is not optimal but instead a binary search will be better suited (cf. Figure 1). This strategy results in an upper bound of $4 + 2 * (\lceil \log_2 \delta \rceil * \omega + 1)$ rounds. Knowing the circuit-dependent parameters δ, ℓ and ω, the more efficient strategy can be chosen. Both strategies can even be combined for optimal results. Another option is to combine multiple gates into one by defining new combined instructions. This can change the overall structure of the circuit and lower its challenge costs, creating trade-off between round complexity and contract complexity.

Reducing contract complexity. For complicated circuits with many instructions the logic of the judge contract can get very complex. Since the majority of its logic is only required for dispute handling and evaluating the PoM, the contract can be split into two parts. This is an approach which has been introduced in [27]. It allows to deploy the dispute logic only in case the Buyer triggers the dispute. This trick drastically reduces the deployment costs and, hence, lowers the costs of the optimistic case.

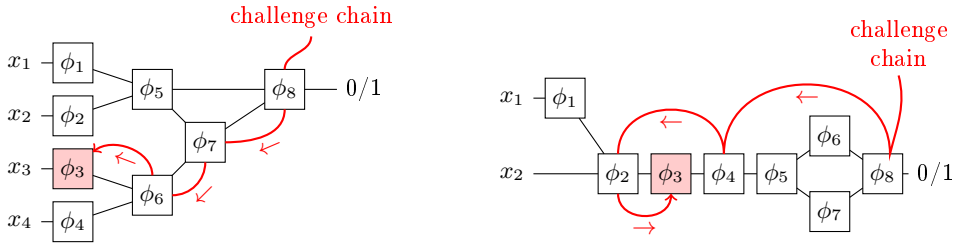

Fig. 1. Back-to-front (left) and binary search (right) challenge strategies.

To generalize over all improvements made in this section we will abstract from specific challenge strategies and encapsulate this into a function NextChallenge that selects the next challenge query based on the circuit and the previous response (more on this in Appendix C and D).

At this point we have presented most features of OptiSwap and can give a full list of all required auxiliary information $\mathsf{aux} = (c, r_z, r_e, r_\phi, a_\phi, f_\mathcal{S}, f_\mathcal{B}, T_1, T_2)$. The parameter r_ϕ denotes a commitment to the verification circuit ϕ and $f_\mathcal{S}, f_\mathcal{B}$ being the fee parameter, which will be explained in the next section. Moreover, aux contains two timeout parameters T_1 and T_2.

4 OptiSwap Evaluation and Transaction Fees

In this section we evaluate the efficiency of our protocol. We start with a discussion about the communication complexity in the honest execution, which is the main improvement of OptiSwap compared to FairSwap. Then, we analyze the runtime and costs of our protocol. We also discuss how we incorporated transaction fees into OptiSwap to prevent grieving. In order to illustrate the evaluation of our protocol, we consider the file sale application which we describe in the following.

File sale application. For exact measurements and better comparison we implemented[3] the file sale application of [14] as a concrete instantiation of circuit ϕ. In this case the witness is a file $\mathbf{x}$ consisting of n chunks $x_1, \ldots x_n$ of size $\lambda = 512$ Byte and it is identified via its Merkle hash $h = \mathsf{MTHash}(\mathbf{x})$. The verification circuit ϕ computes the Merkle hash of the input and compares it with the expected value, i.e., $\phi(\mathbf{x}) = 1 \Leftrightarrow \mathsf{MTHash}(\mathbf{x}) = h$. The required instruction set consists of a Hash function evaluation and one check-if-equal instruction.

4.1 Communication Complexity in the Optimistic Execution

Table 1. Communication overhead in FairSwap [14] for different applications.

Application	Witness size $\lvert\mathbf{x}\rvert$	Encoding size $\lvert\mathbf{z}\rvert$ in [14]	Communication overhead $\frac{\lvert\mathbf{z}\rvert}{\lvert\mathbf{x}\rvert}$ in [14]
file sale (32 Bytes chunk size)	1 GByte	2 GBytes	2
file sale (4 Bytes chunk size)	1 GByte	9 GBytes	9
matrix multiplication (2×2)	256 Bytes	864 Bytes	3.38
matrix multiplication (10×10)	6 400 Bytes	73 568 Bytes	11.5
AES-256	32 Bytes	1 088 000 Bytes	34 000

[3] github.com/CryBtoS/OptiSwap

The goal of OptiSwap is to improve the optimistic execution. Assuming both Seller and Buyer behave honestly, the transferred witness $\mathbf{x}$ is correct and unlike in FairSwap there is no necessity to exchange information about the predicate evaluation $\phi(\mathbf{x})$ in order to prove misbehavior. Concretely, in the optimistic execution of OptiSwap, most of the protocol messages have a constant size (independent of the witness). Only in the first round, the data transferred from Seller to Buyer depends on the size of the witness $\mathbf{x}$. This in particular means that compared to previous solutions like FairSwap [14], the size of this message is independent of the circuit ϕ. Thus, OptiSwap removes the overhead of FairSwap from the first protocol message, where this overhead is given as the ratio of the message size $|\mathbf{z}|$ to the size of the witness $|\mathbf{x}|$. To show the advantage of our OptiSwap protocol compared to FairSwap, we consider three different use cases and show the resulting communication overhead in FairSwap. We start by taking a look at the overhead in the file sale application. Since our protocol can also be used for arbitrary applications that are based on an arithmetic or a Boolean verification circuit, we then analyze the communication overhead for matrix multiplication as well as an AES-256 bug bounty application. Table 1 lists the communication overhead of the first protocol message in FairSwap for the different applications. We stress that the overhead as defined above in OptiSwap is always 1, which basically means there is no overhead.

First, we consider the file sale application as described above. For the file sale application it holds that the size of the verification circuit depends on the size of a single file chunk. The smaller the file chunks, the larger the circuit size and the more data has to be transferred in the first protocol message in FairSwap. Table 1 shows that the overhead increases for the file sale application if the size of each file chunk decreases. This is a drawback since a small file chunk size is preferable over a big one as we will see in the following. In case of dispute, the Buyer is responsible for creating a PoM. Assuming the Seller sent an incorrect witness, the PoM might contain a small part of the file in order to show that its hash results in a different hash value than computed by the Seller. Since the PoM is sent to the judge smart contract, the data is on-chain and, hence, publicly visible. This data leakage increases in the size of the file chunks. An additional drawback is that more storage on the Blockchain results into higher gas costs. In contrast, the smaller the file chunk size the less data needs to be published in the worst case. Using a file chunk size of 1 Byte, the size of the encoding in the FairSwap protocol is 33 times the size of the witness.

It is easy to see that this overhead is way too high for large witnesses, e.g., 1 GByte files, therefore, it is unreasonable to use small file chunks in the FairSwap protocol. On the contrary, OptiSwap does not suffer from the overhead and, hence, the file chunk size can be chosen arbitrarily small. It is important to note that a small file chunk size also results in a higher circuit depth leading to an increased number of rounds in the dispute resolution. However, halving the file chunk size increases the circuit depth only by one.

Second, we consider matrix multiplication as an example for arithmetic circuits. In this scenario, the Buyer asks for two quadratic matrices whose product is equal to a predefined result. We assume that each element in one of the two input matrices is represented by 32 Bytes[4]. The resulting communication overhead increases with the parameter n for $n \times n$-matrix multiplication. Table 1 lists the concrete overhead for $n = 2$ and $n = 10$. If we use larger values for n, the overhead will be even more significant.

As a third use case for evaluating the overheads of our solution compared to FairSwap we consider a bug bounty scenario. Here, the Buyer offers a bounty for a Seller if Seller can break AES-256 by providing an input to the encryption algorithm such that the output is equal to a predefined ciphertext. We use a Boolean circuit of the AES-256 cipher containing 34 000 gates for the analysis. Since the circuit size is high compared to the input size, the overhead becomes very huge. Instead of just transferring the 32 Byte input, the encoding size is greater than 1 MByte. Although the output of a gate within a Boolean circuit is just one bit, we consider the encryption scheme as proposed in [14] where each ciphertext has a

[4] This memory size is based on the integer representation in Solidity.

size of 32 Byte. This can possibly be improved using a different encryption scheme that can efficiently be realized on Ethereum.

We summarize that our OptiSwap protocol is always preferable over the FairSwap protocol in case of two honest parties. The advantage increases for applications with a small witness size and a high verification circuit size. Next, we analyze the gas costs of our reference implementation of the file sale application. We show that our protocol is affordable and costs are roughly the same as for the FairSwap protocol.

4.2 Runtime and Gas Costs

Table 2. Gas cost for the file sale application in OptiSwap.

Standalone		
deploy	2 273 398 gas	$ 5.94
optimistic case	101 307 gas	$ 0.26
pessimistic case	6 412 569 gas	$ 16.77

Split Contract		
optimistic deploy	952 939 gas	$ 2.49
pessimistic deploy	1 962 992 gas	$ 5.13

In Ethereum, transaction fees are paid in *gas* and every instruction of the Ethereum virtual machine code has a gas value assigned to it. This gives a deterministic gas amount for every transaction. The total amount of gas required for a transaction can be used as efficiency indicator. The gasprice[5] can be set individually for every transaction and describes the exchange rate between Ether and gas. We assume that the gasprice will be fixed for the duration of the protocol run.

We compare OptiSwap against the FairSwap protocol implementation of the file sale of 1 GByte with 512 Byte-sized file chunks. In this setting, the circuit depth is $\delta = 21$ and the max fan-in is $\ell = 2$. When we analyze the costs for the OptiSwap protocol we distinguish between deployment costs and the execution costs in the optimistic and pessimistic case (cf. Table 2). We also analyze an optimized version of the contracts which we call split contract. Here we apply the idea from [27], where we create two versions of the judge contract. The first part is responsible for the execution in the optimistic case and the second part only for the dispute. The key idea is that we only need to deploy the second contract in case the parties start the dispute. This reduces the costs for the purely optimistic case. In case of the file sale application the complex logic for the interactive dispute can be encapsulated in the dispute contract which makes the optimistic case even slightly cheaper compared to FairSwap. The authors of [14] claim deployment costs of 1 050 000 gas for the file sale implementation of the FairSwap-protocol. It is reasonable that this value is much less than the deployment costs of the standalone version of OptiSwap, since the challenge-response functionality is not required in FairSwap. However, our optimized split contract implementation shows that the deployment costs for an optimistic execution can be even further reduced. A more extensive analysis and comparison of gas costs is given in Appendix F.

Optimistic case execution Without a complaint the protocol runs in either 4 or 5 rounds, depending on the Buyer. In comparison to FairSwap, the first message of OptiSwap does not contain information about $\phi(\mathbf{x})$ and the Seller does not have to compute $\phi(\mathbf{x})$ in this case. This reduces both the message and computation complexity. The biggest part of transaction

[5] We consider an average gas price of 14.4 GWei and an exchange rate of 181.57 USD per ether. This data is taken from https://etherscan.io on November 18, 2019.

fees in this case are the deployment fees (cf. Table 2) which are paid by the Seller in the first round[6]. The costs for deployment and protocol fees of the Seller can be taken into account when defining the price for the digital good. But recall, that the optimistic case can end in two ways; either the Buyer confirms that he received $\mathbf{x}$ and triggers the payout, or the Seller has to request it after the Buyer did neither confirm nor complain. Fee security for the Seller can be achieved by letting the Buyer lock a small security deposit in the second round. The Seller will receive this deposit on top of the payment unless the Buyer correctly confirms the exchange in round 4; in which case he gets this deposit back.

Pessimistic case execution On top of the four rounds of the honest execution, the parties execute the interactive dispute procedure. This includes at most a_ϕ challenge-response repetitions. We used the basic challenge-response strategy resulting in $a_\phi = ((\delta - 1) * \ell + 1) = ((21 - 1) * 2 + 1) = 41$. This also means that the costs for the pessimistic case execution grow significantly due to the higher transaction load (cf. Table 2). Contrary, the interactive dispute procedure is not required in FairSwap, since all the data is already transferred in the first protocol message. Therefore, FairSwap requires only one transaction from the Buyer providing the PoM costing at most 194 068 gas. Comparing to FairSwap, the number of rounds and the costs of the dispute resolution can be considered as a downside of OptiSwap. However, we like to stress that an honest party always gets compensated at the end of the protocol. This also implies a deterrent effect on malicious parties.

Within the challenge-response procedure, the Buyer will send a relatively small challenge query but forces the Seller to respond with a potentially large transaction. For our implementation of the file sale application, a challenge transaction costs at most 70 057 gas and a response costs at most 645 331 gas. While the maximum number of rounds is fixed, it will be hard in most cases to precisely predict the costs for the transactions. It would be easy for the Buyer to grieve the Seller – force him (at a low cost) to pay a lot of transaction fees to show his correct behavior. The transaction fees are many times higher than the honest execution and could even outweigh the price p. A rational Seller could be forced this way to waive the payment in order to save transaction fees.

To prevent grieving attacks, we present a simple security fee mechanism that should guarantee that an honest party does not carry the costs for the dispute. In particular, the Seller should be compensated if the Buyer cannot prove misbehavior and the Buyer should be compensated if the Seller cheated. To ensure this, both parties need to add some additional coins to each transaction executed in the challenge-response procedure. All coins will be collected in the contract and paid out together with the payment. This ensures that the honest party will receive its deposit back and gets the malicious party's deposit as compensation.

Since the response transactions of the Seller are more expensive and the Buyer might need to compensate for them, his deposit $f_\mathcal{B}$ will be higher than the one of the Seller $f_\mathcal{S}$. The parameters depend on the circuit and are contained in the auxiliary information aux. The value must be large enough to compensate for the worst case transaction size. As a result, it might happen that one or both parties deposit much higher fees than the actual costs for the blockchain transactions. However, since the honest party gets his complete money back, this aspect just increases the temporal deposit costs. In addition, it provides a highly deterrent effect for malicious party and disincentives cheating. We give concrete values for $f_\mathcal{B}$ and $f_\mathcal{S}$ for an exemplary file sale application in the Appendix F.

We note that the Buyer can still grieve the Seller by forcing him to pay the deployment fees in the first round. The Seller pays the deployment fees with the risk that the Buyer aborts and does not send his message in the second round. At this point there is no way to force the Buyer to compensate the Seller for the invested fees.

[6] Without loss of generality the deployment costs could also be carried by the Buyer in the second round, if he gets the auxiliary information from the Seller first.

5 Security Analysis

We formally prove the security of OptiSwap in the Universal Composability (UC) framework (cf. Appendix A for an introduction to the UC model and Appendix E for the full proof). We start by informally arguing about the security properties to give the reader a intuition about the achieved security of the OptiSwap protocol. Then we define the ideal behavior of OptiSwap by the ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ which captures the overall security of our construction. Afterwards we present a short sketch of our security proof.

5.1 Informal Security Discussion

In this section we will informally argue why OptiSwap achieves all security and efficiency properties. In every step of the OptiSwap protocol the parties only have a limited time to send an expected transaction. On one hand this limits the maximal round time while on the other hand it identifies if a party aborted. If one of the parties aborts, the other party calls a timeout-function in the contract. This triggers the judge to verify that the timeout indeed expired and it will terminate after it sent all remaining coins to the opponent. It is important for the security of honest parties that this timeout is chosen large enough, such that honest transactions will be included in the blockchain in time.

Security against malicious Buyers (S1). This property guarantees to the honest Seller that he will get paid if the Buyer learns the witness x. In particular it means that (a) the Buyer learns nothing about x before k is revealed and (b) he cannot forge a proof of misbehavior if the Seller behaves honestly. (a) holds because of the CPA-secure encryption, which guarantees that the ciphertext $\mathbf{z}$ does not leak information about the plaintext $\mathbf{x}$ and the hiding property of the commitment which hides k from the Buyer. This makes it impossible for the Buyer to learn $\mathbf{x}$ without breaking any of the two schemes. (b) is guaranteed because the honest Seller will not cheat with the computation of $\phi(\mathbf{x})$ which makes it impossible for the Buyer to claim this (since he cannot find collisions for hash values in the Merkle tree commitment of $\mathbf{z}$). Additionally, E3 ensures that the Buyer cannot delay the payout forever.

Security against malicious Sellers (S2). This property protects the Buyer and his funds. It guarantees that – no matter what the Seller does – the Buyer will either receive the correct witness $\mathbf{x}$ or get his money back. If the Seller aborts or reveals a wrong key the contract will automatically reimburse the Buyer. This follows from the fact that it is cryptographically hard to break the binding of the commitment c. If the Seller reveals the correct key and the witness is correct, property S2 is also satisfied. So the most interesting case for the analysis is the situation when the Buyer received an incorrect witness $\mathbf{x}' \neq \mathbf{x}$ s.t. $\phi(\mathbf{x}') = 0$. In this case the interactive dispute procedure guarantees that the Buyer will be able to find a statement (PoM) which will make the judge refund the payment to him. Unless the Seller aborts, this statement will either prove that $\phi(\mathbf{x}') = 0$, that the Seller lied about the result of some gate $\phi_i \in \phi$ or that he used a witness $\mathbf{x} \neq \mathsf{Dec}(k, \mathbf{z})$. This guarantee follows from the fact that the Buyer has enough rounds in the dispute resolution to challenge the gates required for creating a PoM and that Seller is forced by the judge to respond to all queries.

Security against grieving (S3). This property protects both parties. As seen in Section 4.2, OptiSwap protects against grieving attacks by using transaction deposits for every step in the interactive dispute phase. These deposits will be paid out to the Buyer if he finds a PoM and to the Seller if no PoM can be provided. From properties S1 and S2 we know that the honest party will always receive the coins. This guarantees to honest parties that all paid dispute fees (and more) will be reimbursed after the dispute has finished.

Communication complexity (E1). The communication complexity in the optimistic case of OptiSwap is mainly determined by the first message of the protocol. All following contract transactions have a fixed size that are independent of the witness $\mathbf{x}$ and the circuit ϕ. The first message has a size of $|\mathbf{z}|$ which is identical to $|\mathbf{x}|$ since it is its encryption. Therefore, the overall communication complexity is $\mathcal{O}(\mathbf{x})$ which satisfies property (E1). This is in contrast to FairSwap [14] where the first message depends on both, the witness and the circuit. In the pessimistic case the message complexity is depended on ϕ. A challenge consists of a single integer while a response contains the output value of a gate along with a Merkle tree proof. The size of a gate output value is at most the size of a witness chunk $\frac{|\mathbf{x}|}{n}$. Hence, the size of a response is at most $\frac{|\mathbf{x}|}{n} + (1 + \log(m))$. The number of challenges is limited by a_ϕ resulting in an overall complexity of a_ϕ integers for the Buyer and $a_\phi \times (\frac{|\mathbf{x}|}{n} + (1 + \log(m)))$ values for the Seller in the worst case. Note that in this case (S3) guarantees that any honest party forced to engage in this procedure, will get compensated.

Round complexity (E2+E3). In the optimistic case the protocol runs either 4 or 5 rounds (in case the Buyer doesn't confirm that he received the witness). This satisfies (E2). In the pessimistic case (E3) the challenge-response procedure is started in the fourth round, and will take at most a_ϕ repetitions and finish with an additional PoM transaction of the Buyer. In case the Buyer does not send this last message, the Seller has to request his payment in the next round, leading to $5 + 2a_\phi$ rounds in the worst case.

5.2 Ideal Functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$

The functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ represents a fair exchange of digital goods within a blockchain-based setting. It describes an exchange of a digital good $\mathbf{x}$ between a Seller $\mathcal{S}$ and a Buyer $\mathcal{B}$. While Seller offers the digital good, Buyer must pay for it. The correctness of the witness $\mathbf{x}$ is defined by a circuit ϕ which either outputs 1 for the correct witness or 0 otherwise. It extends the functionality $\mathcal{F}_{\mathsf{cfe}}^{\mathcal{L}}$ for coin aided fair exchange [14] and also utilizes an idealized ledger functionality $\mathcal{L}$ for the on-chain handling of coins. We do not specify the ledger ideal functionality $\mathcal{L}$ here but refer the reader to Appendix B. The major change in comparison to $\mathcal{F}_{\mathsf{cfe}}^{\mathcal{L}}$ is the modeling of an interactive challenge-response procedure in case of dispute. This includes modeling security fees incorporated in our construction.

Ideal Functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$

The ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ (in session id) for interactive coin aided fair exchange interacts with Seller $\mathcal{S}$ and Buyer $\mathcal{B}$. Moreover, it has access to the global ledger functionality $\mathcal{L}$ and interacts with the ideal adversary Sim.

Initialization

(Round 1) Upon receiving $(sell, id, \mathbf{x}, \phi, p, f_\mathcal{S}, f_\mathcal{B})$ with $p, f_\mathcal{S}, f_\mathcal{B} \in \mathbb{N}$ and $\phi(\mathbf{x}) = 1$ from $\mathcal{S}$, store witness $\mathbf{x}$, circuit ϕ with challenge limit property a_ϕ, price p, and fee parameters $f_\mathcal{S}, f_\mathcal{B}$ and leak $(sell, id, \phi, p, f_\mathcal{S}, f_\mathcal{B}, \mathcal{S})$ to Sim.
Upon receiving $(sell\text{-}fake, id, \mathbf{x}, \phi, p, f_\mathcal{S}, f_\mathcal{B})$ with $p, f_\mathcal{S}, f_\mathcal{B} \in \mathbb{N}$ from corrupted Seller $\mathcal{S}^*$, store witness $\mathbf{x}$, circuit ϕ with challenge limit property a_ϕ, price p, and fee parameters $f_\mathcal{S}, f_\mathcal{B}$.
(Round 2) Upon receiving $(abort, id)$ from Seller $\mathcal{S}$, leak $(abort, id, \mathcal{S})$ to Sim, send $(aborted, id)$ to $\mathcal{B}$, and terminate.
Upon receiving (buy, id, ϕ) from Buyer $\mathcal{B}$, send $(freeze, id, \mathcal{B}, p)$ to $\mathcal{L}$. If $\mathcal{L}$ responds with $(frozen, id, \mathcal{B}, p)$, leak $(buy, id, \mathcal{B})$ to Sim and go to *Revealing* phase.
If no message is received during round 2, terminate.

Revealing

(Round 3) Upon receiving $(abort, id)$ from Buyer $\mathcal{B}$, leak $(abort, id, \mathcal{B})$ to Sim, send $(unfreeze, id, \mathcal{B})$ to $\mathcal{L}$, $(aborted, id)$ to $\mathcal{S}$, and terminate.
Upon receiving $(abort, id)$ from corrupted Seller $\mathcal{S}^*$ in round 3, send $(unfreeze, id, \mathcal{B})$ to $\mathcal{L}$ in the next round and terminate.
If no message is received in round 3, send $(revealed, id, \mathbf{x})$ to $\mathcal{B}$, set $s = challenge$, wait one round, and go to *Interaction* phase.

Interaction

Upon receiving $(freeze, id, \mathcal{B}, a_r)$ with $a_\phi \geq a_r$ from Sim when $s = challenge$, send $(freeze, id, \mathcal{B}, a_r * f_\mathcal{B})$ to $\mathcal{L}$. If $\mathcal{L}$ responds with $(frozen, id, \mathcal{B}, a_r * f_\mathcal{B})$, store a_r, set $a_\phi = a_\phi - a_r$, $s = response$, and wait one round.
Upon receiving $(freeze, id, \mathcal{S}, a_s)$ with $a_s = a_r$ from Sim when $s = response$, send $(freeze, id, \mathcal{S}, a_s * f_\mathcal{S})$ to $\mathcal{L}$. If $\mathcal{L}$ responds with $(frozen, id, \mathcal{S}, a_s * f_\mathcal{S})$, set $s = challenge$ and wait one round.
Upon receiving $(abort, id, \Delta)$ from corrupted Buyer $\mathcal{B}^*$, where $\Delta \in \{0, 1\}$, when $s = challenge$, wait Δ rounds. Then send $(unfreeze, id, \mathcal{S})$ to $\mathcal{L}$ and $(sold, id)$ to $\mathcal{S}$ and terminate.
Upon receiving $(abort, id, \Delta)$ from corrupted Seller $\mathcal{S}^*$, where $\Delta \in \{0, 1\}$, when $s = response$, wait Δ rounds. Then send $(unfreeze, id, \mathcal{B})$ to $\mathcal{L}$ and $(not\ bought, id, \mathbf{x})$ to $\mathcal{B}$ and terminate.
If no message is received, execute *Payout* phase.

Payout

If $\phi(\mathbf{x}) = 1$, send messages $(unfreeze, id, \mathcal{S})$ to $\mathcal{L}$, $(sold, id)$ to $\mathcal{S}$, and $(bought, id, \mathbf{x})$ to $\mathcal{B}$. Otherwise, if $\phi(\mathbf{x}) \neq 1$, send messages $(unfreeze, id, \mathcal{B})$ to $\mathcal{L}$, $(not\ sold, id)$ to $\mathcal{S}$, and $(not\ bought, id, \mathbf{x})$ to $\mathcal{B}$. Terminate the execution after the payout.

We describe the ideal functionality in the following. We first assume two honest parties and describe additional opportunities for malicious parties afterwards.
The ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ starts in the *initialization* phase during which both parties provide their initial input. In the first round, Seller starts the execution by sending the witness $\mathbf{x}$, the verification circuit ϕ, a price p, and fee parameters $f_\mathcal{S}, f_\mathcal{B}$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. For an honest Seller it must hold that $\phi(\mathbf{x}) = 1$ in the initialization. In the second round, Seller may abort the execution or he waits for Buyer to accept the offer. By accepting the offer, the ideal functionality instructs the ledger functionality to lock p coins from Buyer $\mathcal{B}$. If locking p coins from $\mathcal{B}$ is not successful due to insufficient funds, the functionality ends the interactive fair exchange protocol.

In the *revealing* phase in round 3, Buyer may abort the protocol execution and receives his coins back or he waits to learn the witness $\mathbf{x}$. After the witness is revealed, the *interaction* phase is started. Assuming both parties are honest, Seller sent the correct witness and Buyer has no intention to challenge Seller. Therefore, the *payout* phase is executed during which Seller gets the money and the fair exchange is completed.

Next, considering the case that Seller is malicious. He can send an incorrect witness $\mathbf{x}$ such that $\phi(\mathbf{x}) \neq 1$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the *initialization* phase. Since a malicious Seller has not to follow the protocol, he may abort in round 3 by not sending the required message. If he aborts in round 3 before the witness is revealed to Buyer $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ instructs the ledger functionality $\mathcal{L}$ to unfreeze the locked money in Buyer's favor. In case Seller does not abort, $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ reveals the incorrect witness to honest Buyer. Since an honest Buyer is not willing to pay for an incorrect witness, he starts the dispute resolution sub-protocol by challenging Seller. This challenge-response interaction is simulated by the ideal adversary *Sim* within the *interaction* phase. Each challenge or response message costs some fees which are locked by $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ by instructing $\mathcal{L}$ to freeze these fees. At each round during the *interaction* phase in which Seller must provide a response to a challenge query a malicious Seller may abort. In this case, Buyer gets all the money that is locked by the ledger functionality. After several interactions, which is limited by the challenge limit property a_ϕ of the circuit ϕ, Buyer is able to generate a valid proof of misbehavior within the protocol. The ideal functionality executes the *payout* phase and sends all the money to $\mathcal{B}$, since malicious Seller offered an incorrect witness at the start of the execution.

In addition to the already explained actions, a malicious Buyer may challenge Seller even tough he provided the correct witness during the *initialization* phase. These challenges are simulated by the ideal adversary in the *interaction* phase. Again, the number of possible challenges is limited by the challenge limit property a_ϕ of the circuit ϕ. As explained above, a malicious Seller may abort within the *interaction* phase and a malicious Buyer may do this as well. In this case, all the locked money gets unfrozen by the ledger functionality in favor of the Seller. If Seller has sent the correct witness in the *initialization* phase, a malicious Buyer will not be able to generate a valid proof of misbehavior and Seller is able to respond to every challenge posed by Buyer. This ends in the execution of the *payout* phase during which the ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ instructs $\mathcal{L}$ to unfreeze all the money in favor of $\mathcal{S}$.

In order to consider delayed messages during a protocol execution, the ideal adversary *Sim* is able to delay the execution of $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. Whenever *Sim* obtains the instruction from the environment to delay the message of any corrupted party by δ rounds, the simulator instructs the ideal functionality to delay the execution by δ rounds. For sake of clarity, this description is omitted in the definition.

5.3 Proof Sketch

Here, we only provide a sketch of the security proof and refer the reader to the Appendix E for a formal proof. We follow the model of FairSwap [14] and assume synchronous communication and static corruption. As we work in the generalized universal composability (GUC) framework [11] (cf. Appendix B for definition of ledger $\mathcal{L}$ and random oracle $\mathcal{H}$) the formal security statement is as follows:

Theorem 1. *The* OptiSwap *protocol Π stated in Section 3.4 GUC-realizes the ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ within the judge smart contract $(\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}, \mathcal{L}, \mathcal{H})$-hybrid world, where $\mathcal{L}$ denotes a ledger functionality and $\mathcal{H}$ is modeled as a global random programmable oracle.*

To prove this theorem, we need to show that for any possible corruption case, the environment cannot distinguish the ideal world execution from the execution of the OptiSwap protocol. In particular, the outputs of honest parties and all changes in the global functionalities ($\mathcal{L}$, $\mathcal{H}$) must be identical. This must also hold for all messages sent to the corrupted parties. We prove this by constructing a simulator *Sim* which simulates the execution of OptiSwap in the real world, by interacting with the ideal world functionality and corrupted parties in the ideal world. In the following we highlight the role of the simulator in all four corruption cases.

Two honest parties. In case of two honest parties, we need to assume a secure channel between the Seller and the Buyer that is used for transferring the first protocol message. In

this secure channel, the adversary only learns that information is transferred but he gains no knowledge about the content of the message. Without such a secure channel, the simulator would have to create an encrypted witness $\mathbf{z}^*$ without ever learning the correct witness $\mathbf{x}$.

Since both parties are honest, the execution of the protocol only starts if the witness is correct, i.e., $\phi(\mathbf{x}) = 1$. This is ensured by the honest Seller in the hybrid world execution and by the ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the ideal world execution. Therefore, the environment obtains the correct witness $\mathbf{x}$ back after the execution in both worlds and cannot use the input in order to distinguish between the two executions. The commitments created by the honest Seller in the first protocol message depend on a randomly chosen encryption key. Based on the randomly selected keys and the hiding property of the commitment scheme, the environment cannot distinguish between the values computed by the honest Seller and the one created by the simulator.

Finally, a simulation of abort is possible in a straightforward way and the simulator ensures that money is locked and unlocked in the same round as in the real world execution.

Corrupted Seller. For the proof of Theorem 1 in case of a corrupted Seller, we make use of the hybrid argument technique and show indistinguishability by a sequence of security games. The main challenge in this corruption scenario is that the simulator has to provide input to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ on behalf of the corrupted Seller without knowing the witness $\mathbf{x}$. The simulator only learns the encrypted witness $\mathbf{z}$ and the commitment to the key c in the first round. We use the observability feature of the programmable random oracle $\mathcal{H}$ at this point. We need to distinguish two cases of how the environment creates the commitment c:

- c is constructed correctly by querying $\mathcal{H}$: in this case, we use the observability feature of $\mathcal{H}$ to obtain the key k. This allows the simulator to decrypt $\mathbf{z}$ and provide the witness $\mathbf{x}$ as input to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. Only if the environment finds a collision of $\mathcal{H}$, he can distinguish between the two executions. However, based on the binding property of the commitment scheme, this happens only with negligible probability.
- c is constructed incorrectly: in this case $\mathcal{H}$ was not queried before, so the simulator cannot take advantage of the observability feature. However, the environment has to provide correct opening values k and d in round 3 of the protocol, which he can only guess with probability $\frac{1}{2^\mu}$. This is negligible for large μ.

Finally, the simulator has to simulate the interactive dispute resolution procedure.

Corrupted Buyer. The main challenge of the simulation in case of a corrupted Buyer is that the simulator needs to create an encrypted witness $\mathbf{z}^*$ in the first round and to present an encryption key in the third round such that the decryption of $\mathbf{z}^*$ equals the correct witness $\mathbf{x}$ which is known to the simulator only after round 3. To achieve indistinguishability, we make use of the programming feature of $\mathcal{H}$. The encrypted witness $\mathbf{z}^*$ is chosen randomly by the simulator. Based on the IND-CPA-security of the encryption scheme, the environment cannot distinguish between the encrypted witness $\mathbf{z}$ created by the honest Seller and $\mathbf{z}^*$. After the correct witness $\mathbf{x}$ is revealed in the third round, the simulator programs $\mathcal{H}$ such that the decryption of the $\mathbf{z}^*$ equals $\mathbf{x}$.

Although the honest Seller provides a correct witness $\mathbf{x}$, a corrupted Buyer may start the dispute resolution procedure. Therefore, the simulator needs to simulate the interaction. He keeps track of the challenge limit parameter to ensure that the corrupted Buyer may not pose more challenges than allowed. Moreover, the honest Seller is always able to create valid responses and the simulator has enough information to simulate the same behavior.

Two corrupted parties. In case of two corrupted parties, the OptiSwap protocol does not provide any guarantees. In particular, termination is not given and coins may be locked forever. However, the simulation in this corruption scenario need to be shown in order to allow composability.

The simulation is a combination of the single corruption cases and in most aspects straightforward. One difference is the possibility that money is blocked in the judge smart contract. To achieve the same behavior in the simulation, we make use of the blocking feature of the ledger functionality $\mathcal{L}$. This feature allows the simulator to block an unfreezing of coin on behalf of a corrupted party. The result is that the money is locked forever.

6 Conclusion and Future Work

We presented OptiSwap, a smart contract based two-party protocol for realizing a fair exchange for digital commodity against money. In comparison to already existing fair exchange protocols, we significantly improved the execution of the optimistic case in which both parties behave honestly. We integrated an interactive dispute handling in OptiSwap that is only run in the pessimistic case. This allows us to have almost no overhead in communication complexity in the optimistic case.

Furthermore, OptiSwap contains a protection mechanism against so-called grieving attacks, where the attacker tries to harm the honest party by forcing him to pay fees. The protection is based on transaction fees paid by both parties such that the honest party is reimbursed at the end of the protocol execution. We provide a reference implementation of our judge smart contract for Ethereum. Based on that, we estimated gas costs for the optimistic and pessimistic case.

Future work. The execution costs in the optimistic mode strongly depend on the deployment costs of the judge smart contract. By allowing repeated exchanges over one contract, the costs for a single execution might be significantly reduced. An interesting research question is the analysis of minimal bounds on communication and computation complexity. Results can show how far further our construction might be improved.

7 Acknowledgments

This work was partly funded by the iBlockchain project (grant nr. 16KIS0902) funded by the German Federal Ministry of Education and Research (BMBF), by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SFB 1119 – 236615297, and by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

References

1. Ethereum average gas price chart. https://etherscan.io/chart/gasprice. (Accessed on 09/19/2019).
2. Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 443–458. IEEE Computer Society Press, May 2014.
3. N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures (extended abstract). In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 591–606. Springer, Heidelberg, May / June 1998.
4. Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. LNCS, pages 324–356. Springer, Heidelberg, 2017.
5. Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 421–439. Springer, Heidelberg, August 2014.
6. Bitcoin Wiki. Zero knowledge contingent payment. https://en.bitcoin.it/wiki/. (Accessed on 09/19/2019).
7. Christian Cachin and Jan Camenisch. Optimistic fair secure computation. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 93–111. Springer, Heidelberg, August 2000.
8. Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. LNCS, pages 280–312. Springer, Heidelberg, 2018.
9. Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. Zero-knowledge contingent payments revisited: Attacks and payments for services. In *ACM CCS 17*, pages 229–243. ACM Press, 2017.
10. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.
11. Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 61–85. Springer, Heidelberg, February 2007.
12. Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, August 2001.
13. Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 597–608. ACM Press, November 2014.
14. Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. FairSwap: How to fairly exchange digital goods. In *ACM CCS 18*, pages 967–984. ACM Press, 2018.
15. Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. PERUN: Virtual payment channels over cryptographic currencies. Cryptology ePrint Archive, Report 2017/635, 2017. http://eprint.iacr.org/2017/635.
16. Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.
17. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.
18. Mathias Hall-Andersen. Fastswap: Concretely efficient contingent payments for complex predicates. *IACR Cryptology ePrint Archive*, 2019:1296, 2019.
19. Harry A. Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. Arbitrum: Scalable, private smart contracts. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 1353–1370. USENIX Association, 2018.
20. Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer, Heidelberg, March 2013.
21. Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 705–734. Springer, Heidelberg, May 2016.

22. Ranjit Kumaresan and Iddo Bentov. Amortizing secure computation with penalties. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 418–429. ACM Press, October 2016.
23. Ranjit Kumaresan, Vinod Vaikuntanathan, and Prashant Nalini Vasudevan. Improvements to secure computation with penalties. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 406–417. ACM Press, October 2016.
24. Alptekin Küpçü and Anna Lysyanskaya. Usable optimistic fair exchange. In Josef Pieprzyk, editor, *CT-RSA 2010*, volume 5985 of *LNCS*, pages 252–267. Springer, Heidelberg, March 2010.
25. Henning Pagnia and Felix C Gärtner. On the impossibility of fair exchange without a trusted third party. Technical report, Technical Report TUD-BS-1999-02, Darmstadt University of Technology, 1999.
26. Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. *CoRR*, abs/1908.04756, 2019.
27. Eric Wagner, Achim Völker, Frederik Fuhrmann, Roman Matzutt, and Klaus Wehrle. Dispute resolution for smart contract-based two-party protocols. In *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2019, Seoul, Korea (South), May 14-17, 2019*, pages 422–430. IEEE, 2019.
28. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

A Universal Composability Framework

The *universal composability* (UC) framework, introduced by Canetti [10], allows to represent any arbitrary cryptographic protocol and to analyze its security. It aims at providing a systematic way to describe protocols and an intuitive way to create definitions of security. In order to simplify the security analysis, the UC framework allows analyzing the security of a protocol in isolation and preserves the security under composition by means of a special composition operation. This operation is called the *universal composition* operation. It follows that the security of a protocol can be considered in isolation while the security properties still hold in any arbitrary context. Hence, the UC framework offers a way to model cryptographic protocols in complex environments like modern communication networks. Moreover, the universal composition operation allows to build complex protocols out of cryptographic building blocks and proves its security based on the security of the building blocks.

Protocols. A *protocol* consists of several computer programs that are executed by communicating computational entities called *parties*. Along with the parties, the model of protocol execution comprises an environment and an adversary that can control a subset of the parties as well as the communication network. The environment freely chooses inputs for the protocol parties as well as for the adversary and obtains all outputs. The output of a protocol execution consists of all the outputs obtained from the parties as well as the output of the adversary.

A protocol is said to evaluate an arbitrary function f. An *ideal functionality* for f is a computing element that receives all inputs from the parties, computes f, and returns the outputs to each party. It can be considered as an incorruptible *trusted party* that evaluates f. The adversary is restricted in communicating with the ideal functionality via the corrupted parties and the specified interface of the ideal functionality.

A protocol π is said to *UC-realize* an ideal functionality for a function f if for any adversary $\mathcal{A}$ there exists an "ideal adversary" Sim such that no environment $\mathcal{Z}$ can distinguish with non-negligible probability whether it is interacting with π and $\mathcal{A}$ or with Sim and the ideal functionality for f.

Composition. As soon as it is shown that a protocol π UC-realizes an ideal functionality $\mathcal{F}$, this protocol can be used within more complex protocols. Therefore, lets consider a protocol ρ that makes subroutine calls to multiple instances of an ideal functionality $\mathcal{F}$. In this case, ρ is called a *$\mathcal{F}$-hybrid protocol* and $\mathcal{F}$ is called an *hybrid functionality*. The protocol $\rho^{\mathcal{F}\to\pi}$ specifies the protocol ρ where each call to an instance of the ideal functionality $\mathcal{F}$ is replaced by a call to an instance of the protocol π and each output of an instance of π is treated as an output of an instance of $\mathcal{F}$. Depending on the fact that π UC-realizes $\mathcal{F}$ and assuming the protocol ρ UC-realizes an ideal functionality $\mathcal{F}'$, the *universal composition theorem* can be used to prove that the protocol $\rho^{\mathcal{F}\to\pi}$ also UC-realizes $\mathcal{F}'$.

Two major implications of the composition theorem are modularity and stronger security. The composition theorem allows to split a complex task into small protocols. The security of these protocols can be shown in isolation and the protocols can be composed together later on. Moreover, a protocol that UC-realizes an ideal functionality $\mathcal{F}$ can be inserted into any context that makes calls to this ideal functionality. This way, the protocol can be used in any arbitrary context not known during the design process of the protocol.

Ideal and real world execution. The security of a protocol is defined by comparing the protocol execution with an execution of an *ideal protocol*. The ideal protocol consists of an *ideal functionality* and so-called *dummy parties*. The dummy parties replace the main parties of the real protocol within the ideal protocol by simply forwarding inputs to the ideal functionality and outputs from the ideal functionality. The ideal protocol for an ideal functionality $\mathcal{F}$ is denoted by $IDEAL_{\mathcal{F}}$. In contrast, the real protocol is denoted by $REAL_{\mathcal{F}}$.

On the one hand, the execution of the real protocol is done within the so-called *real world*. It consists of the real protocol $REAL_{\mathcal{F}}$ with its participating parties, an adversary $\mathcal{A}$, and an environment $\mathcal{Z}$. The adversary can corrupt parties and hence all internal states and actions are controlled by $\mathcal{A}$. On the other hand, the execution of the ideal protocol $IDEAL_{\mathcal{F}}$ is done within the *ideal world*. It consists of $IDEAL_{\mathcal{F}}$ along with the dummy parties, an ideal world adversary Sim, which is called *simulator*, and an environment $\mathcal{Z}$. Within the ideal world, the simulator may interact with the ideal functionality via the interface of the ideal functionality. In both worlds, the environment provides inputs to the parties and the adversary. It collects outputs from the parties and the adversary with the goal to distinguish between the interaction with the real protocol and the ideal protocol. Hence, the environment acts as a distinguishing entity with the goal to tell apart whether or not it is interacting with the real protocol.

In both worlds, it is possible to include *hybrid functionalities*. We provide several of them for the modeling of our construction in Appendix B.

Generalized universal composability framework. The basic UC framework does not allow the setup of a component that may be accessed by multiple protocol instances. Within the UC framework, each protocol instance has its own hybrid functionality instances with which the protocol may interact. When considering hash functions as objects that should be captured by the model, this situation seems to be not realistic. One concrete hash function, e.g., `keccak256`, is used by multiple parties in multiple different protocol instances. Therefore, Canetti et al. [11] introduced the *generalized UC* (GUC) framework. It allows to model shared ideal functionalities, also called *global functionalities*, which can be accessed by all protocol instances.

Security notion. Informally, a protocol is said to be at least as secure as an ideal functionality if there exists no environment that can tell apart whether it is interacting with the real world or with the ideal world.
Considering a protocol Π which has access to one or more hybrid functionalities $\mathcal{G}_1, \ldots, \mathcal{G}_n$. The output of the environment $\mathcal{Z}$ after the interaction with protocol Π and an adversary $\mathcal{A}$ on input 1^κ and $x \in \{0,1\}^*$ is denoted by

$$REAL_{\Pi,\mathcal{A},\mathcal{Z}}^{\mathcal{G}_1,\ldots,\mathcal{G}_n}(\kappa, x).$$

In the ideal world the protocol is replaced by an ideal protocol consisting of an ideal functionality $\mathcal{F}$ and dummy parties. In addition, the ideal protocol may have access to one or more hybrid functionalities $\mathcal{G}_1, \ldots, \mathcal{G}_n$. The output of the environment $\mathcal{Z}$ after interacting with the ideal protocol and simulator Sim on input 1^κ and $x \in \{0,1\}^*$ is denoted by

$$IDEAL_{Sim,\mathcal{Z}}^{\mathcal{F},\mathcal{G}_1,\ldots,\mathcal{G}_n}(\kappa, x).$$

Using these two random variables, the security of a protocol Π is defined as follows:

Definition 1 (GUC security of protocol Π). *Let $\kappa \in \mathbb{N}$ be a security parameter and Π be a protocol in a hybrid world with hybrid functionalities $\mathcal{G}_1, \ldots, \mathcal{G}_n$. Π is said to GUC-realize an ideal functionality $\mathcal{F}$ in the hybrid world if for every probabilistic polynomial time (ppt) adversary $\mathcal{A}$ attacking Π there exists a ppt algorithm Sim such that the following holds for all ppt environments $\mathcal{Z}$ and for all $x \in \{0,1\}^*$:*

$$IDEAL_{Sim,\mathcal{Z}}^{\mathcal{F},\mathcal{G}_1,\ldots,\mathcal{G}_n}(\kappa, x) \approx_c REAL_{\Pi,\mathcal{A},\mathcal{Z}}^{\mathcal{G}_1,\ldots,\mathcal{G}_n}(\kappa, x).$$

B Model Components

To analyze the security of a protocol, an underlying model must be defined. This section presents the model components. Moreover, our construction is based on these model components. A formal description of OptiSwap is given in Appendix C.

To simplify the presentation and reading of the model and the protocol, the explicit mention of session identifiers and sub-session identifiers, which are typically denoted by *sid* and *ssid*, is omitted. Instead, the contract identifier *id* is used to distinguish different sessions. In reality, each protocol instance uses its own smart contract instance lying on the blockchain and hence the smart contract address may be used to identify a protocol session.

Synchronous Communication. A common and often used abstraction of the communication model within networks is that of *synchronous* communication. Formally, this can be modeled using the ideal functionality $\mathcal{F}_{\text{SYN}}$ given in [10] or using a global clock functionality [20, 21, 4]. We abstract the communication by a synchronous communication model using the following assumptions. The protocol is executed in *rounds* where all parties are aware of the current round. In each round each party receives all messages that were sent to them in the previous round. The communication itself is instantaneous such that messages are sent within one round and received within the next round.

Global Random Oracle. *Random oracles* are widely used in cryptographic security proofs to model idealized hash functions. A random oracle provides on a query value q a completely random response value r unless the value q was queried before. In this case, the random oracle returns the same value. Despite it is not proven that there exists a practical hash function that has the same properties, many security proofs are based on this random oracle model.

Since one instantiation of a hash function like `keccak` is used by multiple protocols and multiple users concurrently, random oracles are often modeled as global functionalities. There are already several variants of global random oracles presented in the literature. The most intuitive definition presented by Canetti et al. [13] is the so-called *strict* global random oracle. It simply offers an interface to query values and nothing more. While this is a natural model for cryptographic hash functions, this model is not appropriate to prove security of some cryptographic building blocks. Canetti and Fischlin showed that a protocol that realizes UC commitments needs a setup that gives the simulator an advantage over the environment [12]. An overview over different variants of global random oracles and more details are given in [8].

We use a global random oracle with restricted programming and observability denoted by $\mathcal{H}$ to model hash functions. We state the definition in the following and refer the reader to [8] for a detailed description.

Using a global random oracle with restricted programming and observability, the simulator *Sim* has control over the random oracle by programming hashes to specific responses and the ability to observe all queries made by the environment $\mathcal{Z}$. This results in an advantage over $\mathcal{Z}$ such that the restricted programmable and observable global random oracle is a practical tool for simulation within a GUC proof.

Global Random Oracle $\mathcal{H}$

The functionality $\mathcal{H}$ is the global random oracle with restricted programming and observability. Internally, it stores a set Q of all legitimate queries, a set P of all programmed inputs, and sets Q_{id} for all sessions id of all illegitimate queries. All sets are initially set to $\emptyset$. The functionality accepts queries of the following types:

Query

Upon receiving message (query, id, q) from a party of session id', proceed as follows:

- If $(id, q, r) \in Q$, respond with (query, q, r).
- If $(id, q, r) \notin Q$, sample $r \in \{0,1\}^{\mu}$, 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 $(\mathsf{program}, id, q, r)$ by the adversary $\mathcal{A}$, check if (id, q, r') is defined in Q. If this is the case, abort. Otherwise, if $r \in \{0,1\}^{\mu}$, store (id, q, r) in Q and (id, q) in P.
Upon receiving message $(\mathsf{isProgrammed}, q)$ from a party of session id, check if $(id, q) \in P$. If this is the case, respond with $(\mathsf{isProgrammed}, 1)$, else respond with $(\mathsf{isProgrammed}, 0)$.

Observe

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

Ledger Functionality. Within our interactive fair exchange protocol, two parties transfer money to a smart contract which locks the coins until the end of the protocol execution. In order to model the basic functionality of transferring and locking coins, a *ledger functionality* is needed. Since the ledger can be used by multiple protocols in parallel and can be accessed over multiple protocol executions, it should be modeled as a global functionality [13, 8]. Our ledger functionality $\mathcal{L}$ is a slightly modified version of the global ledger functionality introduced in [15] and used by [14]. The differences are emphasized in the following description.

Global Ledger Functionality $\mathcal{L}$

Functionality $\mathcal{L}$, running with a set of parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ stores the balance $p_i \in \mathbb{N}$ for every party $\mathcal{P}_i, i \in [n]$ and a partial function L for frozen cash. It accepts queries of the following types:

Update

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

Freeze

Upon receiving message $(freeze, id, \mathcal{P}_i, p)$ from an ideal functionality of session id, check if $p_i > p$. If this is not the case, reply with $(nofunds, \mathcal{P}_i, p)$. Otherwise, set $p_i = p_i - p$ and check if $(id, p') \in L$. If this check holds, update (id, p') to $(id, p' + p)$. Otherwise, store (id, p) in L. Finally, send $(frozen, id, \mathcal{P}_i, p)$ to every entity.

Unfreeze

Upon receiving message $(unfreeze, id, \mathcal{P}_j)$ from an ideal functionality of session id, check if $(id, p) \in L$. If this check holds and no message $(block, id)$ from corrupted party $\mathcal{P}_j^*$ is received in the same round or beforehand, delete (id, p) from L, set $p_j = p_j + p$, and send $(unfrozen, id, \mathcal{P}_j, p)$ to every entity.

The internal state and as well as the *update*- and *freeze*-methods are the same as described in [14]. We modified the *unfreeze*-operation regarding two aspects.

First, it sends all coins locked within the smart contract to party $\mathcal{P}_j$. This means the message contains no parameter to unfreeze only part of the frozen coins. Since our construction makes no use of unfreezing only parts of the frozen coins, the model is simplified by this adaption.
Second, corrupted parties have the ability to refuse an unfreezing in their favor by sending a *block*-message in the same round or already before. The ledger functionality allows the refusing of an unfreezing to model an abort of a corrupted party.
The more intuitive way would be to require an honest party to actively start an unfreezing. However, this would lead to a lot of overhead in the protocol description of the honest case. Therefore, the explained approach is taken to simplify the model and the protocol description. In case no *block*-message was received from the corrupted party $\mathcal{P}_j^*$, the *unfreeze*-operation deletes the entry (id, p) from L and increases the balance p_j of party $\mathcal{P}_j$ by p.
To signal that an ideal functionality $\mathcal{F}$ has access to the global ledger functionality, it is denoted by $\mathcal{F}^{\mathcal{L}}$.

C Formal Protocol Description

In this section, we provide some explanation about the formal definition of the judge smart contract and the two protocol parties, which is given in Section 3.4.

In comparison to the judge functionality used by FairSwap, $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ additionally contains functions for handling the challenge-response procedure as well as timeouts. Moreover, it supports a fourth way to finalize the protocol.

During the challenge-response procedure, the judge smart contract acts as storage for the most recent challenge query as well as the most recent response. Each time Buyer $\mathcal{B}$ sends a challenge and each time Seller $\mathcal{S}$ answers a response, $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ locks a security fee from the Seller of the message.

Upon receiving a *challenge*- or *respond*-message, the judge starts a timeout. As soon as a timeout is passed, the party that is no longer engaging in the protocol is considered malicious. Therefore, the party that sent the last message may finalize the exchange and claim all the locked money.

We like to stress that $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ does not validate each response received from $\mathcal{S}$. Instead, $\mathcal{B}$ is responsible for validating them and only in case he received an invalidate response, he may complain about it. When receiving such a complain, $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ validates the most recent response and resolves the dispute this way. Since the most recent challenge query and response are stored within the contract, the judge is easily feasible to validate them. By shifting the responsibility for validation to Buyer $\mathcal{B}$, the judge must perform at most one response validation.

D Algorithms

In this section, we present algorithms used by our OptiSwap protocol. The usage of algorithms make our construction more modular and ease the description of it. A formal description of OptiSwap is given in Appendix C.

D.1 Initialization

We start with the algorithm used by the Seller at the start of the protocol execution. Before sending the selling offer to the Buyer and initializing the judge smart contract, he executes the Presetup-algorithm. On input the circuit ϕ, the witness $\mathbf{x}$, and the encryption key k, the algorithm outputs the encrypted witness $\mathbf{z}$ along with some auxiliary data. This includes commitments r_z to $\mathbf{z}$, r_ϕ to circuit ϕ, and r_e to the intermediary values of the evaluation $\phi(\mathbf{x})$. See Algorithm 1 for a formal definition.

Algorithm 1 $\mathsf{Presetup}(\phi, \mathbf{x}, k)$

Require: Verification circuit ϕ, witness $\mathbf{x} \in (\{0,1\}^\lambda)^n$, and encryption key k
Ensure: $(\mathbf{z}, r_z, r_e, r_\phi)$, where $\mathbf{z}$ is the encrypted witness, r_z, r_e, and r_ϕ are commitments based on Merkle trees

for $i = 1$ to n **do**
  $z_i = \mathsf{Enc}(k, x_i)$
  $e_i = z_i$
end for
$r_z = \mathsf{MTHash}(\mathbf{z})$
for $i = n+1$ to m **do**
  parse $\phi_i = (i, op_i, I_i)$
  $out_i = op_i(out_{I_i[1]}, \ldots, out_{I_i[l]})$
  $e_i = \mathsf{Enc}(k, out_i)$
end for
$r_e = \mathsf{MTHash}(\mathbf{e})$
$r_\phi = \mathsf{MTHash}(\phi)$
return $(\mathbf{z}, r_z, r_e, r_\phi)$

D.2 Challenge-Response Procedure

The challenge-response procedure is part of the dispute resolution protocol. The Seller and Buyer alternately create challenge queries and responses, respectively. To this end, the parties execute the NextChallenge- and the GenerateResponse-algorithm.

Algorithm 2 defines the NextChallenge-algorithm which is called by the Buyer. It demands a circuit ϕ, the set of all received responses so far R, and some helper data H as input. The helper data contains a circuit gate in focus ϕ_i, the shared encryption key k, and the tuple of all circuit gate output values computed by Buyer himself $\mathbf{o'} = (o'_1, \ldots o'_m)$. The algorithm creates a new challenge query Q and update the helper data. It returns the tuple (Q, H'), where H' denotes the updated helper data.

Upon receiving a challenge query, the Seller is responsible for answering it. Therefore, he calls the GenerateResponse-algorithm which outputs a response R_q. Seller provides circuit ϕ, witness $\mathbf{x}$, encryption key k, and the most recent challenge query Q as input.

Algorithm 2 NextChallenge(ϕ, R, H)

Require: Circuit ϕ, set of responses R, and helper data $H = (\phi_i, k, \mathbf{o}')$, where ϕ_i is the circuit gate put in focus, k is a shared symmetric encryption key, and $\mathbf{o}' = (o'_1, \ldots, o'_m)$ denotes the tuple of all circuit gate output values computed by the challenger himself.

Ensure: (Q, H'), where Q is a challenge query denoting the circuit gates to be challenged and H' being the updated helper data.

if $R = \emptyset$ **then**
  return $(\{m\}, (\phi_m, k, \mathbf{o}'))$
else
  parse $\phi_i = (i, op_i, I_i)$
  set $R_i = \{j \in I_i : (j, e_j, \cdot) \in R\}$
  if $R_i \neq I_i$ **then**
    return $(I_i \setminus R_i, (\phi_i, k, \mathbf{o}'))$
  else
    for $j \in R_i$ **do**
      $o_j = \mathsf{Dec}(k, e_j)$
    end for
    choose $j \in I_i : o_j \neq o'_j$
    parse $\phi_j = (j, op_j, I_j)$
    return $(I_j, (\phi_j, k, \mathbf{o}'))$
  end if
end if

Algorithm 3 GenerateResponse$(\phi, \mathbf{x}, k, \mathbf{q})$

Require: $(\phi, \mathbf{x}, k, Q)$, where ϕ is the verification circuit, $\mathbf{x}$ is the witness, k is the encryption key used within the Presetup-algorithm, and Q is the challenge query containing the indices of all challenged gates

Ensure: R_q, where R_q is a set of all encrypted output values along with their Merkle proofs challenged by the given query

for $i = 1$ to n **do**
  $out_i = x_i$
  $e_i = \mathsf{Enc}(k, x_i)$
end for
for $i = n + 1$ to m **do**
  parse $\phi_i = (i, op_i, I_i)$
  $out_i = op_i(out_{I_i[1]}, \ldots, out_{I_i[l]})$
  $e_i = \mathsf{Enc}(k, out_i)$
end for

$\mathbf{e} = (e_1, \ldots, e_m)$
$M_e = \mathsf{MTHash}(\mathbf{e})$
$R_q = \emptyset$
for $i \in Q$ **do**
  $R_q = R_q \cup \{(i, e_i, \mathsf{MTProof}(M_e, i))\}$
end for
return r

A third algorithm exists for the challenge-response procedure. This one is called the ValidateResponse-algorithm. In order to decrease the computational burden on the judge smart contract, the validation of a response is not done for each one by the contract. Instead, the Buyer is responsible for validating a received response. This includes checking if for each challenge circuit gate a corresponding output value is provided and if all Merkle proofs

verify. As soon as one of these criteria does not hold, the response is invalid. To perform these checks, Buyer executes the ValidateResponse-algorithm given by Algorithm 4. The same algorithm is used by the judge smart contract to check the validity of the most recent response after Buyer complained about it. The algorithm requires as in put a challenge query Q, the corresponding response R, and the Merkle Tree root r_e for the intermediate values of the evaluation $\phi(\mathbf{x})$.

Algorithm 4 ValidateResponse(Q, R, r_e)

Require: (Q, R, r_e), where Q is a challenge query, R is the corresponding response, and r_e is the Merkle root for the encrypted intermediate computation values
Ensure: *valid*, where *valid* = true if the response is valid or false otherwise

```
for i ∈ Q do
    if ∄(i, e_i, π_i) ∈ R then
        return false
    else if MTVerify(π_i, i, r_e) = 0 then
        return false
    end if
end for
return true
```

D.3 Dispute Resolution

After the challenge-response procedure, the dispute has to be resolved eventually. Assuming a cheating Seller, Buyer needs to create proof of misbehavior (PoM). This concise proof technique was introduced in [14] and outlined in Section 2.1. Algorithm 5 states the GenerateProof-algorithm executed by Buyer.

Finally, the judge smart contract needs to adjudicate on the correctness of the fair exchange. To this end, he executes the Judge-algorithm. It checks all Merkle proofs contained in the PoM and recomputes the specified circuit gate. In case the computed value differs to the value provided in the PoM, the proof is considered valid, otherwise it is invalid. Algorithm 6 gives a formal definition of the Judge-algorithm including all required inputs.

E Security Proof

We present a security proof within the generalized universal composability (GUC) framework. This way, we prove that our OptiSwap protocol Π GUC-realizes the ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ within the $(\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}, \mathcal{L}, \mathcal{H})$-hybrid world. A brief introduction into the (generalized) universal composability framework is given in Appendix A. The protocol Π is based on the assumptions of an IND-CPA-secure symmetric encryption scheme and a hash function that is modeled as a global programmable random oracle $\mathcal{H}$. The security model is presented in Appendix B and a formal description of the protocol parties as well as the judge functionality is given in Appendix C.

Assumptions and Simplifications. We assume static corruption, which means that the protocol parties may be corrupted by the adversary only at the beginning of the execution. In the following, the four different cases of corruption are considered in isolation. For each scenario, a formal description of a simulator and a detailed argumentation why the simulator can be used to achieve indistinguishability is given. In addition, the setup of each scenario is depicted to support the understanding of connections between the parties within an execution.

Algorithm 5 $\mathsf{GenerateProof}(k, \phi, R)$

Require: (k, ϕ, R), where k is the decryption key, ϕ is the verification circuit, and R is the set of all responses received so far
Ensure: π, where π is either a proof of misbehavior or false

for $(i, e_i, \pi_i) \in R$ **do**
  set $s = searching$
  $out_i = \mathsf{Dec}(k, e_i)$
  parse $\phi_i = (i, op_i, I_i)$
  for $j = 1$ to l **do**
    if $(I_i[j], e_{I_i[j]}, \pi_{I_i[j]}) \notin R$ **then**
      s = $failed$
      break
    else
      $out_{I_i[j]} = \mathsf{Dec}(k, e_{I_i[j]})$
    end if
  end for
  if $s \neq failed$ **then**
    $out'_i = op_i(out_{I_i[1]}, \ldots, out_{I_i[l]})$
    if $out_i \neq out'_i$ **then**
      $\pi_\phi = \mathsf{MTProof}(\mathsf{MTHash}(\phi), i)$
      $\pi = (\pi_\phi, \pi_i, \pi_{I_i[1]}, \ldots, \pi_{I_i[l]})$
      return π
    end if
  end if
end for
return false

The protocol parties of the hybrid world execution are denoted by $\mathcal{S}$ and $\mathcal{B}$, while the dummy parties of the ideal world are denoted by $\tilde{\mathcal{S}}$ and $\tilde{\mathcal{B}}$. Malicious parties are delineated by *, e.g., a corrupted Seller in the hybrid world is denoted by $\mathcal{S}^*$ and in the ideal world it is denoted by $\tilde{\mathcal{S}}^*$.
In the following simulations, the ideal adversary internally runs the judge smart contract $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$. Running $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ on input m and obtaining the output m' by the simulator is denoted by $m' \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(m)$. This implies that the messages m and m' are sent to the environment $\mathcal{Z}$ and to the parties according to the behavior of $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$. Note, since $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ is only run internally by the simulator, no coins are frozen or unfrozen by $\mathcal{L}$.

The environment has the power to delay message sent to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ and $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ via the adversary. To simplify the simulation, it is assumed that whenever a message is delayed by time δ, the ideal adversary uses the influence port of the ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in order to delay the execution in the ideal world by the exact same amount of time δ. This ensures that the environment cannot distinguish the execution of the hybrid world and the execution of the ideal world based on the delay within an execution. In the following simulations, the effect of delayed messages is not further considered based on this argumentation.

In order to show indistinguishability between the real world execution and the ideal world execution, some proofs are based on a sequence of games $Game_0, \ldots, Game_n$. This common proof technique allows to show indistinguishability between $Game_0$ and $Game_n$ by showing indistinguishability between $Game_i$ and $Game_{i+1}$ for each $i \in [n-1]$. Since the goal is to show indistinguishability between the real world execution and the ideal world execution, $Game_0$ is set to be the real world execution and $Game_n$ is the simulation in the ideal world. Each intermediate game is a hybrid simulation which simplifies the full UC simulation. Such a hybrid simulation can include a simulator that controls inputs and outputs of honest parties.

Algorithm 6 Judge$(k, r_z, r_e, r_\phi, \pi)$

Require: $(k, r_z, r_e, r_\phi, \pi)$, where k is a decryption key, r_z, r_e, r_ϕ are Merkle tree roots for the encrypted witness, the encrypted intermediate computation values, and the circuit, respectively, and π is the proof of misbehavior generated by the GenerateProof-algorithm (cf. Algorithm 5)
Ensure: 1 if proof of misbehavior is valid, otherwise 0

parse $\pi = (\pi_\phi, \pi_{out}, \pi_{I_i[1]}, \ldots, \pi_{I_i[l]})$
parse $\pi_\phi = (\phi_i, l_1^\phi, \ldots, l_{log_2(n)}^\phi)$
parse $\phi_i = (i, op_i, I_i)$
parse $\pi_{out} = (e_i, l_1^o, \ldots, l_{log_2(n)}^o)$
$out_i = \mathsf{Dec}(k, e_i)$
if $\mathsf{MTVerify}(\pi_\phi, i, r_\phi) = 0$ **then**
    return 0
end if
if $\mathsf{MTVerify}(\pi_{out}, i, r_e) = 0$ **then**
    return 0
end if
if $i = m$ and $out_i \neq 1$ **then**
    return 1
end if
for $j \in [l]$ **do**
    if $\mathsf{MTVerify}(\pi_{I_i[j]}, I_i[j], r_e) = 0$ **then**
        return 0
    end if
    parse $\pi_{I_i[j]} = (e_{I_i[j]}, l_1^{I_i[j]}, \ldots, l_{log_2(n)}^{I_i[j]})$
    $out_{I_i[i]} = \mathsf{Dec}(k, e_{I_i[j]})$
end for
$out_i' = op_i(out_{I_i[1]}, \ldots, out_{I_i[l]})$
if $out_i' = out_i$ **then**
    return 0
end if
return 1

Since there exist different types of indistinguishability, we like to note that whenever indistinguishability is mentioned in the following security proof, the computational variant is considered.

Simulation with Two Honest Parties

In the honest execution case, the environment $\mathcal{Z}$ provides input values to honest Seller $\mathcal{S}$ and honest Buyer $\mathcal{B}$. After and during the execution of the protocol Π, both honest parties forward their output values back to the environment. The adversary $\mathcal{A}$ provides additional leakage information to $\mathcal{Z}$ and can influence the execution of the hybrid functionalities $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ and $\mathcal{H}$ as specified by their interfaces. Since it is sufficient to consider a dummy adversary, $\mathcal{A}$ simply forwards all messages received from $\mathcal{Z}$ to the specified recipient and leaks all messages obtained from the hybrid functionalities.

In the ideal world, the simulator needs to provide the same information to the environment as the adversary $\mathcal{A}$ in the hybrid world execution. Therefore, Sim needs to create a transcript of the whole protocol execution and to send the created messages to $\mathcal{Z}$. This includes the message sent from Seller to Buyer in the first round of the protocol as well as all transactions sent to the judge smart contract $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ and all interactions with $\mathcal{H}$.

It is assumed that there exists a secure channel between $\mathcal{S}$ and $\mathcal{B}$. This way, the environment $\mathcal{Z}$ learns that a message was sent, but the content of the message remains secret. This assumption is important to enable the simulation of the honest case. Suppose there exists no secure channel, the simulator has to simulate an encrypted witness $\mathbf{z}^*$ without the

knowledge of the correct witness $\mathbf{x}$. Moreover, the decryption of $\mathbf{z}^*$ using a key k has to equal $\mathbf{x}$.

On the one hand, in the scenario of a corrupted Buyer the programming feature of the global random oracle $\mathcal{H}$ is exploited to ensure the correct decryption. But, since the simulator never gets to know $\mathbf{x}$ during the case of two honest parties, there is no way to exploit the programming feature to ensure the correct decryption. On the other hand, in the scenario of a corrupted Seller, the observability feature of $\mathcal{H}$ is used to learn the encryption key at the beginning of the protocol execution. The knowledge of the key and the encrypted witness $\mathbf{z}$, provided by the corrupted Seller, is sufficient for the simulator to reconstruct the correct witness $\mathbf{x}$ and to achieve a correct simulation. The knowledge of the encryption key would also help in the honest scenario, but the observability feature can only be used to obtain all queries executed from outside of the protocol session. In particular, honest parties always belong to the protocol session and hence queries from these parties cannot be obtained. Therefore, a secure channel must be assumed.

Claim. There exists an efficient algorithm Sim such that for all ppt environments $\mathcal{Z}$ that *do not corrupt any party* it holds that the execution of Π in the $(\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}, \mathcal{L}, \mathcal{H})$-hybrid world in presence of adversary $\mathcal{A}$ is computationally indistinguishable from the ideal world execution of $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ with the ideal adversary Sim.

Proof. We define a simulator Sim, which internally runs $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ and has access to the restricted programmable oracle $\mathcal{H}$.

1. If $\tilde{\mathcal{S}}$ starts the execution with $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the first round, simulator Sim learns $id, \phi, p, f_{\mathcal{S}}, f_{\mathcal{B}}$ from $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. Sim samples a key $k^* \leftarrow \mathsf{Gen}(1^\kappa)$ and sets $\mathbf{x}^* = 1^{n \times \lambda}$. He computes $(c^*, d^*) \leftarrow \mathsf{Commit}(k^*)$ and $(\mathbf{z}^*, r_z^*, r_e^*, r_\phi) \leftarrow \mathsf{Presetup}(\phi, \mathbf{x}^*, k^*)$. Sim simulates the execution of Π by running $(active, id, c^*, r_z^*, r_e^*, r_\phi, a_\phi, p, f_{\mathcal{S}}, f_{\mathcal{B}}) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(initialize, id, c^*, r_z^*, r_e^*, r_\phi, a_\phi, p, f_{\mathcal{S}}, f_{\mathcal{B}})$. In addition, he provides the information to $\mathcal{Z}$ that $\tilde{\mathcal{S}}$ sent a message to $\tilde{\mathcal{B}}$ over the secure channel.
2. If $\tilde{\mathcal{S}}$ aborts the execution in the second round, $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ leaks $(abort, id, \tilde{\mathcal{S}})$ to Sim. The simulator then runs $(aborted, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(abort, id)$ and terminates the simulation.
   If Sim receives $(buy, id, \tilde{\mathcal{B}})$ from $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the second round, he simulates the acceptance of $\tilde{\mathcal{B}}$ by running $(initialized, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(accept, id)$.
3. If $\tilde{\mathcal{B}}$ aborts the execution in the third round, $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ leaks $(abort, id, \tilde{\mathcal{B}})$ to Sim. Then Sim simulates the abort by executing $(aborted, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(abort, id)$ and terminating the simulation.
   If no message is leaked by $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the third round, Sim simulates the revealing of the encryption key by running $(revealed, id, k^*, d^*) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(reveal, id, k^*, d^*)$.
4. In round four, the simulator simulates the finalization of the fair exchange by running $(sold, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(finalize, id)$.

In the following, it is shown that the transcript produced by the Sim is indistinguishable from the transcript produced by the execution of Π in the hybrid world.
When receiving an incorrect witness $\mathbf{x}^*$ from the environment such that $\phi(\mathbf{x}^*) \neq 1$, honest Seller does not start the protocol execution in the hybrid world. The same behavior is defined by $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the ideal world, since the ideal functionality assumes a correct witness from an honest Seller. Hence, only an execution with a correct witness $\mathbf{x}$ such that $\phi(\mathbf{x}) = 1$ is considered. This is a difference to the protocol presented in [14] in which an honest Seller may also send an incorrect witness.

In round 1, the environment learns the information that Seller sent a message to the Buyer. Since the message is sent over a secure channel, $\mathcal{Z}$ cannot extract any further information. The same information is given by *Sim* in round one. Moreover, the environment obtains the *initialize*- and *active*-message when the honest Seller interacts with $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$. The simulator internally runs $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ to obtain the same messages. The values $r_\phi, a_\phi, p, f_\mathcal{S}, f_\mathcal{B}$ are identical in both executions, since they directly depend on the input values provided by $\mathcal{Z}$. The commitment values c^*, r_z^*, r_e^* depend on a randomly sampled encryption key. If $\mathcal{Z}$ can obtain the value $\mathbf{z}^*$ and decrypt it to $\mathbf{x}^* \neq \mathbf{x}$, it can distinguish the executions. Since the underlying commitment scheme used to generate c^* fulfills the hiding property, $\mathcal{Z}$ cannot retrieve $\mathbf{z}^*$ from these commitment values. Thus, $\mathcal{Z}$ cannot distinguish the two executions based on these commitment values.

In the second round, honest Seller may abort the protocol execution. In this case, $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ leaks $(abort, id, \mathcal{S})$ to *Sim*, which internally runs $(aborted, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(abort, id)$. In the hybrid world and in the ideal world, the *abort*- and *aborted*-messages are identical and hence indistinguishable. Furthermore, honest Buyer $\mathcal{B}$ outputs the message $(aborted, id)$ towards $\mathcal{Z}$, which equals the message $(aborted, id)$ sent from $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ to $\mathcal{Z}$ through $\tilde{\mathcal{B}}$ in case of the abort.
Instead of Seller's abort, Buyer may accept the exchange offer. By providing a (buy, id, ϕ)-message to $\mathcal{B}$, $\mathcal{Z}$ initiates the acceptance. Honest Buyer sends a $(accept, id)$-message to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ in the hybrid world and $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ outputs $(initilized, id)$. The exact same messages are generated by *Sim* in the ideal world by running $(initialized, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(accept, id)$. Hence, $\mathcal{Z}$ cannot distinguish the executions in the second round.

Honest Buyer may abort the protocol execution in the third round. The result in the protocol execution Π is $\mathcal{S}$ outputting $(aborted, id)$ and terminating. *Sim* simulates the same messages exchanged with $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ after $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ leaks $(abort, id, \mathcal{B})$ and terminates the simulation. Again, the ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ sends $(aborted, id)$ to $\mathcal{Z}$ through $\tilde{\mathcal{S}}$, which guarantees the same output behavior.
If Buyer does not abort the execution in the third round, honest Seller reveals his encryption key. He sends $(reveal, id, k, d)$ to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$, where $\mathsf{Open}(c, d, k) = 1$. *Sim* simulates the key revealing by running $(revealed, id, k^*, d^*) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(reveal, id, k^*, d^*)$, where $\mathsf{Open}(c^*, d^*, k^*) = 1$. Since both keys k and k^* are sampled randomly using the Gen-algorithm, $\mathcal{Z}$ cannot distinguish between the two keys. It might be possible to use the commitment values r_z^*, r_e^* from the *initialization*-phase to extract further information. Since the opening values for these two Merkle tree commitments are not revealed by honest Seller, it requires to break the hiding property of the commitment scheme in order to gain additional information.
Honest Buyer also outputs the obtained witness $\mathbf{x}$ in the third round of the protocol execution. The ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ sends the value set by $\mathcal{Z}$ at the beginning of the ideal world execution to the dummy Buyer, which forwards the value. In both cases, $\mathcal{Z}$ receives the value set as input to $\mathcal{S}$ and $\tilde{\mathcal{S}}$, respectively.

In round 4, honest Buyer sends a $(finalize, id)$-message to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ and outputs $(bought, id, \mathbf{x})$. Honest Seller receives a message from $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ and outputs $(sold, id)$. Both output messages are sent by $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the ideal world and *Sim* simulates the judge smart contract. Hence, the identical messages are sent within this round. After these messages, the hybrid world and the ideal world terminate the execution.

It remains to show that the money is locked and unlocked in the same rounds. In the protocol Π, p coins are locked from Buyer $\mathcal{B}$ in the second round if $\mathcal{B}$ accepts the fair exchange offer. The acceptance is signaled by the (buy, id, ϕ)-message and is successfully executed only if $\mathcal{B}$ has enough funds. In the ideal world, the ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ freezes p coins in the second round after it received a (buy, id, ϕ)-message from $\mathcal{B}$. Again, the money is only locked if $\mathcal{B}$ controls enough money. If honest Buyer aborts in round 3 before the encryption key is revealed, the money is transferred back to $\mathcal{B}$ by $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ in the hybrid world

and by $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the ideal world. After the encryption key is revealed, the money can only be unlocked in favor of honest Seller, since he provided a correct witness. In the hybrid world, $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ sends the money to $\mathcal{S}$ in round 4 after it received the $(finalize, id)$-message from $\mathcal{B}$. In the ideal world, $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ unlocks the money in favor of $\mathcal{S}$ in round 4.

Finally, it is easy to see that the simulator Sim defined before runs in polynomial time. After all, when considering an environment $\mathcal{Z}$ that does not corrupt any party, it is shown that there exists an efficient simulator such that no environment can distinguish between the execution of the hybrid world and the execution of the ideal world. It is shown that the money is locked and unlocked in the same rounds and the environment $\mathcal{Z}$ cannot distinguish the transcripts of both executions unless it breaks the hiding property of the used commitment schemes.

Simulation with Corrupted Seller

When considering a corrupted Seller, its internal state and program code is fully under the control of the environment $\mathcal{Z}$. Especially, corrupted Seller $\mathcal{S}^*$ may deviate from the protocol at any point in time during the protocol execution. In the ideal world, the simulator $Sim^{\mathcal{S}}$ becomes more complex. In addition to simulate the execution of protocol Π and generating a transcript of this execution, he needs to create all outputs of the corrupted Seller $\tilde{\mathcal{S}}^*$ towards the environment $\mathcal{Z}$ and towards the ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. In particular, each input sent from $\mathcal{Z}$ to the corrupted dummy party $\tilde{\mathcal{S}}^*$ is forwarded by $\tilde{\mathcal{S}}^*$ to $Sim^{\mathcal{S}}$. When receiving an input from $\mathcal{Z}$ through $\tilde{\mathcal{S}}^*$, the simulator needs to create the input to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$.

In the following, a detailed simulator $Sim^{\mathcal{S}}$ is given and it is argued why this simulator achieves indistinguishability. Since a corrupted Seller is also able to follow the protocol throughout the whole execution, it is not argued that the simulation works in this case, which is identical to the honest case. This means, the differences between the honest case as described beforehand and the case of a corrupted Seller are emphasized. The proof makes use of the observability features of the global random oracle $\mathcal{H}$. The same trick is used in the proof in [14].

Claim. There exists an efficient algorithm $Sim^{\mathcal{S}}$ such that for all ppt environments $\mathcal{Z}$ that *only corrupt the Seller* it holds that the execution of Π in the $(\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}, \mathcal{L}, \mathcal{H})$-hybrid world in presence of adversary $\mathcal{A}$ is computationally indistinguishable from the ideal world execution of $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ with the ideal adversary $Sim^{\mathcal{S}}$.

Proof. To show indistinguishability a sequence of games is used as explained at the beginning of this section. Before presenting $Game_2$, which equals the ideal world execution, $Game_1$ is given, which represents a hybrid simulation with the usage of $Sim_1^{\mathcal{S}}$. In addition to controlling the inputs and outputs of the corrupted Seller, $Sim_1^{\mathcal{S}}$ also controls inputs and outputs of the honest Buyer. This simplification is eliminated in $Game_2$, which makes use of the full simulator $Sim^{\mathcal{S}}$.
The construction of simulator $Sim_1^{\mathcal{S}}$ is given in the following. This simulator is used in $Game_1$. Furthermore, it is shown that no ppt environment $\mathcal{Z}$ can distinguish between the real world execution and the hybrid simulation using $Sim_1^{\mathcal{S}}$, i.e., $Game_0 \approx Game_1$. Afterwards, the full simulator $Sim^{\mathcal{S}}$ is stated to construct $Game_2$, which equals the ideal world execution. Again, indistinguishability between $Game_1$ and $Game_2$ is shown, i.e., $Game_1 \approx Game_2$. At the end, the two results can be merged to show that the real world execution is indistinguishable from the ideal world execution.
The structure of this proof is strongly related to the proof of the malicious Seller scenario in [14] and is deliberately chosen to simplify the comparison and readability of the following proof. The differences are highlighted explicitly.

Simulator $Sim_1^{\mathcal{S}}$ for hybrid simulation with corrupted Seller.

1. Upon receiving $(sell, id, \mathbf{z}, \phi)$ through $\tilde{\mathcal{S}}^*$ in the first round, $Sim_1^{\mathcal{S}}$ samples $\mathbf{x}^* \leftarrow \{0,1\}^{n \times \lambda}$. If message $(initialize, id, c, r_z, r_e, r_\phi, a_\phi, p, f_{\mathcal{S}}, f_{\mathcal{B}})$ is also received in round 1 through $\tilde{\mathcal{S}}^*$, $Sim_1^{\mathcal{S}}$ sends $(sell\text{-}fake, id, \mathbf{x}^*, \phi, p, f_{\mathcal{S}}, f_{\mathcal{B}})$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. Furthermore, $Sim_1^{\mathcal{S}}$ simulates the execution of Π by running $(active, id, c, r_z, r_e, r_\phi, a_\phi, p, f_{\mathcal{S}}, f_{\mathcal{B}}) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(initialize, id, c, r_z, r_e, r_\phi, a_\phi, p, f_{\mathcal{S}}, f_{\mathcal{B}})$.

   If not both messages *sell* and *initialize* are received through $\tilde{\mathcal{S}}^*$ in round one, the $Sim_1^{\mathcal{S}}$ terminates the simulation.
2. Upon receiving $(abort, id)$ through $\tilde{\mathcal{S}}^*$ in the second round, $Sim_1^{\mathcal{S}}$ sends $(abort, id)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. Furthermore, he simulates Π by running $(aborted, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(abort, id)$ and terminating the simulation.
   If $Sim_1^{\mathcal{S}}$ receives $(buy, id, \tilde{\mathcal{B}})$ from $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the second round, he simulates the acceptance of $\tilde{\mathcal{B}}$ by running $(initialized, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(accept, id)$.
3. If $\tilde{\mathcal{B}}$ aborts the execution in the third round, $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ leaks $(abort, id, \tilde{\mathcal{B}})$ to $Sim_1^{\mathcal{S}}$. Then, $Sim_1^{\mathcal{S}}$ simulates the abort by executing $(aborted, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(abort, id)$ and terminating the simulation.
   Upon receiving $(reveal, id, k, d)$ from $\tilde{\mathcal{S}}^*$ in round 3 such that $\mathsf{Open}(c, d, k) = 1$, $Sim_1^{\mathcal{S}}$ simulates the revealing of the encryption key. Therefore, he runs $(revealed, id, k, d) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(reveal, id, k, d)$. In the same round, $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ sends $(revealed, id, \mathbf{x}^*)$ to $\tilde{\mathcal{B}}$. Since $Sim_1^{\mathcal{S}}$ controls the inputs and outputs of $\tilde{\mathcal{B}}$, he replaces the message $(revealed, id, \mathbf{x}^*)$ with $(revealed, id, \mathbf{x})$, where $\mathbf{x} = \mathsf{Dec}(k, \mathbf{z})$.
   If no message $(reveal, id, k, d)$ from $\tilde{\mathcal{S}}^*$ is received in round 3 such that $\mathsf{Open}(c, d, k) = 1$, $Sim_1^{\mathcal{S}}$ sends $(abort, id)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the name of $\tilde{\mathcal{S}}^*$ and waits one round. Then, he simulates the refund to $\tilde{\mathcal{B}}$ by running $(aborted, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(abort, id)$ and terminating the simulation.
4. If $\phi(\mathbf{x}) = 1$, $Sim_1^{\mathcal{S}}$ simulates the finalization of the fair exchange executed by honest Buyer within the real world by running $(sold, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(finalize, id)$, sending $(abort, id, 0)$ in the name of $\tilde{\mathcal{B}}$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$, and terminating. Otherwise, if $\phi(\mathbf{x}) \neq 1$, $Sim_1^{\mathcal{S}}$ needs to simulate the dispute resolution sub-protocol. Therefore, he sets $R = \emptyset$ and executes the following steps alternately starting with (a):
   (a) $Sim_1^{\mathcal{S}}$ checks whether or not honest Buyer is able to generate a valid proof of misbehavior by computing $\pi \leftarrow \mathsf{GenerateProof}(k, \phi, R)$. If $\pi \neq \mathsf{false}$, $Sim_1^{\mathcal{S}}$ simulates a valid proof of misbehavior by running $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(prove, id, \pi)$, sending $(abort, id, 0)$ in the name of $\tilde{\mathcal{S}}^*$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$, outputting $(not\ sold, id)$ through $\tilde{\mathcal{S}}^*$, and terminating the simulation. Otherwise, if $\pi = \mathsf{false}$, $Sim_1^{\mathcal{S}}$ computes $Q \leftarrow \mathsf{NextChallenge}(\phi, R)$ and simulates a challenge query by running $(challenged, id, Q) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(challenge, id, Q)$. He sends $(freeze, id, \tilde{\mathcal{B}}, |Q|)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$, sets $Q_r = Q$, and continues with step (b) in the next round.
   (b) Upon receiving $(respond, id, R_q)$ from $\tilde{\mathcal{S}}^*$, $Sim_1^{\mathcal{S}}$ runs $(responded, id, R_q) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(respond, id, R_q)$, sends $(freeze, id, \tilde{\mathcal{S}}^*, |R_q|)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$, and waits one round. If $\mathsf{ValidateResponse}(Q_r, R_q, r_e) = \mathsf{true}$, $Sim_1^{\mathcal{S}}$ continues immediately with step (a). Otherwise, if $\mathsf{ValidateResponse}(Q_r, R_q, r_e) = \mathsf{false}$, he simulates a *complain*-message by running $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(complain, id)$. Furthermore, $Sim_1^{\mathcal{S}}$ unlocks the coins in favor of $\tilde{\mathcal{B}}$ by sending $(abort, id, 0)$ in the name of corrupted Seller $\tilde{\mathcal{S}}^*$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. Then, he outputs $(not\ sold, id)$ through $\tilde{\mathcal{S}}^*$ and terminates the simulation.
   If no $(respond, id, R_q)$ message is received from $\tilde{\mathcal{S}}^*$, $Sim^{\mathcal{S}}$ sends $(abort, id, 1)$

in the name of $\tilde{\mathcal{S}}^*$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ and waits one round. Then, he runs $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(\textit{respond timeout}, id)$, outputs $(\textit{not sold}, id)$ through $\tilde{\mathcal{S}}^*$, and terminates the simulation.

By internally running the hybrid functionality $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ on either inputs given from $\mathcal{Z}$ or honest Buyer $\mathcal{B}$, the generated transcript and outputs are identical in both executions.

In round 1, $Sim_1^{\mathcal{S}}$ forwards the *sell*-message received from $\tilde{\mathcal{S}}^*$ to Buyer and uses the *initialize*-message to run $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$. Moreover, he creates a *sell-fake*-message which is sent to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. The value $\mathbf{x}^*$ sent within this message does not necessarily have to be the decryption of $\mathbf{z}$, since the $(\textit{revealed}, id, \mathbf{x}^*)$-message in round 3 is replaced by $(\textit{revealed}, id, \mathbf{x})$, where $x = \mathsf{Dec}(k, \mathbf{z})$. This means, the witness received by the honest Buyer is set by the simulator after Seller revealed his key in round 3. Therefore, the environment $\mathcal{Z}$ only sees the correct witness $\mathbf{x}$ and cannot distinguish the protocol execution and the ideal world using this influence.
This is possible, since it is assumed that the simulator controls inputs and outputs of honest Buyer. The same assumption is placed in the first game of the proof presented in [14].
If not both messages, the *sell*-message and the *initialize*-message, are received from $\tilde{\mathcal{S}}^*$ in round 1, the simulation terminates, since both messages are needed to start the protocol Π.

In the revealing phase of Π, Seller has to send his encryption key and the correct opening value to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$. Hence, on receiving a valid *reveal*-message from $\tilde{\mathcal{S}}^*$, $Sim_1^{\mathcal{S}}$ simulates the revealing by running $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(\textit{reveal}, id, k, d)$. As described before, the revealed value $\mathbf{x}^*$ is replaced with the correct witness, which was encrypted by $\mathcal{Z}$ in round 1, i.e., $\mathbf{x}$. In this step, the witness $\mathbf{x}$ is obtained by computing $\mathbf{x} = \mathsf{Dec}(k, \mathbf{z})$, which is also done by the Buyer in the execution of Π. Hence, the obtained witnesses are identical.

Suppose the witness $\mathbf{x}$ is correct, i.e., $\phi(\mathbf{x}) = 1$, the simulator must ensure that the money is transferred to Seller. In the protocol execution, honest Buyer $\mathcal{B}$ sends a *finalize*-message in order to instruct $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ to unlock the money in favor of $\mathcal{S}^*$. Since the witness $\mathbf{x}^*$ within $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ is not necessarily equal to the correct witness $\mathbf{x}$, the simulator must ensure that the money is transferred to Seller. Without interaction from the simulator, the ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ would check $\phi(\mathbf{x}^*)$ and as long as $\mathbf{x}^*$ is not the correct witness, it would send the money to Buyer. In order to send the money to Seller instead, $Sim_1^{\mathcal{S}}$ must send an *abort*-message in the name of $\tilde{\mathcal{B}}$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. This way, $Sim_1^{\mathcal{S}}$ instructs $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ to unfreeze the money in favor of $\tilde{\mathcal{S}}^*$. The delay-parameter of the *abort*-message is set to 0 in order to transfer the money in this round and hence to guarantee that the money is transferred in the same round as in the hybrid world execution.

In case the witness $\mathbf{x}$ is not correct, honest Buyer starts to engage in the dispute resolution sub-protocol. The simulation of the dispute resolution is the main difference compared to the honest simulation and compared to the simulation of the FairSwap protocol given in [14]. The sub-protocol is simulated by $Sim_1^{\mathcal{S}}$ in the steps (4a) and (4b). An honest Buyer tries to create a valid proof of misbehavior as soon as possible. Therefore, $Sim_1^{\mathcal{S}}$ checks whether Buyer is able to create such a proof. If this is possible, he simulates a *prove*-message and sends an *abort*-message to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ on behalf of $\tilde{\mathcal{S}}^*$. This message triggers $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ to send the money back to $\tilde{\mathcal{B}}$ and a *not bought*-message to $\tilde{\mathcal{B}}$. The delay-parameter of 0 guarantees that the money and the message are sent immediately and hence in the same round as in the protocol execution. In addition $Sim_1^{\mathcal{S}}$ creates a *not sold*-message as output for $\tilde{\mathcal{S}}^*$.

If $\tilde{\mathcal{B}}$ cannot create a valid proof of misbehavior, he sends a *challenge*-message to $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$. $Sim_1^{\mathcal{S}}$ creates a *challenge*-message the same way as honest Buyer does and hence the transcript is identical. Since a *challenge*-message instructs $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ to lock some fee from Buyer, $Sim_1^{\mathcal{S}}$ sends a $(\textit{freeze}, id, \tilde{\mathcal{B}}, |Q|)$-message to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. By sending this *freeze*-message, $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ is instructed to lock the same amount of money as in the protocol execution.

After honest Buyer challenged Seller, a *respond*-message must be generated by Seller. $Sim_1^{\mathcal{S}}$ simulates protocol Π by running $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ upon receiving a $(respond, id, R_q)$-message. In addition he instructs $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ to lock fees from $\tilde{\mathcal{S}}^*$. As soon as a *respond*-message is received by honest Buyer $\mathcal{B}$, he checks on the validity of this response. In case the response is not valid, $Sim_1^{\mathcal{S}}$ needs to simulate a *complain*-message and to terminate the simulation by sending an *abort*-message to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the name of $\tilde{\mathcal{S}}^*$. This way, the money is unlocked in favor of $\mathcal{B}$. Again, the delay-parameter of the *abort*-message is set to 0 in order to perform the actions immediately and hence in the same round as in the protocol execution.
If the malicious Seller $\tilde{\mathcal{S}}^*$ does not send a *respond*-message, $Sim_1^{\mathcal{S}}$ aborts the simulation in the name of $\tilde{\mathcal{S}}^*$ and simulates a *respond timeout*-message. By doing this, the money is transferred to honest Buyer. This time, the *abort*-message contains a delay-parameter of 1. This is necessary, since honest Buyer is only able to detect the abort in the next round and therefore can only trigger the payoff with a *respond timeout*-message in the next round.

The timing of the simulation simulates all messages in the same round as in the protocol execution. Since the simulator has the same knowledge as honest Buyer $\mathcal{B}$, which is the set R, the key k, and the circuit ϕ, the generated *challenge*-messages are identical. Moreover, the complete transcript of the dispute resolution sub-protocol is identical to the protocol execution and $Sim_1^{\mathcal{S}}$ ensures that money is locked and unlocked in the same rounds. Hence, the hybrid world execution and the hybrid simulation using simulator $Sim_1^{\mathcal{S}}$ are indistinguishable for any ppt environment $\mathcal{Z}$.

Next, it is necessary to remove the power to control the inputs and outputs of honest Buyer from the simulator. In particular, the *abort*-message in round 4 of $Sim_1^{\mathcal{S}}$ and the replacement of $\mathbf{x}^*$ with $\mathbf{x}$ after $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ revealed the witness have to be removed. This can be achieved by inputting the correct witness $\mathbf{x}$ to the ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. To this end, the same approach as in [14] is taken. The observability feature of the global random oracle $\mathcal{H}$ is used. By querying $\mathcal{H}$ for all already executed queries from outside the session, the simulator is able to obtain the encryption key. Using the key, the simulator is able to extract the witness $\mathbf{x}$ from $\mathbf{z}$ at the start of the simulation. Simulator $Sim^{\mathcal{S}}$ is presented in the following. By showing the indistinguishability between the hybrid simulation using $Sim_1^{\mathcal{S}}$ and the ideal world execution using $Sim^{\mathcal{S}}$, Claim E is shown.

Simulator $Sim^{\mathcal{S}}$ for simulation with corrupted Seller.

1. Upon receiving $(sell, id, \mathbf{z}, \phi)$ and $(initialize, id, c, r_z, r_e, r_\phi, a_\phi, p, f_{\mathcal{S}}, f_{\mathcal{B}})$ through $\tilde{\mathcal{S}}^*$ in the first round, $Sim^{\mathcal{S}}$ obtains Q_{id} by querying $\mathcal{H}(\mathsf{observe})$. If $(k||d, c) \in Q_{id}$, he computes $\mathbf{x} = \mathsf{Dec}(k, \mathbf{z})$. Otherwise, if no such query exists, $Sim^{\mathcal{S}}$ sets $\mathbf{x} = 1^{n \times \lambda}$. Then, he computes $\phi(\mathbf{x})$. If $\phi(\mathbf{x}) = 1$, he sends $(sell, id, \mathbf{x}, \phi, p, f_{\mathcal{S}}, f_{\mathcal{B}})$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. Otherwise, if $\phi(\mathbf{x}) \neq 1$, he sends $(sell\text{-}fake, id, \mathbf{x}, \phi, p, f_{\mathcal{S}}, f_{\mathcal{B}})$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. In addition, $Sim^{\mathcal{S}}$ simulates the execution of Π by running $(active, id, c, r_z, r_e, r_\phi, a_\phi, p, f_{\mathcal{S}}, f_{\mathcal{B}}) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(initialize, id, c, r_z, r_e, r_\phi, a_\phi, p, f_{\mathcal{S}}, f_{\mathcal{B}})$.
   If only $(initialize, id, c, r_z, r_e, r_\phi, a_\phi, p, f_{\mathcal{S}}, f_{\mathcal{B}})$ is received through $\tilde{\mathcal{S}}^*$ in the first round, $Sim^{\mathcal{S}}$ runs $(active, id, c, r_z, r_e, r_\phi, a_\phi, p, f_{\mathcal{S}}, f_{\mathcal{B}}) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(initialize, id, c, r_z, r_e, r_\phi, a_\phi, p, f_{\mathcal{S}}, f_{\mathcal{B}})$.
   If not both messages *sell* and *initialize* are received through $\tilde{\mathcal{S}}^*$ in round one, $Sim^{\mathcal{S}}$ terminates the simulation.
2. Upon receiving $(abort, id)$ through $\tilde{\mathcal{S}}^*$ in the second round, $Sim^{\mathcal{S}}$ sends $(abort, id)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. Furthermore, he simulates Π by running $(aborted, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(abort, id)$ and terminates the simulation.
   If $Sim^{\mathcal{S}}$ receives $(buy, id, \tilde{\mathcal{B}})$ from $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the second round, he simulates the acceptance of $\tilde{\mathcal{B}}$ by running $(initialized, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(accept, id)$.

3. If $\tilde{\mathcal{B}}$ aborts the execution in the third round, $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ leaks $(abort, id, \tilde{\mathcal{B}})$ to $Sim^{\mathcal{S}}$. Then, $Sim^{\mathcal{S}}$ simulates the abort by executing $(aborted, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(abort, id)$ and terminates the simulation.
   Upon receiving $(reveal, id, k, d)$ from $\tilde{\mathcal{S}}^*$ in round 3 such that $\mathsf{Open}(c, d, k) = 1$, $Sim^{\mathcal{S}}$ simulates the revealing of the encryption key. Therefore, he runs $(revealed, id, k, d) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(reveal, id, k, d)$.
   If no message $(reveal, id, k, d)$ from $\tilde{\mathcal{S}}^*$ is received in round 3 such that $\mathsf{Open}(c, d, k) = 1$, $Sim^{\mathcal{S}}$ sends $(abort, id)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the name of $\tilde{\mathcal{S}}^*$ and terminates the simulation.
4. If $\phi(\mathbf{x}) = 1$, $Sim^{\mathcal{S}}$ simulates the finalization of the fair exchange executed by honest Buyer within the real world by running $(sold, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(finalize, id)$ and he terminates. Otherwise, if $\phi(\mathbf{x}) \neq 1$, $Sim^{\mathcal{S}}$ needs to simulate the dispute resolution sub-protocol. Therefore, he sets $R = \emptyset$ and executes the following steps alternately:
   (a) $Sim^{\mathcal{S}}$ checks whether or not honest Buyer is able to generate a valid proof of misbehavior by computing $\pi \leftarrow \mathsf{GenerateProof}(k, \phi, R)$. If $\pi \neq \mathsf{false}$, $Sim^{\mathcal{S}}$ simulates a valid proof of misbehavior by running $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(prove, id, \pi)$ and terminating the simulation. Otherwise, if $\pi = \mathsf{false}$, $Sim^{\mathcal{S}}$ computes $Q \leftarrow \mathsf{NextChallenge}(\phi, R)$ and simulates a challenge query by running $(challenged, id, Q) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(challenge, id, Q)$. He sends $(freeze, id, \tilde{\mathcal{B}}, |Q|)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$, sets $Q_r = Q$, and continues with step (b) in the next round.
   (b) Upon receiving $(respond, id, R_q)$ from $\tilde{\mathcal{S}}^*$, Sim^* runs $(responded, id, R_q) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(respond, id, R_q)$, sends $(freeze, id, \tilde{\mathcal{S}}^*, |R_q|)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$, and waits one round. If $\mathsf{ValidateResponse}(Q_r, R_q, r_e) = \mathsf{true}$, $Sim^{\mathcal{S}}$ continues immediately with step (a). Otherwise, if $\mathsf{ValidateResponse}(Q_r, R_q, r_e) = \mathsf{false}$, he simulates a *complain*-message by running $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(complain, id)$. Furthermore, $Sim^{\mathcal{S}}$ unlocks the coins in favor of $\tilde{\mathcal{B}}$ by sending $(abort, id, 0)$ in the name of corrupted Seller $\tilde{\mathcal{S}}^*$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. Then, he outputs $(not\ sold, id)$ to $\tilde{\mathcal{S}}^*$ and terminates the simulation.
   If no $(respond, id, R_q)$ message is received from $\tilde{\mathcal{S}}^*$, $Sim^{\mathcal{S}}$ sends $(abort, id, 1)$ in the name of $\tilde{\mathcal{S}}^*$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ and waits one round. Then, he runs $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(respond\ timeout, id)$, outputs $(not\ sold, id)$ through $\tilde{\mathcal{S}}^*$, and terminates the simulation.

The differences to the previous simulator $Sim_1^{\mathcal{S}}$ are highlighted in the following. $Sim^{\mathcal{S}}$ must provide the correct witness $\mathbf{x}$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in round 1. To this end, he makes use of the observability property of the global random oracle $\mathcal{H}$. The same argumentation as in [14] is used to show the indistinguishability. By querying $\mathcal{H}(\mathsf{observe})$, $Sim^{\mathcal{S}}$ obtains a list of all queries executed by the environment beforehand. The goal of the simulator is to find out the encryption key used by $\mathcal{Z}$ to create $\mathbf{z}$.
The *initialize*-message received from $\mathcal{Z}$ through $\mathcal{S}^*$ in the first round contains a commitment value c to the used encryption key. Two cases can be distinguished. Either the environment $\mathcal{Z}$ created the commitment correctly or the value c is not a valid commitment.

In case the environment created the commitment using the $\mathsf{Commit}(k)$-algorithm, it posed a query to $\mathcal{H}$ beforehand. Therefore, the set Q_{id} queried by $\mathcal{H}(\mathsf{observe})$ contains a tuple $(k||d, c)$. $Sim^{\mathcal{S}}$ uses the Dec-algorithm to extract the correct witness $\mathbf{x}$ the same way as honest Buyer does after the encryption key is revealed in the protocol execution. Honest Buyer uses the revealed key in round 3 to decrypt $\mathbf{z}$. Hence, it is only possible for $\mathcal{Z}$ to distinguish these two executions if $\mathcal{Z}$ finds a collision of the hash function, i.e., it finds two distinct tuples (k, d) and (k', d') such that $\mathsf{Open}(c, d, k) = \mathsf{Open}(c, d', k') = 1$. Based on

the binding property of the commitment scheme, this is not possible except with negligible probability. Hence, if the commitment was created correctly, the environment $\mathcal{Z}$ cannot distinguish between the hybrid and the ideal world execution as long as the commitment scheme is binding.

Now, considering the case that the environment computed the commitment value c incorrectly. In this case, $\mathcal{Z}$ has to guess a key k and an opening value d such that $\mathsf{Open}(c, d, k) = 1$. The Open-algorithm only returns 1, if $\mathcal{H}$ returns c on input $(k||d)$. Since the random oracle $\mathcal{H}$ samples the query response randomly over $\{0,1\}^\mu$, the probability to correctly guess k and d is $\frac{1}{2^\mu}$, which is negligible for large μ. Suppose $\mathcal{Z}$ is not able to guess k and d correctly, $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ does not accept the *reveal*-message from $\mathcal{S}^*$ in the hybrid world and honest Buyer can abort the protocol execution in order to get his money back. In the ideal world, $Sim^{\mathcal{S}}$ simulates the exact same behavior. If he does not receive a *reveal*-message from $\mathcal{Z}$ through $\tilde{\mathcal{S}}^*$ in round 3 such that the Open-algorithm outputs 1, he sends an *abort*-message to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the name of $\tilde{\mathcal{S}}^*$. This message instructs the ideal functionality to unfreeze the money in favor of $\mathcal{B}$.

Since the witness provided as input to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the first round equals the witness $\mathbf{x}$, $Sim^{\mathcal{S}}$ gets rid of the *abort*-message in (4a) after the honest Buyer created a valid proof of misbehavior. Without the *abort*-message, the *Payout*-phase of $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ is triggered which checks $\phi(\mathbf{x})$. Based on the fact that honest Buyer is only able to create a valid proof if the output of the evaluation is 0, $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ sends the money to $\tilde{\mathcal{B}}$ in the *Payout*-phase. This way, the *Payout*-phase simulates the exact same behavior as the protocol execution in the hybrid world.

To conclude, any environment $\mathcal{Z}$ that only corrupts Seller cannot distinguish the hybrid and the ideal world execution except with negligible probability as long as the underlying commitment scheme is binding.

Lastly, it is worth mentioning that the simulator $Sim^{\mathcal{S}}$ runs in polynomial time. This is not trivial, since the simulation of the dispute resolution sub-protocol consists of repetitive actions. Nevertheless, the number of challenged circuit gates is limited by the parameter a_ϕ which is set in step 1. Hence, the total number of challenge queries is upper bounded by this parameter. Since an honest Buyer tries to create a valid proof of misbehavior as soon as possible, the whole simulation terminates after at most a_ϕ challenge queries.

Simulation with Corrupted Buyer

Next, the case of a corrupted Buyer is considered. It is symmetric to the setup in case of a corrupted Seller. In detail, the internal state and the program code of corrupted Buyer $\mathcal{B}^*$ is under full control of the environment $\mathcal{Z}$. Thus, $\mathcal{B}^*$ may deviate from the protocol at any point in time. The ideal world comprises the ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$, the dummy parties $\tilde{\mathcal{S}}$ and $\tilde{\mathcal{B}}^*$, and the simulator $Sim^{\mathcal{B}}$. $Sim^{\mathcal{B}}$ needs to generate a transcript of the execution of Π as well as he needs to define all inputs and outputs of corrupted Buyer. This includes the generation of all outputs from $\tilde{\mathcal{B}}^*$ towards the environment and towards $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. Hence, on messages from $\mathcal{Z}$ through $\tilde{\mathcal{B}}^*$, $Sim^{\mathcal{B}}$ needs to generate inputs to the ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. Again, the proof of the following Claim focuses on the differences between the simulation with an honest Buyer and with a corrupted Buyer.

Claim. There exists an efficient algorithm $Sim^{\mathcal{B}}$ such that for all ppt environments $\mathcal{Z}$ that *only corrupt Buyer* it holds that the execution of Π in the $(\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}, \mathcal{L}, \mathcal{H})$-hybrid world in presence of adversary $\mathcal{A}$ is indistinguishable from the ideal world execution of $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ with the ideal adversary $Sim^{\mathcal{B}}$.

Proof. The difficulty of the simulation with a corrupted Buyer is to create an encrypted witness $\mathbf{z}$ in the first round and to present an encryption key k in the third round such that the decryption of $\mathbf{z}$ using k equals the correct witness $\mathbf{x}$, which is known to the simulator only after round 3. The values z_i, for $i \in [n]$, are also contained in the vector $\mathbf{e}$ of intermediate

values of the computation of $\phi(\mathbf{x})$. Since the witness $\mathbf{x}$ is not known in round one, the simulator must ensure that as soon as $\mathbf{x}$ is revealed, the decryption of each e_i equals the output of the circuit gate ϕ_i on evaluating $\phi(\mathbf{x})$.
Moreover, a commitment c to the encryption key k must be created in the first round. When revealing the key k in the third round, an opening value d has to be provided such that d opens the commitment c to k.

We use again a sequence of games in order to show indistinguishability between the hybrid world and the ideal world execution. The same approach is taken in [14], but the following proof is shortened to two games. In $Game_1$, simulator $Sim_1^{\mathcal{B}}$ has the additional power to control all inputs and outputs of honest parties, i.e., of honest Seller. Hence, he learns the correct witness $\mathbf{x}$ at the start of the execution which he would not have known otherwise. Using $\mathbf{x}$, $Sim_1^{\mathcal{B}}$ samples a key k and creates an encrypted witness $\mathbf{z}$ and the intermediate values $\mathbf{e}$ using the $\mathsf{Presetup}$-algorithm.

In contrast to the proof in [14], the presented simulator needs to simulate the dispute resolution sub-protocol. $Sim_1^{\mathcal{B}}$ is given as follows.

Simulator $Sim_1^{\mathcal{B}}$ for hybrid simulation with corrupted Buyer.

1. The simulation starts when $\tilde{\mathcal{S}}$ sends $(sell, id, \mathbf{x}, \phi, p, f_{\mathcal{S}}, f_{\mathcal{B}})$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$, where $\phi(\mathbf{x}) = 1$. $Sim_1^{\mathcal{B}}$ learns the witness $\mathbf{x}$ during this step. He samples a key $k \leftarrow \mathsf{Gen}(1^\kappa)$, computes $(c, d) \leftarrow \mathsf{Commit}(k)$ and $(\mathbf{z}, r_z, r_e, r_\phi) \leftarrow \mathsf{Presetup}(\phi, \mathbf{x}, k)$. Then, he simulates the execution of Π by running $(active, id, c, r_z, r_e, r_\phi, a_\phi, p, f_{\mathcal{S}}, f_{\mathcal{B}}) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(initialize, id, c, r_z, r_e, r_\phi, a_\phi, p, f_{\mathcal{S}}, f_{\mathcal{B}})$. In addition, he sends the message $(sell, id, \mathbf{z}, \phi)$ to $\tilde{\mathcal{B}}^*$.
2. Upon receiving $(abort, id, \tilde{\mathcal{S}})$ from $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the second round, $Sim_1^{\mathcal{B}}$ simulates the abort by running $(aborted, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(abort, id)$ and terminating the simulation.
   Upon receiving $(accept, id)$ from $\tilde{\mathcal{B}}^*$ in the second round, $Sim_1^{\mathcal{B}}$ sends (buy, id, ϕ) to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ and simulates the acceptance of $\tilde{\mathcal{B}}^*$ by running $(initialized, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(accept, id)$.
3. If $Sim_1^{\mathcal{B}}$ receives $(abort, id)$ from $\tilde{\mathcal{B}}^*$ in the third round, he sends $(abort, id)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$, runs $(aborted, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(abort, id)$, and terminates the simulation.
   Otherwise, $Sim_1^{\mathcal{B}}$ simulates the encryption key revealing by running $(revealed, id, k, d) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(reveal, id, k, d)$. Moreover, $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ sends $(revealed, id, \mathbf{x})$ to $\tilde{\mathcal{B}}^*$ in the third round if $\tilde{\mathcal{B}}^*$ did not abort.
4. If $Sim_1^{\mathcal{B}}$ receives $(finalize, id)$ from $\tilde{\mathcal{B}}^*$ in the fourth round, he simulates the finalization of the protocol Π by running $(sold, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(finalize, id)$ and terminating.

   If $\tilde{\mathcal{B}}^*$ sends a $(challenge, id, Q$ message, $Sim_1^{\mathcal{B}}$ needs to simulate the dispute resolution sub-protocol. Therefore, he executes the following steps alternately:
   (a) Upon receiving $(challenge, id, Q)$ from $\tilde{\mathcal{B}}^*$ when $|Q| \leq a_\phi$, $Sim_1^{\mathcal{B}}$ runs $(challenged, id, Q) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(challenge, id, Q)$, sets $a_\phi = a_\phi - |Q|$, sends $(freeze, id, \tilde{\mathcal{B}}^*, |Q|)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$, waits one round, and proceeds with step (b).
   Upon receiving $(prove, id, \pi)$ from $\tilde{\mathcal{B}}^*$, $Sim_1^{\mathcal{B}}$ needs to simulate an invalid proof of misbehavior. Therefore, he sends no message to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in order to trigger the *Payout*-phase of it and runs $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(prove, id, \pi)$. Then, he terminates the simulation.
   Upon receiving $(complain, id)$ from $\tilde{\mathcal{B}}^*$, $Sim_1^{\mathcal{B}}$ needs to simulate an invalid *complain*-message. To this end, he sends no message to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in order to trigger the *Payout*-phase, runs $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(complain, id)$, and terminates the simulation.
   If no message is received from $\tilde{\mathcal{B}}^*$ during the challenge-response-phase, $Sim_1^{\mathcal{B}}$

sends $(abort, id, 1)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the name of $\tilde{\mathcal{B}}^*$ and waits one round. Then, he runs $(sold, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(challenge\ timeout, id)$, to simulate the *timeout*-message sent by honest Seller, and outputs $(bought, id, \mathbf{x})$ to $\tilde{\mathcal{B}}^*$.

(b) After Buyer challenged Seller, $Sim_1^{\mathcal{B}}$ needs to simulate a valid response. To this end, he computes $R_q \leftarrow \mathsf{GenerateResponse}(\phi, \mathbf{x}, k, Q)$ and sends $(freeze, id, \tilde{\mathcal{S}}, |R_q|)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. He simulates Π by running $(responded, id, R_q) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(respond, id, R_q)$. Then, he waits one round, and proceeds with step (a).

If no message is received from $\tilde{\mathcal{B}}^*$ in round 4, $Sim_1^{\mathcal{B}}$ sends $(abort, id, 1)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the name of $\tilde{\mathcal{B}}^*$ and waits one round. Then, he runs $(sold, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(finalize, id)$, to simulate the finalization executed by honest Seller, and outputs $(bought, id, \mathbf{x})$ to $\tilde{\mathcal{B}}^*$.

Based on the extra power of the simulator $Sim_1^{\mathcal{B}}$, he learns the witness $\mathbf{x}$ in the first round before it is sent to the ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. Since Seller is honest, the witness $\mathbf{x}$ is correct in the sense that $\phi(\mathbf{x}) = 1$, otherwise the ideal functionality won't accept the *sell*-message.
$Sim_1^{\mathcal{B}}$ can execute the initialization steps in the same way as honest Seller can, because he knows the witness. Hence, after the initialization, it is guaranteed that the encrypted witness $\mathbf{z}$ can be decrypted to $\mathbf{x}$ using the key k which is sampled by $Sim_1^{\mathcal{B}}$.
Also the computation $\phi(\mathbf{x})$ can be executed before the commitment r_e has to be generated and thus the decryption of any value e_i, which is verifiable with a Merkle proof and the commitment r_e, equals the correct output of circuit gate ϕ_i, for each $i \in [m]$.

The simulation of round 2 and 3 is straightforward and similar to the honest case with the addition that the simulator must generate the input to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ from $\tilde{\mathcal{B}}^*$. In round 4, corrupted Buyer can either send a *finalize*-message, abort the execution, or start the dispute resolution sub-protocol. Again, the simulation of the dispute resolution sub-protocol is a main difference in comparison to the proof of the FairSwap protocol presented in [14].

In case of a *finalize*-message received from $\tilde{\mathcal{B}}^*$, $Sim_1^{\mathcal{B}}$ simulates the finalization of Π and terminates the simulation. Since no message is sent to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in this round, the ideal functionality executes the *Payout*-phase and transfers the money to honest Seller. If $\tilde{\mathcal{B}}^*$ aborts the execution, $Sim_1^{\mathcal{B}}$ sends a $(abort, id, 1)$-message to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. The delay-parameter is set to 1 in order to delay the unlocking of the coins by one round. This results in the unlocking of the coins in favor of $\tilde{\mathcal{S}}$ in the next round, which corresponds to the *finalize*-message sent by honest Seller in the protocol Π. This message is simulated by the internal execution of $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ after $Sim_1^{\mathcal{B}}$ waited one round.

Even if an honest Seller always provides a correct witness $\mathbf{x}$, a corrupted Buyer may engage in the dispute resolution sub-protocol. Therefore, when receiving a *challenge*-message from $\tilde{\mathcal{B}}^*$, $Sim_1^{\mathcal{B}}$ simulates the execution of $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ internally. The simulator keeps track of the upper bound of possible challenged gates using the parameter a_ϕ. This guarantees that corrupted Buyer may not pose more challenges than allowed which is controlled by $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ in the hybrid world execution. In addition, $Sim_1^{\mathcal{B}}$ sends a *freeze*-message to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in order to lock the right amount of fees.
After $\tilde{\mathcal{B}}^*$ sent a *challenge*-message, an honest Seller is always able to create valid responses. This is simulated by $Sim_1^{\mathcal{B}}$ in step (4b) by using the $\mathsf{generateResponse}$-algorithm and internally running $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$. Again, the simulator sends a *freeze*-message to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ to lock the same money from $\tilde{\mathcal{S}}$ as in the hybrid world execution.

Corrupted Buyer may also send a *complain*-message or a *prove*-message after $\tilde{\mathcal{S}}$ answered a challenge-query. However, these messages are invalid, since honest Seller provided the correct witness and answers always with correct responses. Thus, $Sim_1^{\mathcal{B}}$ internally runs $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ and sends no message to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. The absence of a message results in the execution of

the *Payout*-phase of $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. Since the provided witness is correct, the money is transferred to honest Seller, which corresponds to the same behavior as in the protocol execution.

Finally, corrupted Buyer may also abort at any point in time during the *challenge-response*-phase. In this case, $Sim_1^{\mathcal{B}}$ sends an *abort*-message to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ and waits one round. The wait is necessary, since the *challenge timeout*-message from $\tilde{\mathcal{S}}$ appears in the next round after $\tilde{\mathcal{B}}^*$ aborted. The *abort*-message with delay-parameter 1 instructs $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ to send the money to $\tilde{\mathcal{S}}$ in the next round. Hence, the money is sent in the identical round as in the protocol execution.

$Sim_1^{\mathcal{B}}$ is used in the hybrid simulation in $Game_1$. $Game_1$ is indistinguishable from the hybrid world execution but the simulator has extra power. $Game_2$ removes this extra power from the simulator to give a full UC simulation. In order to achieve indistinguishability of $Game_1$ and $Game_2$, the programming feature of the global random oracle $\mathcal{H}$ is used. The following argumentation about the indistinguishability is similar to the once given in [14].

Instead of learning the witness $\mathbf{x}$ in the first round, $Sim^{\mathcal{B}}$ only gets to know the witness after the ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ reveals it in round three. Since, the encrypted witness $\mathbf{z}^*$ must be given to $\tilde{\mathcal{B}}^*$ in the first round, $\mathbf{z}^*$ is sampled randomly. In addition, the whole vector of intermediate values $\mathbf{e}^*$ is sampled randomly to create a commitment r_e^* to it.
In contrast to the simulation in [14], the commitment c to the encryption key k is not sampled randomly. Instead, $Sim^{\mathcal{B}}$ samples a key and creates the commitment c using the Commit-algorithm. The simulation in [14] has to use the programming feature of $\mathcal{H}$ since the commitment c is defined by the environment $\mathcal{Z}$ and hence cannot be chosen by the simulator. However, the absence of the commitment c in the messages from $\mathcal{Z}$ to Seller does not affect the security of the protocol but it simplifies the simulation.
Based on the IND-CPA-security of the underlying encryption scheme, the environment $\mathcal{Z}$ cannot distinguish between $\mathbf{z}$ computed as the encryption of the correct witness $\mathbf{x}$ and a randomly selected $\mathbf{z}^*$. When the correct witness $\mathbf{x}$ is revealed in round 3, the simulator has to ensure that the decryption of $\mathbf{z}^*$ equals $\mathbf{x}$ and the decryption of each e_i equals the output of ϕ_i of the evaluation on $\mathbf{x}$ for each $i \in [m]$. To this end, the programming feature of the global random oracle $\mathcal{H}$ is used. To give an example, for $i \in [n]$ the message $(\mathsf{program}, id, (k||i), o_i \oplus z_i^*)$ is sent to $\mathcal{H}$, where $o_i = x_i$. This results in the situation, that the global random oracle $\mathcal{H}$ returns $r_i = o_i \oplus z_i^*$ on input $\mathcal{H}(k||i)$ such that the encryption algorithm $\mathsf{Enc}(k, z_i^*) = z_i^* \oplus \mathcal{H}(k||i) = o_i$ will decrypt z_i^* to $o_i = x_i$.

Since $Sim^{\mathcal{B}}$ programs the output of several queries, it might seem possible for $\mathcal{Z}$ to detect the programming and distinguish the two execution this way. However, the $\mathsf{isProgrammed}$-queries are executed either through corrupted Buyer $\tilde{\mathcal{B}}^*$ or through the ideal adversary. In both cases, $Sim^{\mathcal{B}}$ controls the responses and is able to lie by answering each query with $(\mathsf{isProgrammed}, 0)$. Hence, the programming can only be detected by $\mathcal{Z}$ if it sends the exact same queries that are programmed by $Sim^{\mathcal{B}}$ in round 3 beforehand.
Since the query values are randomly and $\mathcal{Z}$ can only execute polynomial many queries, the probability to detect a programming this way is negligible.

The full simulator $Sim^{\mathcal{B}}$ is given below. By achieving indistinguishability between $Game_1$ and $Game_2$, it is shown that the hybrid world execution is indistinguishable from the ideal world execution.

Simulator $Sim^{\mathcal{B}}$ for simulation with corrupted Buyer.

1. Upon receiving $(sell, id, \phi, p, f_{\mathcal{S}}, f_{\mathcal{B}}, \tilde{\mathcal{S}})$ from $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the first round, $Sim^{\mathcal{B}}$ randomly samples $z_i^* \leftarrow \{0,1\}^{\lambda}$ for $i \in [n]$, $e_j^* \leftarrow \{0,1\}^{\mu}$ for $j \in \{n+1, \ldots, m\}$, and $k^* \leftarrow \mathsf{Gen}(1^{\kappa})$. He sets $\mathbf{z}^* = (z_1^*, \ldots, z_n^*)$ and $\mathbf{e}^* = (z_1^*, \ldots, z_n^*, e_{n+1}^* \ldots, e_m^*)$. Then, he computes $(c^*, d^*) \leftarrow \mathsf{Commit}(k^*)$, $r_z^* = \mathsf{MTHash}(\mathbf{z}^*)$, $r_e^* = \mathsf{MTHash}(\mathbf{e}^*)$, and $r_\phi = \mathsf{MTHash}(\phi)$. $Sim^{\mathcal{B}}$ simulates the execution of Π by running $(active, id, c^*, r_z^*, r_e^*, r_\phi, a_\phi, p, f_{\mathcal{S}}, f_{\mathcal{B}}) \leftarrow$

$\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(initialize, id, c^*, r_z^*, r_e^*, r_\phi, a_\phi, p, f_\mathcal{S}, f_\mathcal{B})$, where a_ϕ is the challenge limit property of ϕ, and generating the message $(sell, id, \mathbf{z}^*, \phi)$ from $\tilde{\mathcal{S}}$ to $\tilde{\mathcal{B}}^*$.

2. Upon receiving $(abort, id, \tilde{\mathcal{S}})$ from $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the second round, $Sim^{\mathcal{B}}$ simulates the abort by running $(aborted, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(abort, id)$ and terminating the simulation.
   Upon receiving $(accept, id)$ from $\tilde{\mathcal{B}}^*$ in the second round, $Sim^{\mathcal{B}}$ sends (buy, id, ϕ) to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ and simulates the acceptance of $\tilde{\mathcal{B}}^*$ by running $(initialized, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(accept, id)$.
3. If $Sim^{\mathcal{B}}$ receives $(abort, id)$ from $\tilde{\mathcal{B}}^*$ in the third round, he sends $(abort, id)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$, runs $(aborted, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(abort, id)$, and terminates the simulation.
   Otherwise, $Sim^{\mathcal{B}}$ simulates the key revealing executed by honest Seller by running $(revealed, id, k^*, d^*) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(reveal, id, k^*, d^*)$. In the same round, $Sim^{\mathcal{B}}$ learns $\mathbf{x}$ from the message $(revealed, id, \mathbf{x})$, which is sent from $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ to $\tilde{\mathcal{B}}^*$. Then, $Sim^{\mathcal{B}}$ needs to map the randomly selected values in the first round to the correct witness by performing the following steps.
   - For all $i \in [n]$ set $o_i = x_i$ and for all $i \in \{n+1, \ldots, m\}$ and $\phi_i := (i, op_i, I_i)$ compute $o_i := op_i(o_{I_i[1]}, \ldots, o_{I_i[l]})$.
   - Then, program the random oracle $\mathcal{H}$ in such a way that the decryption of $\mathbf{z}^*$ and $\mathbf{e}^*$ equals the correct witness and the intermediate outputs of $\phi(\mathbf{x})$, respectively. Therefore, send the message $(\mathsf{program}, id, (k||i), o_i \oplus z_i^*)$ for all $i \in [n]$ and $(\mathsf{program}, id, (k||i), o_i \oplus e_i^*)$ for all $i \in \{n+1, \ldots, m\}$.
4. If $Sim^{\mathcal{B}}$ receives $(finalize, id)$ from $\tilde{\mathcal{B}}^*$ in the fourth round, he simulates the finalization of the protocol Π by running $(sold, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(finalize, id)$ and terminating.

   If $\tilde{\mathcal{B}}^*$ sends a $(challenge, id, Q)$ message, $Sim^{\mathcal{B}}$ needs to simulate the dispute resolution sub-protocol. Therefore, he executes the following steps alternately:
   (a) Upon receiving $(challenge, id, Q)$ from $\tilde{\mathcal{B}}^*$ when $|Q| \leq a_\phi$, $Sim^{\mathcal{B}}$ runs $(challenged, id, Q) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(challenge, id, Q)$, sets $a_\phi = a_\phi - |Q|$, sends $(freeze, id, \tilde{\mathcal{B}}^*, |Q|)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$, waits one round, and proceeds with step (b).
   Upon receiving $(prove, id, \pi)$ from $\tilde{\mathcal{B}}^*$, $Sim^{\mathcal{B}}$ needs to simulate an invalid proof of misbehavior. Therefore, he sends no message to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in order to trigger the *Payout*-phase of it and runs $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(prove, id, \pi)$. Then, he terminates the simulation.
   Upon receiving $(complain, id)$ from $\tilde{\mathcal{B}}^*$, $Sim^{\mathcal{B}}$ needs to simulate an invalid *complain*-message. To this end, he sends no message to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in order to trigger the *Payout*-phase, runs $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(complain, id)$, and terminates the simulation.
   If no message is received from $\tilde{\mathcal{B}}^*$ during the challenge-response-phase, $Sim^{\mathcal{B}}$ sends $(abort, id, 1)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the name of $\tilde{\mathcal{B}}^*$ and waits one round. Then, he runs $(sold, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(challenge\ timeout, id)$, to simulate the *timeout*-message sent by honest Seller, and outputs $(bought, id, \mathbf{x})$ to $\tilde{\mathcal{B}}^*$.
   (b) After Buyer challenged honest Seller, $Sim^{\mathcal{B}}$ needs to simulate a valid response. To this end, he computes $R_q \leftarrow \mathsf{GenerateResponse}(\phi, \mathbf{x}, k^*, Q)$ and sends $(freeze, id, \tilde{\mathcal{S}}, |R_q|)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. He simulates Π by running $(responded, id, R_q) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(respond, id, R_q)$. Then, he waits one round, and proceeds with step (a).
   If no message is received from $\tilde{\mathcal{B}}^*$ in round 4, $Sim^{\mathcal{B}}$ sends $(abort, id, 1)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the name of $\tilde{\mathcal{B}}^*$ and waits one round. Then, he runs $(sold, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(finalize, id)$, to simulate the finalization executed by honest Seller, and outputs $(bought, id, \mathbf{x})$ to $\tilde{\mathcal{B}}^*$.

Simulation with Two Corrupted Parties

The case of two corrupted parties, i.e., a corrupted Seller and a corrupted Buyer is a combination of the two previous cases in which only one party is corrupted. Since both parties are corrupted, the environment $\mathcal{Z}$ has full control over $\mathcal{S}^*$ and $\mathcal{B}^*$ in the hybrid world execution. The ideal world consists of the two corrupted dummy parties $\tilde{\mathcal{S}}^*$ and $\tilde{\mathcal{B}}^*$, the ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$, and the ideal adversary $Sim^{\mathcal{SR}}$. $Sim^{\mathcal{SR}}$ is responsible for all inputs and outputs of the dummy parties towards the ideal functionality. Hence, on receiving a message through a dummy party from the environment $\mathcal{Z}$, $Sim^{\mathcal{SR}}$ needs to generate the appropriate inputs to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ and to simulate the execution of Π. In addition, the outputs of $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ must be converted into outputs to $\mathcal{Z}$ through the dummy parties.

It is important to note that the protocol does not provide any guarantees if none of the parties behaves honestly. Especially, the protocol may never terminate and coins may be locked beyond the protocol execution. This is easy to see when considering a protocol execution in which neither of the two parties sends a *finalize*-message or any other message in round 4. The protocol waits for the next message and hence does not terminate. Moreover, the p coins are locked forever.
This example shows that considering a protocol execution with two corrupted parties is not reasonable in the given two-party protocol for a fair exchange. However, to allow composability of Π and other protocols, the indistinguishability in this corruption setup has to be shown. The proof of the following Claim makes use of techniques already used within the single corruption setups. Therefore, the proof focuses on the most challenging aspect of the simulation and is based on the argumentation already stated in the single corruption scenarios.

Claim. There exists an efficient algorithm $Sim^{\mathcal{SB}}$ such that for all ppt environments $\mathcal{Z}$ that *corrupt both, Seller and Buyer,* it holds that the execution of Π in the $(\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}, \mathcal{L}, \mathcal{H})$-hybrid world in presence of adversary $\mathcal{A}$ is indistinguishable from the ideal world execution of $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ with the ideal adversary $Sim^{\mathcal{SB}}$.

Proof. In the scenario of two corrupted parties, i.e., a corrupted Seller $\tilde{\mathcal{S}}^*$ and a corrupted Buyer $\tilde{\mathcal{B}}^*$, the simulator $Sim^{\mathcal{SB}}$ controls all outputs of these parties towards the environment $\mathcal{Z}$ and he needs to provide inputs to the ideal functionality $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ on behalf of the corrupted parties. $Sim^{\mathcal{SB}}$ is a combination of the simulators in the single corruption cases presented previously.

Simulator $Sim^{\mathcal{SB}}$ for simulation with corrupted Seller and corrupted Buyer.

1. Upon receiving $(initialize, id, c, r_z, r_e, r_\phi, a_\phi, p, f_{\mathcal{S}}, f_{\mathcal{B}})$ from $\tilde{\mathcal{S}}^*$ in the first round, $Sim^{\mathcal{SR}}$ simulates the execution of Π by running $(active, id, c, r_z, r_e, r_\phi, a_\phi, p, f_{\mathcal{S}}, f_{\mathcal{B}}) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(initialize, id, c, r_z, r_e, r_\phi, a_\phi, p, f_{\mathcal{S}}, f_{\mathcal{B}})$. If $Sim^{\mathcal{SR}}$ also receives $(sell, id, \mathbf{z}, \phi)$ in round 1 through $\tilde{\mathcal{S}}^*$, he sets $\mathbf{x}^* = 1^{n \times \lambda}$ and sends $(sell\text{-}fake, id, \mathbf{x}^*, \phi, p, f_{\mathcal{S}}, f_{\mathcal{B}})$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$.
   If not both messages, *sell* and *initialize*, are received through $\tilde{\mathcal{S}}^*$ in round one, $Sim^{\mathcal{SR}}$ terminates the simulation.
2. Upon receiving $(abort, id)$ through $\tilde{\mathcal{S}}^*$ in the second round, $Sim^{\mathcal{SR}}$ sends $(abort, id)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. Furthermore, he simulates Π by running $(aborted, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(abort, id)$ and terminating the simulation.
   Upon receiving $(accept, id)$ through $\tilde{\mathcal{B}}^*$ in the second round, $Sim^{\mathcal{SR}}$ sends (buy, id, ϕ) to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ and simulates the acceptance of $\tilde{\mathcal{B}}^*$ by running $(initialized, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(accept, id)$.
   If no message was received in round 2, $Sim^{\mathcal{SR}}$ terminates the simulation.

3. If $Sim^{\mathcal{SR}}$ receives $(abort, id)$ from $\tilde{\mathcal{B}}^*$ in the third round, he sends $(abort, id)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$, runs $(aborted, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(abort, id)$, and terminates the simulation.
   Upon receiving $(reveal, id, k, d)$ through $\tilde{\mathcal{S}}^*$ in round 3 such that $\mathsf{Open}(c, d, k) = 1$, $Sim^{\mathcal{SR}}$ simulates the revealing of the encryption key, Therefore, he runs $(revealed, id, k, d) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(reveal, id, k, d)$. In addition, he computes $\mathbf{x} = \mathsf{Dec}(k, \mathbf{z})$ and replaces the message $(revealed, id, \mathbf{x}^*)$ from $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ to $\tilde{\mathcal{B}}^*$ with $(revealed, id, \mathbf{x})$.
   If no message is received in round 3, $Sim^{\mathcal{SR}}$ simulates the abort of both parties by sending $(abort, id)$ in the name of $\tilde{\mathcal{S}}^*$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ and sending a $(block, id)$ message in the name of $\tilde{\mathcal{B}}^*$ to $\mathcal{L}$. Then, he terminates the simulation.
4. If $Sim^{\mathcal{SR}}$ receives $(finalize, id)$ from $\tilde{\mathcal{B}}^*$ in the fourth round, he simulates the finalization of the protocol Π by running $(sold, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(finalize, id)$. Then, he sends $(abort, id, 0)$ in the name of $\tilde{\mathcal{B}}^*$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$, outputs $(bought, id, \mathbf{x})$ through $\tilde{\mathcal{B}}^*$, and terminates.
   If $\tilde{\mathcal{B}}^*$ sends a $(challenge, id, Q)$ message, $Sim^{\mathcal{SR}}$ needs to simulate the dispute resolution sub-protocol. Therefore, he executes the following steps alternately starting with (a):
   (a) Upon receiving $(challenge, id, Q)$ from $\tilde{\mathcal{B}}^*$ when $|Q| \leq a_\phi$, $Sim^{\mathcal{SB}}$ runs $(challenged, id, Q) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(challenge, id, Q)$, sets $a_\phi = a_\phi - |Q|$ and $Q_r = Q$, sends $(freeze, id, \tilde{\mathcal{B}}^*, |Q|)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$, waits one round, and proceeds with step (b).
   Upon receiving $(prove, id, \pi)$ from $\tilde{\mathcal{B}}^*$, $Sim^{\mathcal{SB}}$ needs to simulate a judgment on the given proof of misbehavior. Therefore, he runs $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(prove, id, \pi)$ and computes $\mathsf{Judge}(k, r_z, r_e, r_\phi, \pi)$. If the result is 1, $Sim^{\mathcal{SB}}$ sends $(abort, id, 0)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ on behalf of $\tilde{\mathcal{S}}^*$ in order to trigger the payoff in favor of Buyer. Additionally, he outputs $(not\ sold, id)$ through $\tilde{\mathcal{S}}^*$ and terminates. Otherwise, if the result is 0, $Sim^{\mathcal{SB}}$ sends $(abort, id, 0)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ on behalf of $\tilde{\mathcal{B}}^*$, outputs $(bought, id, \mathbf{x})$ through $\tilde{\mathcal{B}}^*$, and terminates.
   Upon receiving $(complain, id)$ from $\tilde{\mathcal{B}}^*$, $Sim^{\mathcal{SB}}$ needs to simulate a judgment on the given *complain*-message. To this end, he computes $\mathsf{ValidateResponse}(Q_r, R_r, r_e)$. If the output is false, $Sim^{\mathcal{SB}}$ sends $(abort, id, 0)$ in the name of $\tilde{\mathcal{S}}^*$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ and outputs $(not\ sold, id)$ through $\tilde{\mathcal{S}}^*$. Otherwise, if the output is true, he sends $(abort, id, 0)$ in the name of $\tilde{\mathcal{B}}^*$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ and outputs $(bought, id)$ through $\tilde{\mathcal{B}}^*$. In both cases, he runs $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(complain, id)$, and terminates the simulation.
   If no message is received from $\tilde{\mathcal{B}}^*$ in this round, $Sim^{\mathcal{SB}}$ sends $(abort, id, 1)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the name of $\tilde{\mathcal{B}}^*$ and waits one round. Then, if he receives a $(challenge\ timeout, id)$ from $\tilde{\mathcal{S}}^*$, he simulates Π by running $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(challenge\ timeout, id)$. Otherwise, if no message is received from $\tilde{\mathcal{S}}^*$, $Sim^{\mathcal{SB}}$ needs to refuse the unfreezing by sending a $(block, id)$ message in the name of $\tilde{\mathcal{S}}^*$ to $\mathcal{L}$. Afterwards, he terminates the simulation.
   (b) Upon receiving $(respond, id, R_q)$ from $\tilde{\mathcal{S}}^*$, $Sim^{\mathcal{SB}}$ runs $(responded, id, R_q) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(respond, id, R_q)$, sets $R_r = R_q$, sends $(freeze, id, \tilde{\mathcal{S}}, |R_q|)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$, waits one round, and continues with step (a).
   If no message is received from $\tilde{\mathcal{S}}^*$ in this round, $Sim^{\mathcal{SB}}$ sends $(abort, id, 1)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the name of $\tilde{\mathcal{S}}^*$ and waits one round. Then, if he receives a $(response\ timeout, id)$ from $\tilde{\mathcal{B}}^*$, he simulate Π by running $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(respond\ timeout, id)$. Otherwise, if no message is received from $\tilde{\mathcal{B}}^*$,

$Sim^{\mathcal{SB}}$ needs to refuse the unfreezing by sending a $(block, id)$ message in the name of $\tilde{\mathcal{B}}^*$ to $\mathcal{L}$. Afterwards, he terminates the simulation.

If no message is received from $\tilde{\mathcal{B}}^*$ in round 4, $Sim^{\mathcal{SB}}$ sends $(abort, id, 1)$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the name of $\tilde{\mathcal{B}}^*$ and waits one round. Then, if he receives $(finalize, id)$ from $\tilde{\mathcal{S}}^*$, he runs $(sold, id) \leftarrow \mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}(finalize, id)$, to simulate the finalization executed by the Seller, and outputs $(bought, id, \mathbf{x})$ to $\tilde{\mathcal{B}}^*$. Otherwise, if no $(finalize, id)$ message is received from $\tilde{\mathcal{S}}^*$, $Sim^{\mathcal{SB}}$ needs to refuse the unfreezing by sending a $(block, id)$ message in the name of $\tilde{\mathcal{S}}^*$ to $\mathcal{L}$. Afterwards, he terminates the simulation.

The simulation is straightforward with some exceptions. First, it is noteworthy that the simulator provides a wrong witness $\mathbf{x}^*$ as input to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ in the first round. Since $Sim^{\mathcal{SB}}$ controls all outputs of the corrupted parties towards $\mathcal{Z}$, he can replace the message $(revealed, id, \mathbf{x}^*)$ in round 3 from $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ to $\tilde{\mathcal{B}}^*$ with the message $(revealed, id, \mathbf{x})$, where $\mathbf{x} = \mathsf{Dec}(k, \mathbf{z})$ is computed in round 3 after the Seller revealed the encryption key. This way, the *revealed*-message in the ideal world execution carries the same witness $\mathbf{x}$ as in the hybrid world execution.

Another important aspect in the simulation with two corrupted parties is the blocking of money. Lets consider the following situation. Buyer accepts the exchange offer from $\tilde{\mathcal{S}}^*$ by locking p coins in the smart contract. After this step, Seller has to reveal his key. Since $\tilde{\mathcal{S}}^*$ is malicious, he may abort and never reveal his key. $\tilde{\mathcal{B}}^*$ may send an *abort*-message to get his money back. However, since Buyer is also malicious, he may also abort and refrain from getting this money back. This sounds not reasonable, but it must be shown that the environment cannot distinguish the hybrid and the ideal world in this scenario.
In the hybrid world, the money becomes locked forever, since $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ is not triggered to unlock the coins. To simulate the same behavior in the ideal world, the simulator makes use of the blocking feature of $\mathcal{L}$. If no message is received in round 3, i.e., no *reveal*-message from $\tilde{\mathcal{S}}^*$ and no *abort*-message from $\tilde{\mathcal{B}}^*$, $Sim^{\mathcal{SB}}$ sends a $(abort, id)$-message in the name of $\tilde{\mathcal{S}}^*$ to $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$. This message triggers $\mathcal{F}_{\mathsf{icfe}}^{\mathcal{L}}$ to instruct $\mathcal{L}$ to unlock coins in favor of $\tilde{\mathcal{B}}^*$. However, by sending a $(block, id)$-message on behalf of the corrupted party $\tilde{\mathcal{B}}^*$ to $\mathcal{L}$, $Sim^{\mathcal{SB}}$ prevents the unlocking of the coins in favor of $\tilde{\mathcal{B}}^*$. Hence, the same behavior is achieved by the simulator. This procedure is executed each time both parties abort in the same round to achieve termination of the execution without unlocking the coins.

Next, it is important to take a look at the simulation of the dispute resolution subprotocol. In step (4a), the simulator awaits a message from the corrupted Buyer $\tilde{\mathcal{B}}$. This can be either a *challenge*-message, a *prove*-message, or a *complain*-message. In case of a *prove*- or a *complain*-message, the simulator has to judge on the validity of this message. Since Seller can also act maliciously, $Sim^{\mathcal{SB}}$ does not know in advance whether or not the received message is valid. Therefore, he evaluates the Judge- or the ValidateResponse-algorithm, respectively. This carries out the same operation as the hybrid functionality $\mathcal{G}_{\mathsf{jc}}^{\mathcal{L},\mathcal{H}}$ and hence the simulated behavior matches the execution in the hybrid world.

By showing that the environment cannot take advantage of this influence and based on the argumentation already stated in the proofs of the scenarios with single corruption, it concludes that the execution of the hybrid world and the execution of the ideal world are indistinguishable for any ppt environment $\mathcal{Z}$.

F Evaluation and Results

The performance of FairSwap [14] depends on the circuit used for witness verification. The authors claimed that circuits with a small instruction set and a small fan-in are promising candidates. The small instruction set imply that the judge smart contract does not need to be able to recompute many different instructions. Hence, the contract implementation

may be more efficient. We recall that the fan-in denotes the maximal number of inputs to a single circuit gate. Therefore, a small fan-in results in a small proof of misbehavior. Finally, Dziembowski et al. note that not only the number of instructions influences the efficiency but also the actual instructions. This observation is based on the fact that some instructions can be executed at low cost while other ones are very costly.

While all these parameters influence the efficiency of the judge implementation, they do not effect the message complexity of the first protocol message. This only depends on the size of the verification circuit. All in all, Dziembowski et al. considered the file sale application during their evaluation. Indeed, the FairSwap protocol is an efficient candidate for this use case.

For evaluating the efficiency of OptiSwap, we need to additionally consider the dispute resolution protocol. Its efficiency can be indicated by the number of rounds required in worst case. This value depends on the characteristics of the verification circuit like size, depth, and max fan-in.

In the following, we consider the file sale application to compare the efficiency of OptiSwap with FairSwap and a protocol built on the SmartJudge architecture [27]. We start with a brief description of the file sale application, outline a smart contract implementation for Ethereum, and present the evaluation results afterwards.

F.1 File Sale Application

For the file sale application, we consider objects that are identifiable via their Merkle hash, e.g., movies and software executables. Seller $\mathcal{S}$ offers a file $\mathbf{x} = (x_1, \ldots, x_n)$, where each element x_i is of size λ for $i \in [n]$. The Merkle hash of $\mathbf{x}$ is given by $h = \mathsf{MTHash}(\mathbf{x})$. The verification circuit ϕ computes the Merkle hash of the input and compares it with the expected value h, i.e., $\phi(\mathbf{x}) = 1 \Leftrightarrow \mathsf{MTHash}(\mathbf{x}) = h$.

In this scenario, the instruction set of ϕ consists of two instructions. A hash function $H : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^\mu$ that hashes two input elements and a function $eq : \{0,1\}^\mu \times \{0,1\}^\mu \to \{0,1\}$ to compare two values on equality. Figure 2 depicts the verification circuit for a file of $n = 8$ elements.

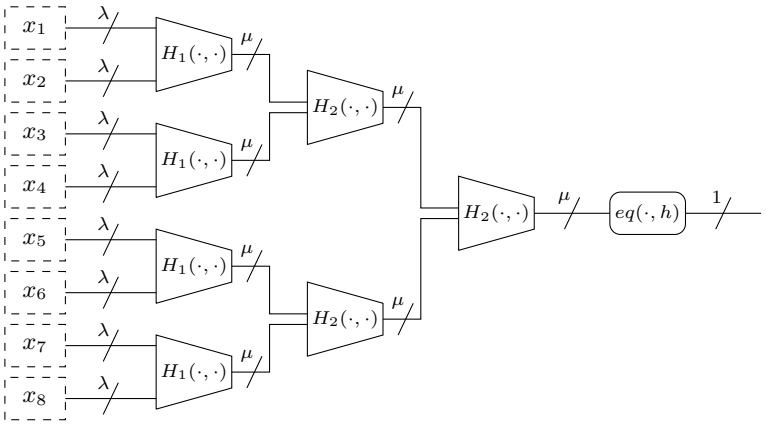

Fig. 2. Example of a circuit for the file sale application.

F.2 Implementation

Dziembowski et al. provided an implementation for the file sale application based on their FairSwap protocol on GubitHub[7]. We took their implementation and extend it by incorporating methods for the challenge-response procedure. This adaptation contains functions for challenging and responding as well as complaining about a passed timeout. Additionally, a function for validating the most recent response is added. The full smart contract code can be found on GitHub[8].

F.3 Results Comparison

In the following we compare our construction with FairSwap [14] and SmartJudge [27] regarding the message complexity and the gas costs.

Message Complexity. The major drawback of FairSwap is the overhead of the first protocol message. This overhead results from the information needed to create a valid proof of misbehavior. Although a proof of misbehavior needs to be created only in case of dispute, the data must be transferred anyway. The actual size of the message does not only depend on the size of the witness but also on the size of the verification circuit. The higher the circuit size, the bigger is the resulting message.
Dziembowski et al. stated that for the file sale application the message size may by at most twice as large as the file itself. Considering other circuits, the overhead might become even worse.
In contrast to FairSwap, our construction shrinks the message size to just the size of the witness itself. Hence, the overhead is completely removed. As a trade-off, the number of rounds can increase. But, contrary to FairSwap, this efficiency reduction happens only in case of dispute where at least one party behaves maliciously. The honest execution has the same round complexity and improved message complexity.

Gas Costs. Based on the smart contract implementation, we provide estimates about the gas costs of our protocol execution. We used the Remix Solidity IDE[9] to determine gas costs of the deployment and execution of the smart contract.
In order to enable comparability with FairSwap [14] and SmartJudge [27], we present the gas costs in two price models. The first one, called the average model $\mathcal{A}$, has a fixed ether price of 500 USD and a gas price of 3 GWei. These exchange rates are the same as in the evaluation of [14] and [27] and hence allow a comparison. The second model, called the current model $\mathcal{C}$, uses an ether price of 181.57 USD and a gas price of 14.4 GWei as of November 18, 2019 [1].

Optimistic execution. In case both parties behave honestly, the gas costs of the protocol execution include the deployment costs as well as the costs for exchanging the encryption key against the money. After at most 5 rounds, the honest execution terminates. Table 3 summarizes the gas costs in the optimistic execution. In this table it is assumed that the Buyer sends a *finalize*-message in order to finish the protocol execution. Table 4 compares the gas costs with FairSwap [14] and SmartJudge [27]. Since the smart contract code contains additional methods for the dispute resolution, it is reasonable that the deployment costs are higher for OptiSwap. In Section 3.5 we propose a solution to reduce the deployment cost for the optimistic execution. The costs for calling the smart contract functions are nearly

[7] The source code of the file sale application based on the FairSwap protocol can be found at https://github.com/lEthDev/FairSwap.
[8] The source code of the OptiSwap protocol can be found on https://github.com/CryBtoS/OptiSwap.
[9] Remix Solidity IDE: https://remix.ethereum.org

Table 3. Gas costs of all functions executed in the optimistic execution.

Function	Caller	Costs		
		Gas Costs	$\mathcal{A}$ [USD]	$\mathcal{C}$ [USD]
Deployment	Seller	2 273 398	3.41	5.94
`accept`	Buyer	32 394	0.05	0.08
`revealKey`	Seller	55 051	0.08	0.14
`noComplain`	Buyer	13 862	0.02	0.04

the same. This is due to the fact that the honest execution does not differ in both protocols. The small difference may result from slightly different costs for the function call dispatcher of the Ethereum Virtual Machine.

Table 4. Gas cost comparison between OptiSwap, FairSwap [14] and SmartJudge [27]. The gas costs for the deployment and the honest execution (without deployment) are stated. The values for FairSwap and SmartJudge are taken as claimed in the respective work.

Function	Gas Costs		
	Interactive Protocol	**FairSwap**	**SmartJudge**
Deployment	2 273 398	1 050 000	1 947 000
Honest Execution	101 307	103 333 [10]	143 000

Dispute resolution. In case at least one party behaves maliciously, the dispute resolution must be executed. This protocol phase includes many costly blockchain transactions. However, our dispute resolution is constructed in a way that an honest party is reimbursed at the end. This is based on the incorporated fee system. For the sake of completeness and as a guidance for the determination of fee parameters, we computed gas cost estimates for a dispute resolution.
Based on the file sale application with a file size of 1 GByte and a file chunk size of 512 Bytes, Table 5 contains the gas costs for the whole dispute resolution. This comprises the gas costs for all challenge queries, all response messages, and the complaint.

Table 5. Aggregated gas costs for the challenge-response procedure for a file sale based on OptiSwap.

Function	Caller	Costs		
		Gas Costs	$\mathcal{A}$ [USD]	$\mathcal{C}$ [USD]
`challenge`	Buyer	962 623	1.44	2.52
`response`	Seller	5 255 878	7.88	13.74
`complainAboutLeaf`	Buyer	194 068	0.29	0.51
	Total:	6 412 569	9.62	16.77

Fee Parameters. The fee mechanism is incorporated into the dispute resolution in order to reimburse honest parties. As shown in Table 5, participating in the challenge-response procedure is very costly.
We designed the fee mechanism in such a way that the reimbursement is ensured no matter

[10] The value is derived form the costs of 1.73 USD claimed in [14] for the honest protocol execution and an ether price of 500 USD and a gas price of 3 GWei. In addition the deployment costs are subtracted.

after how many rounds the challenge-response procedure ends. It may end either by a successful dispute resolution or by the abort of a malicious party.
Comparing the costs for creating a challenge query and generating a response message, we observed that the Buyer has to pay higher fees. We investigated the costs for each challenge query and each response transaction during the dispute resolution of the file sale application to find out how much gas is required for the most expensive challenge and response, respectively. It turned out that the first challenge query is the most expensive one for the Buyer with gas costs of 79 426. We further observed that the second response has the highest gas costs of 645 331 out of all response transactions.
Based on these two gas costs, we are able to determine static fee parameters. Considering the average price model $\mathcal{A}$, a Seller fee of 238 278 GWei per response and a Buyer fee of 1 935 993 GWei is an appropriate choice. These parameters guarantee a reimbursement no matter when the dispute resolution terminates.
Since the parameters represent an upper bound of gas costs paid during the dispute resolution, the overall amount of fees may be higher than the actual costs for the transactions. This may increase the financial burden for both parties but at the same time includes a deterrent effect for malicious parties.