Accountable Fair Exchange Protocols on Blockchain

Riku Mochizuki[1], Ryosuke Abe[1] and Shigeya Suzuki[1]
[1]*Graduate School of Media and Governance, Keio University*, Kanagawa, Japan
riku-m@keio.jp, chike@sfc.wide.ad.jp, shigeya@wide.ad.jp

***Abstract*—Fair exchange protocols on blockchain enable atomic exchange of digital goods and cryptocurrency between untrusted parties. Two prominent protocols, Zero-Knowledge Contingent Payment (ZKCP) and FairSwap, guarantee fairness: either both parties receive the expected items (digital goods and cryptocurrency) or neither does. However, both protocols lack accountability: when an exchange terminates abnormally, the protocol cannot identify which party caused the failure. This lack of accountability undermines the applicability of fair exchange protocols in decentralized settings. To fill this gap, we identify the common accountability issue in both protocols and revise them by adding signature exchange and chaining. We formally define the revised protocols in the universal composability framework and provide a proof sketch, and measure the computational overhead of the revised protocols in our implementation and experiment.**

***Index Terms*—Fair Exchange, Accountability, Blockchain**

I. Introduction

Fairly exchanging digital goods and cryptocurrency between two untrusted parties, such as a seller and buyer, in an atomic manner is essential for commercial transactions in the digital space. In such exchanges, we desire to ensure fairness, which is to say that either both parties receive the expected goods and cryptocurrency, or both parties receive nothing, but we face a problem known as the "Seller and Buyer Dilemma [1]" where either a seller might receive cryptocurrency but never deliver the goods, or a buyer might receive goods but fail to pay cryptocurrency in the digital space. The exchange with fairness is well-known as "Fair Exchange [2]." To mitigate the problem, prior fair exchange studies incorporate a Trusted Third Party (TTP) to mediate the exchange and showed that ensuring fairness is only possible with the TTP mediation[1]. It is well-known that the TTP mediation introduces threats: the TTP becoming a single point of failure and the possibility for the TTP to engage in dishonest behavior despite parties being honest [1], [3], [4].

Blockchain, a distributed ledger technology, eliminates the threats associated with specific TTP mediation [5]. There are two prominent fair exchange protocols on blockchain: Zero-Knowledge Contingent Payment (ZKCP) [6], [7] and FairSwap [8]. ZKCP is the first fair exchange protocol on blockchain that uses zero-knowledge proofs to enable an atomic exchange of digital goods and cryptocurrency [6], [7]. While ZKCP has a practical limitation of dealing with large data sizes, FairSwap enables such exchanges without using zero-knowledge proofs and incorporates "fraud detection," where blockchain verifies only when malicious behavior is detected, making it a groundbreaking fair exchange protocol optimized for efficiency [8], [9].

A malicious party can deceive an honest party into believing they intend to exchange, thereby forcing the honest party to bear the monetary cost of initiating the fair exchange protocol on blockchain. Fair exchange protocols incur monetary costs when running on blockchain. In public blockchains such as Bitcoin and Ethereum, each party incurs monetary costs to execute arbitrary transactions including fair exchange protocols. Thus, a malicious party can force the honest party to pay the monetary cost of exchange. Both protocols satisfy fairness because both parties receive nothing, but an honest party is still at a disadvantage regarding the monetary cost and lacks incentive mechanisms for deterring malicious behavior.

To address the problem, we introduce the requirement of "accountability" for fair exchange protocols. Prior studies define accountability in cryptographic/distributed protocol as the ability of a protocol to point out which parties have been misbehaving [10]–[14]. Accountability is essential in decentralized settings because it creates an incentive to deter malicious behavior by enabling punishment and reward mechanisms for malicious behavior through reputation systems [11]–[13]. Indeed, Abe et al. propose a reputation system for escrow services based on fair exchange protocols by recording the exchange history (e.g., the status of fair exchange termination) to assess the trustworthiness of the party on blockchain [15].

However, we found that existing prominent fair exchange protocols "ZKCP" and "FairSwap" lack accountability in abnormal termination, where both parties cannot obtain the expected goods and cryptocurrency because at least one party acts dishonestly. Since execution on the blockchain incurs monetary costs, a malicious party can exploit this ambiguity to force the honest party to bear costs while escaping blame, and the consequences extend beyond mere monetary burden in some cases. In the study by Abe et al., abnormal termination without distinguishing the dishonest party leads to both parties being considered untrustworthy, even if only one is truly dishonest [15]. This creates a disadvantage for the honest party, potentially undermining the effectiveness of the reputation system in decentralized settings.

To fill the gap, we identify the common accountability issue in both ZKCP and FairSwap and revise the protocols to provide accountability in abnormal termination. Our key idea is simple and straightforward to implement on existing protocols: we add *signature exchange and chaining* to ZKCP

This is the extended version of a paper accepted at IEEE COMPSAC 2026. This version includes proof sketches of the security theorems that were omitted from the conference version due to space constraints.

[1]This proof assumes a deterministic fair exchange protocol.

and FairSwap without increasing the number of messages on the blockchain. We revise ZKCP and FairSwap and provide a proof sketch of their security in the universal composability (UC) framework [16].

We organize this paper as follows. Section II introduces the preliminaries of this study. Section III and Section IV show the detailed description of ZKCP and FairSwap and identify the accountability issues in the protocols respectively. Section V formalizes the common issue and presents our revision idea. Section VI and Section VII show the revised ZKCP and FairSwap and provide the proof sketch of the security. Section VIII discusses the computational overhead of our revision. Section IX concludes the study.

II. Preliminaries

A. Blockchain and Smart Contract

Blockchain, a distributed ledger technology, facilitates transactions and operates in a peer-to-peer network, where each participating node maintains a replica of the ledger. The key feature of blockchain technology is a tamper-resistant datastore realized without relying on a specific administrator; it is based on cryptographic techniques and consensus algorithms.

The data structure of a blockchain comprises a sequence of blocks which includes a set of transactions, and the blockchain generates new blocks at regular intervals. This interval also serves as a relative time metric within the blockchain because all blocks are identified by a total order, often called the "block number." For example, the block interval of Ethereum [17], a well-known blockchain implementation, is approximately 12 seconds as of January 2026 [18]. The interval can be controlled by consensus algorithms, such as Proof-of-Work [19] and Proof-of-Stake [20].

"Smart contract" is an arbitrary procedure to compute, record, and transition between states on the blockchain, first implemented in Ethereum [17]. When a smart contract is deployed or invoked, it is executed on the Ethereum Virtual Machine (EVM) of all blockchain nodes through the consensus mechanism. Therefore, a smart contract is tamper-resistant and transparent because all blockchain nodes can validate the execution results through the consensus algorithm. Computation costs in the EVM are represented as "gas," which is also a monetary cost, as the gas is paid in cryptocurrency such as Ether (ETH) in Ethereum by the smart contract's executor to the blockchain nodes. In this study, we refer to a message sent to the blockchain as an *on-chain* message and a message sent directly between parties without using the blockchain as an *off-chain* message. We refer to a smart contract that implements a fair exchange protocol and mediates between the seller and buyer as a *judge contract*.

B. Fair Exchange Protocols

Fair exchange protocols are cryptographic protocols designed to enable an atomic exchange of digital items (e.g., goods, cryptocurrency, signatures, receipts, and so on) between two or more parties [8], [21]–[25]. Our study focuses on fair exchange protocols designed to facilitate the exchange of digital goods and cryptocurrency between a seller and buyer, which is the most common fair exchange scenario on blockchain [6]–[9]. The seller provides digital goods to the buyer, and the buyer pays the seller in cryptocurrency. In this study, we denote the expected goods as x such that $\phi(x) = 1$, where ϕ is a predicate function that evaluates the validity of the goods, and we assume ϕ is known and agreed upon by both parties. Prominent fair exchange protocols on blockchain are Zero-Knowledge Contingent Payment (ZKCP) [6], [7] and FairSwap [8].

To establish the necessary security properties for fair exchange, we build upon the notations and concepts from previous studies [2], [8], [9], [25], [26]. We define the following security properties that must be satisfied:

Seller Fairness. An honest seller can obtain the payment once the buyer obtains the expected goods. The seller cannot deny sending the goods (as non-repudiation of origin [25]).

Buyer Fairness. An honest buyer can obtain the expected goods once the seller has provided them. The buyer cannot deny receiving the goods (as non-repudiation of receipt [25]).

Exchange Timeliness. The execution of a fair exchange is expected to converge to one of the following states within a specific time: *Normal Termination*: Both parties obtain the expected goods and cryptocurrency. *Abnormal Termination*: Although two parties agree to initiate the fair exchange, one or both parties do not follow the procedures of the fair exchange protocol.

Confidentiality. The goods can only be revealed to the buyer if the exchange succeeds.

C. Cryptographic Primitives

We introduce several cryptographic primitives that are used in ZKCP and FairSwap and our revised protocols.

1) Symmetric Encryption: A symmetric encryption scheme uses a single shared key for both encryption and decryption. We define $\Pi_{SE} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$: $k \leftarrow \Pi_{SE}.\mathsf{KeyGen}(1^\lambda)$ generates a symmetric key k given a security parameter λ. $z \leftarrow \Pi_{SE}.\mathsf{Encrypt}(k, m)$ encrypts a plaintext m with key k to produce a ciphertext z. $m \leftarrow \Pi_{SE}.\mathsf{Decrypt}(k, z)$ decrypts a ciphertext z with key k to recover the plaintext m. We assume the scheme satisfies *IND-CPA* (Indistinguishability under Chosen Plaintext Attack) security in the presence of any PPT *(Probabilistic Polynomial-Time)* adversary.

2) Digital Signatures Based on Public-Key Cryptography: We introduce the digital signature scheme that is a core component of our revised protocols. We provide the definition of the digital signature scheme $\Pi_{SIG} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$: $(sk, pk) \leftarrow \Pi_{SIG}.\mathsf{KeyGen}(1^\lambda)$ denotes the generation of a signing key (private key) sk and a verification key (public key) pk given a security parameter λ. $\sigma \leftarrow \Pi_{SIG}.\mathsf{Sign}_{sk}(m)$ denotes the generation of a signature σ for a message m using a signing key (private key) sk based on public-key cryptography. $\Pi_{SIG}.\mathsf{Verify}_{pk}(\sigma, m)$ denotes the verification of the signature σ where pk is the verification key (public key).

If $\Pi_{SIG}.\mathsf{Verify}_{pk}(\sigma, m)$ is 1, then the signature σ is valid; otherwise the signature is invalid. We assume the scheme meets *EUF-CMA* (Existential Unforgeability under Chosen Message Attack) security in the presence of any PPT adversary.

3) Cryptographic Hash Function: A cryptographic hash function $H : \{0,1\}^* \to \{0,1\}^n$ maps an arbitrary-length input to a fixed-length output. It must satisfy: *Preimage Resistance*: Given h, it is computationally infeasible to find x such that $H(x) = h$. *Collision Resistance*: It is computationally infeasible to find $x \neq x'$ such that $H(x) = H(x')$. Practical instantiations include SHA-3 [27].

4) Commitment Scheme: A commitment scheme allows a party to commit to a message without revealing it until a later time. We define $\Pi_{COM} = (\mathsf{Commit}, \mathsf{Open})$: $(c, d) \leftarrow \Pi_{COM}.\mathsf{Commit}(m)$ generates a commitment c and an opening value d for a message m. $\Pi_{COM}.\mathsf{Open}(c, d, m)$ outputs 1 if the commitment is valid and 0 otherwise.

5) Merkle Tree Function: We introduce the Merkle tree function that is a core component of FairSwap. A Merkle tree is a binary tree in which data elements are hashed and structured hierarchically for efficient verification of data integrity. We provide the definition of the Merkle tree function $\Pi_{MT} = (\mathsf{Mtree}, \mathsf{Root})$: $M \leftarrow \Pi_{MT}.\mathsf{Mtree}(x_1, \cdots, x_n)$ denotes the construction of a Merkle tree where $x_1, \cdots, x_n$ are elements of the tree, and $\Pi_{MT}.\mathsf{Root}(M)$ returns the root hash of the Merkle tree M. We omit the details of Merkle tree functions as they are defined in the same way as in a prior study [8].

6) Zero-Knowledge Proofs: A zero-knowledge proof allows a prover to convince a verifier that a statement is true without revealing any information beyond its validity. Given a relation $R(stmt, w)$ that defines the validity condition between a public statement $stmt$ and a witness w, we define $\Pi_{ZKP} = (\mathsf{Prove}, \mathsf{Verify})$: $\pi \leftarrow \Pi_{ZKP}.\mathsf{Prove}(R, stmt, w)$ generates a proof π that the prover knows a witness w such that $R(stmt, w) = 1$. $\Pi_{ZKP}.\mathsf{Verify}(\pi, stmt)$ outputs 1 if the proof is valid and 0 otherwise. A zero-knowledge proof satisfies: *Completeness*: If $R(stmt, w) = 1$, an honest prover always convinces the verifier. *Soundness*: If no w satisfies $R(stmt, w) = 1$, no PPT adversary can convince the verifier except with negligible probability. *Zero-Knowledge*: The verifier learns nothing beyond the validity of $stmt$. A succinct non-interactive argument of knowledge (SNARK) [28] enables non-interactive and efficiently verifiable proofs; Practical instantiations include Groth16 [29].

III. Zero-Knowledge Contingent Payment (ZKCP)

We provide a brief description of ZKCP, a fair exchange protocol for the exchange of digital goods and cryptocurrency (which we refer to as *coins*) [6], [7]. ZKCP leverages zero-knowledge proofs to enable an atomic exchange of digital goods and coins. In ZKCP, the predicate $\phi_{\text{ZKCP}}(\cdot)$ is realized through a hash function H where the seller demonstrates that $H(x) = y$ for a publicly known output y.

A. The ZKCP Procedure

We explain the procedure of ZKCP. Detailed explanations and rigorous definitions can be found in the literature [6], [7].

Initialize Phase.

- The seller $\mathcal{S}$ generates a symmetric key k using $\Pi_{SE}.\mathsf{KeyGen}(1^\lambda)$ and computes $z \leftarrow \Pi_{SE}.\mathsf{Encrypt}(k, x)$ and $h \leftarrow H(k)$.
- $\mathcal{S}$ defines the relation $R_{\text{ZKCP}}(stmt, w) = 1$ iff (1) $z = \Pi_{SE}.\mathsf{Encrypt}(k, x)$, (2) $h = H(k)$, and (3) $\phi_{\text{ZKCP}}(x) = 1$, where $stmt = (z, h, \phi_{\text{ZKCP}})$ and $w = (k, x)$. $\mathcal{S}$ generates a zero-knowledge proof $\pi \leftarrow \Pi_{ZKP}.\mathsf{Prove}(R_{\text{ZKCP}}, stmt, w)$.
- $\mathcal{S}$ sends z, h, π, and p to the buyer $\mathcal{B}$ via the message $(\text{sell}, z, h, \pi, p)$.
- The buyer $\mathcal{B}$ verifies the proof π using $\Pi_{ZKP}.\mathsf{Verify}(\pi, stmt)$ where $stmt = (z, h, \phi_{\text{ZKCP}})$. If valid, the buyer $\mathcal{B}$ initiates an instance of fair exchange on the blockchain by sending the message $(\text{accept}, p, coin, h)$ to the judge contract, which registers h and locks an amount p of coins for the exchange.

Reveal Phase.

- The seller $\mathcal{S}$ discloses the key k to the judge contract via the message (reveal, k), and the judge contract verifies that $H(k) = h$.
- If $H(k) = h$, the contract releases the amount p of coins to the seller $\mathcal{S}$. The buyer $\mathcal{B}$ obtains k from the judge contract and decrypts the encrypted goods z to obtain x using k.
- If the seller $\mathcal{S}$ does not reveal k before the timeout, the contract refunds the amount p of coins to the buyer $\mathcal{B}$.

B. Accountability in ZKCP

We identify an accountability issue in ZKCP. After the buyer $\mathcal{B}$ sends the message $(\text{accept}, p, coin, h)$ to the judge contract, the seller $\mathcal{S}$ is expected to send the message (reveal, k). However, the seller $\mathcal{S}$ may not send the message (reveal, k) for two reasons:

- **Seller's Responsibility**: The buyer $\mathcal{B}$ verified $\Pi_{ZKP}.\mathsf{Verify}(\pi, stmt)$ and registered the correct h and confirmed p on the judge contract. The seller $\mathcal{S}$ has k such that $H(k) = h$ but refuses to send the message (reveal, k).
- **Buyer's Responsibility**: The buyer $\mathcal{B}$ registered a hash value h' and an amount p' on the judge contract via the message $(\text{accept}, p', coin, h')$ that is inconsistent with h and p sent in the off-chain message $(\text{sell}, z, h, \pi, p)$. The seller $\mathcal{S}$ cannot send a valid message (reveal, k) because $H(k) \neq h'$.

The judge contract receives only the message $(\text{accept}, p, coin, h)$ from the buyer $\mathcal{B}$ and does not receive the off-chain message $(\text{sell}, z, h, \pi, p)$ from the seller $\mathcal{S}$. Therefore, the judge contract cannot verify whether the h in the message $(\text{accept}, p, coin, h)$ matches the h in the message $(\text{sell}, z, h, \pi, p)$. When the seller $\mathcal{S}$ does not send the message (reveal, k), the judge contract cannot distinguish whether the

seller $\mathcal{S}$ refuses to reveal k for the correct h and p, or the buyer $\mathcal{B}$ registered inconsistent h and p.

IV. FairSwap

We provide a brief description of FairSwap. FairSwap leverages Merkle trees to enable fraud detection even for goods with large data sizes. In FairSwap, the predicate $\phi_{\text{FS}}(\cdot)$ is used directly to evaluate the validity of goods: $\phi_{\text{FS}}(x) = 1$ if and only if the goods x satisfy the agreed-upon condition. Both the buyer and the seller share ϕ_{FS} and the description of the goods to be exchanged, such as fingerprints (e.g., cryptographic hash values).

A. Functions defined in FairSwap

We provide a simplified explanation. Detailed explanations and rigorous definitions can be found in the original literature [8], [9].

- Encode(ϕ_{FS}, x, k): Encrypts the digital goods x and the predicate function $\phi_{\text{FS}}(\cdot)$ that evaluates the validity of the goods x. We let $z \leftarrow$ Encode(ϕ_{FS}, x, k). The output is 1 if the verification is successful and 0 otherwise.
- Extract(ϕ_{FS}, z, k): Decrypts the encrypted goods z to obtain the digital goods x and checks if they satisfy the predicate function $\phi_{\text{FS}}(\cdot)$. We let $(x, \pi) \leftarrow$ Extract(ϕ_{FS}, z, k). If the goods are valid, it outputs $(x, \perp)$; if the goods are invalid, it generates a proof of fraud π.
- Judge($k, r_z, r_{\phi_{\text{FS}}}, \pi$): Verifies the proof of fraud π when the buyer claims that the goods are fraudulent. We let Judge($k, r_z, r_{\phi_{\text{FS}}}, \pi$) $= 1$ or 0. The output is 1 if the proof is valid and 0 otherwise.

B. The FairSwap Procedure

We explain the procedure of FairSwap.

Initialize Phase.

- The seller $\mathcal{S}$ generates a symmetric key k and computes the commitment c using $\Pi_{COM}.\mathsf{Commit}(k)$. The output is $(c, d) \leftarrow \Pi_{COM}.\mathsf{Commit}(k)$.
- $\mathcal{S}$ encrypts the digital goods x with the key k and encodes them along with ϕ_{FS} using Encode(ϕ_{FS}, x, k) to create the encrypted data z. The output is $z \leftarrow$ Encode(ϕ_{FS}, x, k).
- $\mathcal{S}$ computes the Merkle roots $r_{\phi_{\text{FS}}} \leftarrow \Pi_{MT}.\mathsf{Root}(\Pi_{MT}.\mathsf{Mtree}(\phi_{\text{FS}}))$ and $r_z \leftarrow \Pi_{MT}.\mathsf{Root}(\Pi_{MT}.\mathsf{Mtree}(z))$. $\mathcal{S}$ sends z, ϕ_{FS}, the commitment c, and an amount p of coins to the buyer $\mathcal{B}$ via the message $(\text{sell}, z, \phi_{\text{FS}}, c, p)$. The seller then initiates the fair exchange instance on the judge contract by sending the message $(\text{init}, p, c, r_{\phi_{\text{FS}}}, r_z)$ to the judge contract, which registers the Merkle roots $r_{\phi_{\text{FS}}}$ and r_z.
- The buyer $\mathcal{B}$ computes $r'_{\phi_{\text{FS}}} \leftarrow \Pi_{MT}.\mathsf{Root}(\Pi_{MT}.\mathsf{Mtree}(\phi_{\text{FS}}))$ and $r'_z \leftarrow \Pi_{MT}.\mathsf{Root}(\Pi_{MT}.\mathsf{Mtree}(z))$ from the received ϕ_{FS} and z, and verifies $r'_{\phi_{\text{FS}}} = r_{\phi_{\text{FS}}}$ and $r'_z = r_z$. If consistent, the buyer locks the amount p of coins in the judge contract via the message $(\text{accept}, coin)$.

Reveal Phase.

- The seller $\mathcal{S}$ discloses the key k to the judge contract via the message (reveal, d, k), and the judge contract verifies the commitment using $\Pi_{COM}.\mathsf{Open}(c, d, k)$. An output of 1 indicates successful verification.
- The buyer $\mathcal{B}$ decrypts z using the disclosed key k and checks if the decrypted goods x satisfy the predicate $\phi_{\text{FS}}(\cdot)$ using Extract(ϕ_{FS}, z, k). The output is $(x, \pi) \leftarrow$ Extract(ϕ_{FS}, z, k).

Payout Phase.

- If x is correct ($\pi = \perp$), the buyer $\mathcal{B}$ sends the message (complete) to the judge contract, and the contract releases the amount p of coins to the seller $\mathcal{S}$.
- If x is incorrect ($\pi \neq \perp$), the buyer $\mathcal{B}$ generates a proof of fraud π and submits it to the judge contract via the message $(\text{complain}, \pi)$.
- The judge contract verifies the proof of fraud using Judge($k, r_z, r_{\phi_{\text{FS}}}, \pi$). The output is 1 if the proof is valid.
- If the proof is valid, the contract refunds the amount p of coins to the buyer $\mathcal{B}$; if invalid, the amount p of coins is released to the seller $\mathcal{S}$.
- If the buyer $\mathcal{B}$ does not send the message (complete) or $(\text{complain}, \pi)$ before the timeout, the contract releases the amount p of coins to the seller $\mathcal{S}$.

C. Accountability in FairSwap

We identify an accountability issue in FairSwap. After the seller $\mathcal{S}$ sends the message $(\text{init}, p, c, r_{\phi_{\text{FS}}}, r_z)$ to the judge contract, the buyer $\mathcal{B}$ is expected to send the message $(\text{accept}, coin)$. However, the buyer $\mathcal{B}$ may not send the message $(\text{accept}, coin)$ for two reasons:

- **Seller's Responsibility**: The seller $\mathcal{S}$ registered Merkle roots r_z and $r_{\phi_{\text{FS}}}$, commitment c, and amount p on the judge contract via the message $(\text{init}, p, c, r_{\phi_{\text{FS}}}, r_z)$ that are inconsistent with z, ϕ_{FS}, c, and p sent in the off-chain message $(\text{sell}, z, \phi_{\text{FS}}, c, p)$. The buyer $\mathcal{B}$ correctly refuses to send the message $(\text{accept}, coin)$.
- **Buyer's Responsibility**: The Merkle roots r_z and $r_{\phi_{\text{FS}}}$ are consistent with z and ϕ_{FS}, but the buyer $\mathcal{B}$ refuses to send the message $(\text{accept}, coin)$.

The judge contract receives only the message $(\text{init}, p, c, r_{\phi_{\text{FS}}}, r_z)$ from the seller $\mathcal{S}$ and does not receive the off-chain message $(\text{sell}, z, \phi_{\text{FS}}, c, p)$. Therefore, the judge contract cannot verify whether r_z, $r_{\phi_{\text{FS}}}$, c, and p in the message $(\text{init}, p, c, r_{\phi_{\text{FS}}}, r_z)$ are consistent with z, ϕ_{FS}, c, and p in the message $(\text{sell}, z, \phi_{\text{FS}}, c, p)$. When the buyer $\mathcal{B}$ does not send the message $(\text{accept}, coin)$, the judge contract cannot distinguish whether the Merkle roots and/or the amount p are inconsistent with the agreed values, or the buyer $\mathcal{B}$ simply refuses to send the message.

V. Issues and Revision Idea

A. Accountability in Fair Exchange

We define accountability in fair exchange protocols as follows: when a fair exchange terminates abnormally, the judge

contract can determine which party is responsible for the abnormal termination, following prior seminal studies [10]–[14]. Since the judge contract's computation is publicly verifiable on the blockchain, any third party can independently confirm the responsible party. Accountability must satisfy two properties: (i) *blame soundness*, where an honest party is never blamed, and (ii) *blame completeness*, where a dishonest party is always blamed when responsible for abnormal termination.

As identified in Sections III and IV, both ZKCP and FairSwap lack this accountability property: the judge contract cannot observe the parameter verification between the seller and buyer, so the judge contract cannot unambiguously identify the responsible party when an abnormal termination occurs. When a party does not send an expected message to the judge contract, two cases are possible:

M-1: The parameter verification between the parties failed, therefore, the party does not send the message.

M-2: The parameter verification between the parties succeeded, but the party chose not to send the message.

B. Our Signature Exchange and Chaining

Each fair exchange protocol requires one party to verify protocol-specific parameters via off-chain messages before the exchange on the judge contract begins. In ZKCP, the buyer receives h in message (sell, z, h, π) and verifies h by checking the proof π, then registers h on the judge contract with the message $(\text{accept}, coin, h)$. In FairSwap, the buyer verifies that the Merkle roots r_z and $r_{\phi_{\text{FS}}}$ registered by the seller in message $(\text{init}, p, c, r_{\phi_{\text{FS}}}, r_z)$ are consistent with z and ϕ_{FS} received in message $(\text{sell}, z, \phi_{\text{FS}}, c)$, then sends the message $(\text{accept}, coin)$. We denote these protocol-specific parameters as *params*. The judge contract cannot independently verify whether *params* are correct because it does not receive the data sent via off-chain messages needed for the verification. This is the root cause of the M-1/M-2 ambiguity.

We introduce signature exchange and chaining to provide the judge contract with cryptographic evidence that both parties verified and accepted *params* before the exchange on the judge contract begins. With this evidence, the judge contract can distinguish M-1 from M-2. We assume that each party generates a key pair $(sk, pk) \leftarrow \Pi_{SIG}.\mathsf{KeyGen}(1^\lambda)$.

Our signature exchange and chaining proceeds as follows. One party computes $\sigma \leftarrow \Pi_{SIG}.\mathsf{Sign}_{sk}(id \| \textit{params})$, where id is a unique fair exchange instance identifier that binds signatures to a single instance and prevents replay attacks across instances, and sends *params* and σ to the other party. The other party verifies *params* and checks $\Pi_{SIG}.\mathsf{Verify}_{pk}(\sigma, id \| \textit{params}) \stackrel{?}{=} 1$. If both verifications succeed, the other party computes $\sigma' \leftarrow \Pi_{SIG}.\mathsf{Sign}_{sk}(id \| \sigma)$ and sends the message $(\text{approve}, \sigma')$. We call this procedure *exchange* because the two parties mutually send their signatures σ and σ', and *chaining* because σ' signs σ itself, cryptographically linking the two signatures.

The judge contract uses the pair (σ, σ') to resolve the M-1/M-2 ambiguity. The party that sends the initialization message to the judge contract includes σ and σ', and the judge contract verifies both signatures. The signature σ certifies that one party committed to *params*, and the counter-signature σ' certifies that the other party verified *params* and agreed to proceed with the fair exchange using these *params*. Therefore, a valid pair (σ, σ') on the judge contract proves that both parties agreed on *params*. If a party does not send the expected message after initiation on the judge contract, the judge contract determines that the cause is M-2: the party agreed to *params* but chose not to proceed.

C. Accountability for Protocol Deviations

We clarify the scope of accountability when parties deviate from the signature exchange and chaining procedure.

Signature exchange and chaining may fail. For example, a party may not send the message $(\text{approve}, \sigma')$, or may send an invalid σ' such that $\Pi_{SIG}.\mathsf{Verify}_{pk}(\sigma', id \| \sigma) \neq 1$. In either case, the party that sends the initialization message to the judge contract cannot include a valid pair (σ, σ'), and no fair exchange instance is created on the judge contract. This study addresses accountability only for abnormal terminations that occur after an instance is created on the judge contract, i.e., after both parties complete the off-chain signature exchange and the judge contract accepts the initialization.

A party may compute $\sigma' \leftarrow \Pi_{SIG}.\mathsf{Sign}_{sk}(id \| \sigma)$ without actually verifying *params* but the judge contract cannot check the actual verification result. Even in this case, the pair (σ, σ') proves that the parameter verification succeeded, and the judge contract treats the abnormal termination as M-2. If the party signs σ' without verifying *params* and the exchange terminates abnormally due to inconsistent parameters, the judge contract outputs (blame, id, P), where P is the party that signed σ'. This is because the valid pair (σ, σ') classifies the abnormal termination as M-2. Since signing σ' without verifying *params* is a deviation from the protocol, P is a dishonest party; blaming P is therefore consistent with blame completeness. The judge contract also checks $\Pi_{SIG}.\mathsf{Verify}_{pk}(\sigma, id \| \textit{params}) \stackrel{?}{=} 1$ against the submitted parameters in the initialization message. If the submitted parameters do not match σ, the judge contract rejects the initialization, so no instance is created on the judge contract and no accountability issue arises.

VI. Revision of ZKCP

We apply signature exchange and chaining as described in Section V to revise ZKCP.

A. Protocol Modifications

In ZKCP, the accountability issue arises when the seller $\mathcal{S}$ does not send the message (reveal, k) after the buyer $\mathcal{B}$ locks an amount p of coins. In the original protocol, the buyer $\mathcal{B}$ directly sends the message $(\text{accept}, p, coin, h)$ to register h on the judge contract and lock an amount p of coins, but the judge contract has no evidence that the registered h matches the seller $\mathcal{S}$'s parameters sent via the off-chain message. Hence, the judge contract cannot determine whether the seller $\mathcal{S}$'s failure to reveal is due to the seller $\mathcal{S}$'s malice or the buyer $\mathcal{B}$'s registration of an inconsistent h. We modify ZKCP so

that signature exchange and chaining provide cryptographic evidence that the buyer $\mathcal{B}$ properly verified h before the exchange on the judge contract begins.

Specifically, we modify the protocol as follows. The seller $\mathcal{S}$ prepares $\sigma \leftarrow \Pi_{SIG}.\mathsf{Sign}_{sk_S}(id\|h\|p)$ and sends the message $(\text{sell}, z, h, \pi, p, \sigma)$ to the buyer $\mathcal{B}$. Upon receiving the message, the buyer $\mathcal{B}$ performs the following verifications:

- Verifies the zero-knowledge proof π using $\Pi_{ZKP}.\mathsf{Verify}(\pi, stmt)$ where $stmt = (z, h, \phi_{\text{ZKCP}})$.
- Verifies the digital signature σ using $\Pi_{SIG}.\mathsf{Verify}_{pk_S}(\sigma, id\|h\|p)$.

If all verifications succeed, the buyer $\mathcal{B}$ generates $\sigma' \leftarrow \Pi_{SIG}.\mathsf{Sign}_{sk_B}(id\|\sigma)$ and replies with the message $(\text{approve}, \sigma')$. The buyer $\mathcal{B}$ then sends the message $(\text{accept}, p, coin, h, \sigma, \sigma', pk_S, pk_B)$ to the judge contract. The contract verifies both σ and σ', and checks that the h and p in σ match the submitted h and p. If all verifications pass, the contract registers h and locks an amount p of coins. The Reveal phase is identical to the original ZKCP.

By this modification, the judge contract has evidence that both parties agreed on h and p. Therefore, if the seller $\mathcal{S}$ does not send the message (reveal, k) within the timeout, the judge contract can determine that the cause is M-2, and the seller $\mathcal{S}$ is responsible for the abnormal termination. The only message added by our revision is the off-chain message $(\text{approve}, \sigma')$; the on-chain message (accept) includes (σ, σ') but no new on-chain message type is introduced.

B. Proof Sketch

Due to space constraints, we provide a proof sketch that the revised ZKCP satisfies the security properties within the universal composability (UC) framework [16].

1) Framework and Assumptions: We work in the generalized universal composability (GUC) framework [30] with a global ledger functionality $\mathcal{L}$ and a random oracle $\mathcal{H}$ modeling the hash function H used in ZKCP. We omit the details of the ideal functionalities $\mathcal{L}$ and $\mathcal{H}$ and give a brief description. The global ledger $\mathcal{L}$ models a cryptocurrency: it maintains balances for each party and supports $(\text{freeze}, \mathcal{P}, p)$ and $(\text{unfreeze}, p, \mathcal{P}')$ operations, where $(\text{freeze}, \dots)$ locks an amount p of coins from a party $\mathcal{P}$'s balance for a contract and $(\text{unfreeze}, \dots)$ unlocks the locked coins to party $\mathcal{P}'$. The global random oracle $\mathcal{H}$ models an idealized hash function that returns uniformly random values for each distinct query; we follow the formalism of a restricted programmable and observable random oracle [8].

We also assume a synchronous communication model where the protocol proceeds in discrete rounds and all participants are aware of the current round, and the communication channel between $\mathcal{S}$ and $\mathcal{B}$ is confidential, as in FairSwap [8]. This assumption is used instead of the modeling of an ideal functionality for global clocks modeled for block number for simplicity. A message sent in round r is delivered at the beginning of round $r + 1$. In the synchronous model, if a corrupted party does not send its expected message by the deadline of a given round, this is treated as the corrupted party having aborted; Sim models this by invoking (abort, id) on $\mathcal{F}$ on behalf of that party.

We consider a PPT adversary $\mathcal{A}$ that can corrupt either $\mathcal{S}$ or $\mathcal{B}$ (static corruption); the global ledger $\mathcal{L}$ and random oracle $\mathcal{H}$ are not corrupted. We write $\mathcal{S}^*$ or $\mathcal{B}^*$ to denote a corrupted party controlled by $\mathcal{A}$ in the real world. In the ideal world, honest parties are modeled as dummy parties that forward inputs from the environment $\mathcal{Z}$ to $\mathcal{F}$ [8]. The simulator Sim must produce a view for $\mathcal{Z}$ indistinguishable from the real execution.

The revised ZKCP preserves the cryptographic core of the original protocol unchanged and does not introduce additional on-chain messages or communication rounds; therefore, the existing security properties (Seller Fairness, Buyer Fairness, Exchange Timeliness, and Confidentiality) are maintained. Our proof sketch verifies blame soundness and blame completeness (defined in Section V) for the accountability property introduced by the added signature exchange and chaining.

2) Ideal Functionalities and Security: We define the ideal functionality $\mathcal{F}_{\text{zkcp}}^{\mathcal{L}}$ for fair exchange with zero-knowledge proofs in Figure 1. $\mathcal{F}_{\text{zkcp}}^{\mathcal{L}}$ models the ZKCP exchange and outputs (blame, id, P) identifying the responsible party P upon abnormal termination, enabling accountability verification within the UC framework. We also define the judge contract functionality $\mathcal{G}_{\text{zkcp}}^{\mathcal{L},\mathcal{H}}$ in Figure 2. $\mathcal{G}_{\text{zkcp}}^{\mathcal{L},\mathcal{H}}$ verifies both signatures σ and σ' in the Initialize phase and outputs (blame, id, P) upon abnormal termination based on cryptographic evidence. The GUC-security proof ensures that the blame output in the ideal world (from $\mathcal{F}_{\text{zkcp}}^{\mathcal{L}}$) is indistinguishable from the blame output in the real world (from $\mathcal{G}_{\text{zkcp}}^{\mathcal{L},\mathcal{H}}$), thereby formally verifying accountability. We explicitly specify round numbers in the definition of the ideal functionalities $\mathcal{F}_{\text{zkcp}}^{\mathcal{L}}$ and these correspond to rounds in the contract functionality $\mathcal{G}_{\text{zkcp}}^{\mathcal{L},\mathcal{H}}$.

Theorem 1 (Revised ZKCP)**.** *The revised ZKCP protocol* Π_{ZKCP}*, running in the* $(\mathcal{G}_{\text{zkcp}}^{\mathcal{L},\mathcal{H}}, \mathcal{L}, \mathcal{H})$*-hybrid world, GUC-realizes* $\mathcal{F}_{\text{zkcp}}^{\mathcal{L}}$*.*

To prove Theorem 1, we construct a simulator Sim such that $\mathcal{Z}$ cannot distinguish between the real execution of Π_{ZKCP} and the ideal execution of $\mathcal{F}_{\text{zkcp}}^{\mathcal{L}}$ with Sim. In our sketch, we show blame soundness and blame completeness only if either the seller $\mathcal{S}$ or the buyer $\mathcal{B}$ is corrupted.

Lemma 1 (ZKCP: Corrupted Seller)**.** *If the seller* $\mathcal{S}$ *is corrupted and the buyer* $\mathcal{B}$ *is honest, every abnormal termination after initiation on the judge contract* $\mathcal{G}_{\text{zkcp}}^{\mathcal{L},\mathcal{H}}$ *outputs* $(\text{blame}, id, \mathcal{S})$*, and* Sim *produces the corresponding* $(\text{blame}, id, \mathcal{S})$ *output in* $\mathcal{F}_{\text{zkcp}}^{\mathcal{L}}$*.*

Proof Sketch. We show blame soundness and blame completeness. In the real world, the honest buyer $\mathcal{B}$ receives $(\text{sell}, z, h, \pi, p, \sigma)$ from $\mathcal{S}^*$, verifies π via $\Pi_{ZKP}.\mathsf{Verify}$ and σ via $\Pi_{SIG}.\mathsf{Verify}_{pk_S}$, produces $\sigma' \leftarrow \Pi_{SIG}.\mathsf{Sign}_{sk_B}(id\|\sigma)$, and sends $(\text{accept}, id, p, h, \sigma, \sigma', pk_S, pk_B)$ to $\mathcal{G}_{\text{zkcp}}^{\mathcal{L},\mathcal{H}}$. In the ideal world, Sim receives the leakage from $\mathcal{F}_{\text{zkcp}}^{\mathcal{L}}$ and performs the same verification and signing steps in the protocol

The ideal functionality $\mathcal{F}_{\text{zkcp}}^{\mathcal{L}}$ (in session id) interacts with a buyer $\mathcal{B}$, a seller $\mathcal{S}$, the ideal adversary Sim, and the global ledger $\mathcal{L}$.

Initialize

(Round 1) Upon receiving (sell, id, x, p) from $\mathcal{S}$ with $p \in \mathbb{N}$, leak $(\text{sell}, id, x, p, \mathcal{S})$ to Sim and, store x, and p.
(Round 2) Upon receiving (buy, id, p) from $\mathcal{B}$, leak $(\text{buy}, id, p, \mathcal{B})$ to Sim, and send $(\text{freeze}, \mathcal{B}, p)$ to $\mathcal{L}$. If $\mathcal{L}$ responds with $(\text{frozen}, \mathcal{B}, p)$, go to Reveal. Upon (abort, id) from corrupted $\mathcal{B}^*$: send $(\text{unfreeze}, p, \mathcal{B})$ to $\mathcal{L}$, output $(\text{blame}, id, \mathcal{B})$ to all, terminate.

Reveal

(Round 3) Upon receiving (abort, id) from corrupted $\mathcal{S}^*$, send $(\text{unfreeze}, p, \mathcal{B})$ to $\mathcal{L}$, output $(\text{blame}, id, \mathcal{S})$ to all, and terminate. Otherwise, send (bought, id, x) to $\mathcal{B}$, $(\text{unfreeze}, p, \mathcal{S})$ to $\mathcal{L}$, and (sold, id) to $\mathcal{S}$.

Fig. 1: Ideal functionality $\mathcal{F}_{\text{zkcp}}^{\mathcal{L}}$ for Revised ZKCP.

$\mathcal{G}_{\text{zkcp}}^{\mathcal{L},\mathcal{H}}$ acts as a judge contract for session id, interacting with $\mathcal{L}$ and parties $\mathcal{S}$, $\mathcal{B}$. Stores h, p, state s.

Initialize Phase

(Round 2) Upon receiving $(\text{accept}, id, p, h, \sigma, \sigma', pk_S, pk_B)$ from $\mathcal{B}$ when $\Pi_{SIG}.\mathsf{Verify}_{pk_S}(\sigma, id\|h\|p) \stackrel{?}{=} 1$ and $\Pi_{SIG}.\mathsf{Verify}_{pk_B}(\sigma', id\|\sigma) \stackrel{?}{=} 1$: store h and p, send $(\text{freeze}, \mathcal{B}, p)$ to $\mathcal{L}$. If frozen, set $s = \textit{active}$ and output $(\text{accepted}, id)$.

Reveal Phase

(Round 3) Upon (reveal, id, k) from $\mathcal{S}$ when $s = \textit{active}$ and $\mathcal{H}(k) = h$: output $(\text{revealed}, id, k)$ to all, and send $(\text{unfreeze}, p, \mathcal{S})$ to $\mathcal{L}$.
Upon timeout when $s = \textit{active}$: $(\text{unfreeze}, p, \mathcal{B})$ to $\mathcal{L}$, output $(\text{blame}, id, \mathcal{S})$, $(\text{refunded}, id)$.

Fig. 2: Ideal functionality $\mathcal{G}_{\text{zkcp}}^{\mathcal{L},\mathcal{H}}$ for judge contract of Revised ZKCP.

transcript for $\mathcal{Z}$. If $\mathcal{S}^*$'s proof π or signature σ is invalid, Sim halts the simulation before initiation; no blame output is produced in either world.

In the real world, $\mathcal{G}_{\text{zkcp}}^{\mathcal{L},\mathcal{H}}$ verifies both σ and σ' and registers h. By EUF-CMA security, σ binds $\mathcal{S}^*$ to $id\|h\|p$; thus $\mathcal{S}^*$ cannot deny having committed to h and p, and cannot shift blame onto $\mathcal{B}$. If the verification passes, (σ, σ') certifies that $\mathcal{S}^*$ committed to h and p, and $\mathcal{S}^*$ cannot later deny this acceptance, establishing blame soundness.

If $\mathcal{S}^*$ does not reveal k before the timeout: in the real world, $\mathcal{G}_{\text{zkcp}}^{\mathcal{L},\mathcal{H}}$ outputs $(\text{blame}, id, \mathcal{S})$ and refunds $\mathcal{B}$; in the ideal world, $\mathcal{F}_{\text{zkcp}}^{\mathcal{L}}$ outputs $(\text{blame}, id, \mathcal{S})$ upon the corrupted $\mathcal{S}^*$'s failure to reveal. This establishes blame completeness. The blame outputs and refunds are identical in both worlds; hence $\mathcal{Z}$ cannot distinguish. □

Lemma 2 (ZKCP: Corrupted Buyer)**.** *If the buyer $\mathcal{B}$ is corrupted and the seller $\mathcal{S}$ is honest, every abnormal termination after initiation on the judge contract $\mathcal{G}_{\text{zkcp}}^{\mathcal{L},\mathcal{H}}$ outputs $(\text{blame}, id, \mathcal{B})$, and Sim produces the corresponding $(\text{blame}, id, \mathcal{B})$ output in $\mathcal{F}_{\text{zkcp}}^{\mathcal{L}}$.*

Proof Sketch. We show blame soundness; blame completeness holds vacuously as shown below. In the real world, the honest seller $\mathcal{S}$ computes $\sigma \leftarrow \Pi_{SIG}.\mathsf{Sign}_{sk_S}(id\|h\|p)$ and sends $(\text{sell}, z, h, \pi, p, \sigma)$ to $\mathcal{B}^*$. In the ideal world, Sim receives the leakage from $\mathcal{F}_{\text{zkcp}}^{\mathcal{L}}$ and produces the corresponding $(\text{sell}, z, h, \pi, p, \sigma)$ message in the transcript for $\mathcal{Z}$. If $\mathcal{B}^*$ does not respond with a valid σ', Sim halts the simulation before initiation; no blame output is produced in either world.

In the real world, $\mathcal{B}^*$ submits $(\text{accept}, id, p, h, \sigma, \sigma', pk_S, pk_B)$ to $\mathcal{G}_{\text{zkcp}}^{\mathcal{L},\mathcal{H}}$, which verifies $\Pi_{SIG}.\mathsf{Verify}_{pk_S}(\sigma, id\|h\|p) \stackrel{?}{=} 1$ and $\Pi_{SIG}.\mathsf{Verify}_{pk_B}(\sigma', id\|\sigma) \stackrel{?}{=} 1$. By EUF-CMA security, $\mathcal{B}^*$ cannot forge σ without sk_S. If $\mathcal{B}^*$ tampers with any of σ, σ', pk_S, or pk_B, or submits $h' \neq h$ or $p' \neq p$, the verification fails and the contract rejects; no instance is created and no ambiguity arises.

If $\mathcal{B}^*$ submits the correct h and p with the authentic $(\sigma, \sigma', pk_S, pk_B)$, the contract accepts: in the real world, since $\mathcal{S}$ is honest, $\mathcal{S}$ sends (reveal, k) with $H(k) = h$ and the exchange completes normally; in the ideal world, Sim simulates this reveal step and $\mathcal{F}_{\text{zkcp}}^{\mathcal{L}}$ completes normally. No (blame, id, P) output is produced in either world, and $\mathcal{Z}$ cannot distinguish. Thus, no abnormal termination after initiation occurs when $\mathcal{S}$ is honest. Blame completeness holds vacuously. □

VII. Revision of FairSwap

We apply signature exchange and chaining as described in Section V to revise FairSwap.

A. Protocol Modifications

In FairSwap, the accountability issue arises when the buyer $\mathcal{B}$ does not send the message $(\text{accept}, coin)$ after the seller $\mathcal{S}$ initiates the instance on the judge contract. Without signature exchange and chaining, the judge contract cannot distinguish whether the cause is M-1 or M-2. We modify FairSwap so that signature exchange and chaining provide evidence that the parameter verification succeeded, allowing the judge contract to determine that the cause is M-2.

Specifically, we modify the protocol as follows. The seller $\mathcal{S}$ prepares $\sigma \leftarrow \Pi_{SIG}.\mathsf{Sign}_{sk_S}(id\|r_{\phi_{\text{FS}}}\|r_z\|c\|p)$ and sends the message $(\text{sell}, z, \phi_{\text{FS}}, c, p, \sigma)$ to the buyer $\mathcal{B}$. Upon receiving the message, the buyer $\mathcal{B}$ performs the following verifications:

- Verifies the digital signature σ using $\Pi_{SIG}.\mathsf{Verify}_{pk_S}(\sigma, id\|r_{\phi_{\text{FS}}}\|r_z\|c\|p)$.
- Generates $r'_{\phi_{\text{FS}}}$ from ϕ_{FS} and r'_z from z and verifies that they match $r_{\phi_{\text{FS}}}$ and r_z committed in σ.

If all verifications succeed, the buyer $\mathcal{B}$ generates $\sigma' \leftarrow \Pi_{SIG}.\mathsf{Sign}_{sk_B}(id\|\sigma)$ and replies with the message $(\text{approve}, \sigma')$.

The seller $\mathcal{S}$ then sends the message $(\text{init}, p, c, r_{\phi_{\text{FS}}}, r_z, \sigma, \sigma', pk_S, pk_B)$ to the judge contract, which verifies both σ and σ'. If all verifications pass, the contract records the parameters and proceeds. The subsequent phases (Reveal, Payout) are identical to the original FairSwap.

By this modification, the judge contract has evidence that the parameter verification succeeded. Therefore, if the message $(\text{accept}, coin)$ is not sent within a specific period, the judge contract can determine that the cause is M-2, and the buyer $\mathcal{B}$ is responsible for the abnormal termination.

B. Proof Sketch

The GUC framework and assumptions are the same as in Section VI-B. The revised FairSwap also preserves the cryptographic core of the original protocol unchanged and does not introduce additional on-chain messages or communication rounds; therefore, the existing security properties are maintained, and our proof sketch verifies blame soundness and blame completeness for the accountability property.

We define the ideal functionality $\mathcal{F}_{\text{FS}}^{\mathcal{L}}$ for fair exchange with fraud detection in Figure 3. $\mathcal{F}_{\text{FS}}^{\mathcal{L}}$ models the FairSwap exchange and outputs (blame, id, P) identifying the responsible party P upon abnormal termination. We also define the judge contract functionality $\mathcal{G}_{\text{FS}}^{\mathcal{L},\mathcal{H}}$ in Figure 4. $\mathcal{G}_{\text{FS}}^{\mathcal{L},\mathcal{H}}$ verifies both signatures σ and σ' in the Initialize phase and outputs (blame, id, P) upon abnormal termination based on cryptographic evidence. The GUC-security proof ensures that the blame outputs are indistinguishable between the ideal and real worlds. We explicitly specify round numbers in the definition of the ideal functionalities $\mathcal{F}_{\text{FS}}^{\mathcal{L}}$ and $\mathcal{G}_{\text{FS}}^{\mathcal{L},\mathcal{H}}$, and these correspond to rounds in the contract functionality $\mathcal{G}_{\text{FS}}^{\mathcal{L},\mathcal{H}}$.

Theorem 2 (Revised FairSwap)**.** *The revised FairSwap protocol Π_{FS}, running in the $(\mathcal{G}_{\text{FS}}^{\mathcal{L},\mathcal{H}}, \mathcal{L}, \mathcal{H})$-hybrid world, GUC-realizes $\mathcal{F}_{\text{FS}}^{\mathcal{L}}$.*

To prove Theorem 2, we construct a simulator Sim such that $\mathcal{Z}$ cannot distinguish between the real execution of Π_{FS} and the ideal execution of $\mathcal{F}_{\text{FS}}^{\mathcal{L}}$ with Sim. In our sketch, we show blame soundness and blame completeness only if either the seller $\mathcal{S}$ or the buyer $\mathcal{B}$ is corrupted.

Lemma 3 (FairSwap: Corrupted Seller)**.** *If the seller $\mathcal{S}$ is corrupted and the buyer $\mathcal{B}$ is honest, every abnormal termination after initiation on the judge contract $\mathcal{G}_{\text{FS}}^{\mathcal{L},\mathcal{H}}$ outputs $(\text{blame}, id, \mathcal{S})$, and Sim produces the corresponding $(\text{blame}, id, \mathcal{S})$ output in $\mathcal{F}_{\text{FS}}^{\mathcal{L}}$.*

The ideal functionality $\mathcal{F}_{\text{FS}}^{\mathcal{L}}$ (in session id) interacts with a buyer $\mathcal{B}$, a seller $\mathcal{S}$, the ideal adversary Sim, and the global ledger $\mathcal{L}$.

Initialize

(Round 1) Upon receiving $(\text{sell}, id, \phi_{\text{FS}}, p, x)$ with $p \in \mathbb{N}$ from $\mathcal{S}$, leak $(\text{sell}, id, \phi_{\text{FS}}, p, \mathcal{S})$ to Sim, store goods x, predicate ϕ_{FS}, and price p.
(Round 2) Upon receiving $(\text{buy}, id, \phi_{\text{FS}}, p)$ from $\mathcal{B}$, leak $(\text{buy}, id, \phi_{\text{FS}}, \mathcal{B})$ to Sim, and send $(\text{freeze}, \mathcal{B}, p)$ to $\mathcal{L}$. If $\mathcal{L}$ responds with $(\text{frozen}, \mathcal{B}, p)$, go to Reveal. Upon (abort, id) from corrupted $\mathcal{B}^*$: $(\text{unfreeze}, p, \mathcal{B})$ to $\mathcal{L}$, output $(\text{blame}, id, \mathcal{B})$ to all, terminate.

Reveal

(Round 3) Upon receiving (abort, id) from corrupted $\mathcal{S}^*$, send $(\text{unfreeze}, p, \mathcal{B})$ to $\mathcal{L}$, output $(\text{blame}, id, \mathcal{S})$ to all, and terminate. Otherwise, send (bought, id, x) to $\mathcal{B}$ and go to Payout.

Payout

(Round 4) Upon receiving (abort, id) from corrupted $\mathcal{B}^*$, send (sold, id) to $\mathcal{S}$, $(\text{unfreeze}, p, \mathcal{S})$ to $\mathcal{L}$ and terminate. Otherwise:

- If $\phi_{\text{FS}}(x) = 1$: $(\text{unfreeze}, p, \mathcal{S})$ to $\mathcal{L}$, (sold, id) to $\mathcal{S}$.
- If $\phi_{\text{FS}}(x) \neq 1$: $(\text{unfreeze}, p, \mathcal{B})$ to $\mathcal{L}$, output $(\text{blame}, id, \mathcal{S})$ to all.

Fig. 3: Ideal functionality $\mathcal{F}_{\text{FS}}^{\mathcal{L}}$ for Revised FairSwap.

Proof Sketch. We show blame soundness and blame completeness. In the real world, the honest buyer $\mathcal{B}$ receives $(\text{sell}, z, \phi_{\text{FS}}, c, p, \sigma)$ from $\mathcal{S}^*$, verifies σ via $\Pi_{SIG}.\mathsf{Verify}_{pk_S}$ and the parameter consistency, and produces $\sigma' \leftarrow \Pi_{SIG}.\mathsf{Sign}_{sk_B}(id\|\sigma)$. In the ideal world, Sim receives the leakage from $\mathcal{F}_{\text{FS}}^{\mathcal{L}}$ and performs the same verification and signing steps in the protocol transcript for $\mathcal{Z}$. If $\mathcal{S}^*$'s signature σ or the parameter consistency check fails, Sim halts the simulation before initiation; no blame output is produced in either world.

In the real world, $\mathcal{S}^*$ submits $(\text{init}, id, p, c, r_{\phi_{\text{FS}}}, r_z, \sigma, \sigma', pk_S, pk_B)$ to $\mathcal{G}_{\text{FS}}^{\mathcal{L},\mathcal{H}}$, which verifies both signatures. By EUF-CMA security, σ binds $\mathcal{S}^*$ to $id\|r_{\phi_{\text{FS}}}\|r_z\|c\|p$; if $\mathcal{S}^*$ tampers with any parameter, the verification fails and the contract rejects. Thus $\mathcal{S}^*$ cannot deny having committed to these parameters, establishing blame soundness.

In the real world, after initiation, the honest buyer sends (accept, id) and the judge contract locks coins. The following two cases establish blame completeness. If $\mathcal{S}^*$ fails to reveal

$\mathcal{G}_{\text{FS}}^{\mathcal{L},\mathcal{H}}$ acts as a judge contract for session id, interacting with $\mathcal{L}$ and parties $\mathcal{S}$, $\mathcal{B}$. Stores $pk_{\mathcal{S}}$, $pk_{\mathcal{B}}$, p, c, k, r_z, $r_{\phi_{\text{FS}}}$, σ, σ', state s.

Initialize Phase

(Round 1) Upon receiving $(\text{init}, id, p, c, r_{\phi_{\text{FS}}}, r_z, \sigma, \sigma', pk_S, pk_B)$ from $\mathcal{S}$ when $\Pi_{SIG}.\mathsf{Verify}_{pk_B}(\sigma', id\|\sigma) \stackrel{?}{=} 1$ and $\Pi_{SIG}.\mathsf{Verify}_{pk_S}(\sigma, id\|r_{\phi_{\text{FS}}}\|r_z\|c\|p) \stackrel{?}{=} 1$, store parameters, output $(\text{initialized}, id)$, set $s =$ *initialized*.
(Round 2) Upon (accept, id) from $\mathcal{B}$ when $s =$ *initialized*, send $(\text{freeze}, \mathcal{B}, p)$ to $\mathcal{L}$. If frozen, set $s =$ *active* and output $(\text{accepted}, id)$.
Upon timeout when $s =$ *initialized*: output $(\text{blame}, id, \mathcal{B})$, terminate.

Reveal Phase

(Round 3) Upon $(\text{reveal}, id, d, k)$ from $\mathcal{S}$ when $s =$ *active* and $\Pi_{COM}.\mathsf{Open}(c, d, k) = 1$, send $(\text{revealed}, id, d, k)$ to all, set $s =$ *revealed*.
Upon timeout when $s =$ *active*: $(\text{unfreeze}, p, \mathcal{B})$ to $\mathcal{L}$, output $(\text{blame}, id, \mathcal{S})$.

Payout Phase

(Round 4) Upon m from $\mathcal{B}$ when $s =$ *revealed*, set $s =$ *finalized*:

- If $m = (\text{complain}, id, \pi)$ and $\text{Judge}(k, r_z, r_{\phi_{\text{FS}}}, \pi) = 1$: $(\text{unfreeze}, p, \mathcal{B})$ to $\mathcal{L}$, output $(\text{blame}, id, \mathcal{S})$.
- Otherwise: $(\text{unfreeze}, p, \mathcal{S})$ to $\mathcal{L}$.

Upon timeout when $s =$ *revealed*: $(\text{unfreeze}, p, \mathcal{S})$ to $\mathcal{L}$.

Fig. 4: Ideal functionality $\mathcal{G}_{\text{FS}}^{\mathcal{L},\mathcal{H}}$ for judge contract of Revised FairSwap.

(d, k) or sends invalid parameters, $\mathcal{G}_{\text{FS}}^{\mathcal{L},\mathcal{H}}$ outputs $(\text{blame}, id, \mathcal{S})$ and refunds $\mathcal{B}$. If $\mathcal{S}^*$ reveals (d, k) but the decoded goods fails $\phi_{\text{FS}}(\cdot)$, the honest buyer submits a complaint and $\mathcal{G}_{\text{FS}}^{\mathcal{L},\mathcal{H}}$ outputs $(\text{blame}, id, \mathcal{S})$ via fraud detection. In the ideal world, $\mathcal{F}_{\text{FS}}^{\mathcal{L}}$ outputs $(\text{blame}, id, \mathcal{S})$ in both cases. The blame outputs and refunds are identical in both worlds; hence $\mathcal{Z}$ cannot distinguish. □

Lemma 4 (FairSwap: Corrupted Buyer)**.** *If the buyer $\mathcal{B}$ is corrupted and the seller $\mathcal{S}$ is honest, every abnormal termination after initiation on the judge contract $\mathcal{G}_{\text{FS}}^{\mathcal{L},\mathcal{H}}$ outputs $(\text{blame}, id, \mathcal{B})$, and Sim produces the corresponding $(\text{blame}, id, \mathcal{B})$ output in $\mathcal{F}_{\text{FS}}^{\mathcal{L}}$.*

Proof Sketch. We show blame soundness and blame completeness. In the real world, the honest seller $\mathcal{S}$ computes $\sigma \leftarrow \Pi_{SIG}.\mathsf{Sign}_{sk_S}(id\|r_{\phi_{\text{FS}}}\|r_z\|c\|p)$ and sends $(\text{sell}, z, \phi_{\text{FS}}, c, p, \sigma)$ to $\mathcal{B}^*$. In the ideal world, Sim receives the leakage from $\mathcal{F}_{\text{FS}}^{\mathcal{L}}$ and produces the corresponding $(\text{sell}, z, \phi_{\text{FS}}, c, p, \sigma)$ message in the transcript for $\mathcal{Z}$. If $\mathcal{B}^*$ does not respond with a valid σ', Sim halts the simulation before initiation; no blame output is produced in either world.

In the real world, $\mathcal{G}_{\text{FS}}^{\mathcal{L},\mathcal{H}}$ verifies both σ and σ' upon receiving (init) from the honest seller. By EUF-CMA security, $\mathcal{B}^*$ cannot forge σ without sk_S; if σ' is tampered, the verification fails and the contract rejects. If the verification passes, (σ, σ') certifies that $\mathcal{B}^*$ accepted $r_{\phi_{\text{FS}}}$, r_z, c, and p, and $\mathcal{B}^*$ cannot later deny this acceptance, establishing blame soundness.

The following two cases after initiation establish blame completeness. If $\mathcal{B}^*$ does not send (accept, id) before the timeout: in the real world, $\mathcal{G}_{\text{FS}}^{\mathcal{L},\mathcal{H}}$ outputs $(\text{blame}, id, \mathcal{B})$; in the ideal world, $\mathcal{F}_{\text{FS}}^{\mathcal{L}}$ outputs $(\text{blame}, id, \mathcal{B})$ upon the corrupted $\mathcal{B}^*$'s failure to accept. The blame outputs are identical; hence $\mathcal{Z}$ cannot distinguish. If $\mathcal{B}^*$ sends (accept, id): in the real world, the honest seller reveals (d, k); since $\mathcal{S}$ is honest and $\phi_{\text{FS}}(x) = 1$, any complaint π from $\mathcal{B}^*$ is rejected by $\text{Judge}(k, r_z, r_{\phi_{\text{FS}}}, \pi) = 0$, and the coins are released to $\mathcal{S}$; if $\mathcal{B}^*$ does not complain, the coins are also released to $\mathcal{S}$. In the ideal world, Sim simulates the reveal step and $\mathcal{F}_{\text{FS}}^{\mathcal{L}}$ completes normally. No (blame, id, P) output is produced in either world, and $\mathcal{Z}$ cannot distinguish. Thus, the only abnormal termination after initiation is the timeout at the accept phase, correctly producing $(\text{blame}, id, \mathcal{B})$ in both worlds, establishing blame completeness. □

VIII. Feasibility Discussion

This section measures the computational overhead of the revised ZKCP and FairSwap.

We implement the revised ZKCP and FairSwap protocols using Solidity with compiler version 0.8.27 and measure their gas costs on an Ethereum Virtual Machine. For cryptographic primitives, we use the Ethereum's native ECDSA for digital signatures, and Keccak-256 [27] in FairSwap and Poseidon [31] in ZKCP for the hash functions. The judge contract of both protocols handles only the symmetric key and hash of goods. Therefore, the gas costs remain approximately constant regardless of the size of the goods.

We measure the gas costs of the revised protocols compared to the originals. Table I presents the results. We only measure the gas costs for the modified messages: $(\text{accept}, \dots)$ in ZKCP and $(\text{init}, \dots)$ in FairSwap. Whether the sending party is corrupted or honest does not affect the gas costs. This is because the judge contract performs the same signature verification on every received message regardless of its content.

The revised ZKCP incurs approximately $2.77\times$ the gas costs of the original, and the revised FairSwap incurs approximately $2.25\times$ the gas costs of the original. Both protocols exhibit nearly the same increase rate because our modification is identical for both: the judge contract additionally verifies and stores two digital signatures (σ, σ').

TABLE I: Comparison of Gas Costs

Protocol	Message	Original	Revised	Rate
ZKCP	accept	111,352	307,967	276.6%
FairSwap	init	157,693	355,520	225.4%

IX. Conclusion

We identified that the prominent fair exchange protocols ZKCP and FairSwap lack accountability in abnormal termination. The lack of accountability limits the applicability of fair exchange protocols in decentralized settings. To fill this gap, we proposed *signature exchange and chaining* that provides the judge contract with cryptographic evidence that both parties verified and accepted protocol-specific parameters before the exchange begins. We revised the two protocols with this approach and provided a proof sketch, and measured approximately $2.5\times$ increase in gas cost.

References

[1] A. Asgaonkar and B. Krishnamachari, "Solving the buyer and seller's dilemma: A dual-deposit escrow smart contract for provably cheat-proof delivery and payment for a digital good without a trusted mediator," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. Seoul, Korea (South): IEEE, 2019, pp. 262–267.

[2] N. Asokan, "Fairness in electronic commerce," Ph.D. dissertation, University of Waterloo, 1998. [Online]. Available: http://www.pqdtcn.com/thesisDetails/D99F5B10FA887285C8295656E5B1171E

[3] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," Cryptology ePrint Archive, Paper 2014/129, 2014, https://eprint.iacr.org/2014/129. [Online]. Available: https://eprint.iacr.org/2014/129

[4] P. Resnick and R. Zeckhauser, "Trust among strangers in internet transactions: Empirical analysis of ebay' s reputation system." in *The Economics of the Internet and E-commerce*, ser. Advances in Applied Microeconomics, M. R. Baye, Ed. Emerald, 2002, vol. 11, pp. 127–157. [Online]. Available: http://dblp.uni-trier.de/db/books/collections/B2002.html#ResnickZ02

[5] R. Song, B. Xiao, Y. Song, S. Guo, and Y. Yang, "A survey of blockchain-based schemes for data sharing and exchange," *IEEE Transactions on Big Data*, vol. 9, no. 6, pp. 1477–1495, 2023.

[6] Bitcoin Wiki, "Zero knowledge contingent payment," https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment, 2018, (Accessed on 09/14/2024).

[7] M. Campanelli, R. Gennaro, S. Goldfeder, and L. Nizzardo, "Zero-knowledge contingent payments revisited: Attacks and payments for services," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 229–243. [Online]. Available: https://doi.org/10.1145/3133956.3134060

[8] S. Dziembowski, L. Eckey, and S. Faust, "Fairswap: How to fairly exchange digital goods," in *Proceedings of the 2018ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 967–984. [Online]. Available: https://doi.org/10.1145/3243734.3243857

[9] L. Eckey, S. Faust, and B. Schlosser, "Optiswap: Fast optimistic fair exchange," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 543–557. [Online]. Available: https://doi.org/10.1145/3320269.3384749

[10] R. Künnemann, D. Garg, and M. Backes, "Accountability in the decentralised-adversary setting," in *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*, 2021, pp. 1–16.

[11] R. Küsters, T. Truderung, and A. Vogt, "Accountability: definition and relationship to verifiability," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 526–535. [Online]. Available: https://doi.org/10.1145/1866307.1866366

[12] A. Haeberlen, P. Kouznetsov, and P. Druschel, "Peerreview: practical accountability for distributed systems," in *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 175–188. [Online]. Available: https://doi.org/10.1145/1294261.1294279

[13] A. R. Yumerefendi and J. S. Chase, "The role of accountability in dependable distributed systems," in *Proceedings of the First Conference on Hot Topics in System Dependability*, ser. HotDep'05. USA: USENIX Association, 2005, p. 3.

[14] ——, "Trust but verify: accountability for network services," in *Proceedings of the 11th Workshop on ACM SIGOPS European Workshop*, ser. EW 11. New York, NY, USA: Association for Computing Machinery, 2004, p. 37–es. [Online]. Available: https://doi.org/10.1145/1133572.1133585

[15] R. Abe, S. Kurita, M. Kobayashi, and S. Suzuki, "Mitigation of seller and buyer's dilemma with transaction history and escrow," in *Proceedings of the 18th Asian Internet Engineering Conference*, ser. AINTEC '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 1–9. [Online]. Available: https://doi.org/10.1145/3630590.3630591

[16] R. Canetti, "Universally composable security: a new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. Newport Beach, CA, USA: IEEE, 2001, pp. 136–145.

[17] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger berlin version beacfbd– 2022-10-24," 2014. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[18] Etherscan, "Ethereum average block time chart — etherscan," https://etherscan.io/chart/blocktime, 2023, (Accessed on 01/05/2024).

[19] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," https://bitcoin.org/bitcoin.pdf, 2008, (Accessed on 02/06/2023).

[20] E. Foundation, "ethereum/consensus-specs: Ethereum proof-of-stake consensus specifications," https://github.com/ethereum/consensus-specs, 2023, (Accessed on 01/06/2024).

[21] M. Abadi and N. Glew, "Certified email with a light on-line trusted third party: Design and implementation," in *Proceedings of the 11th International Conference on World Wide Web*, ser. WWW '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 387–395. [Online]. Available: https://doi.org/10.1145/511446.511497

[22] N. Asokan, V. Shoup, and M. Waidner, "Optimistic fair exchange of digital signatures," *IEEE journal on selected areas in communications*, vol. 18, no. 4, pp. 593–610, Apr 2000. [Online]. Available: https://ieeexplore.ieee.org/document/839935

[23] M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest, "A fair protocol for signing contracts," *IEEE transactions on information theory*, vol. 36, no. 1, pp. 40–46, Jan 1990. [Online]. Available: https://ieeexplore.ieee.org/document/50372

[24] M. Mut-Puigserver, M. A. Cabot-Nadal, and M. M. Payeras-Capella, "Removing the trusted third party in a confidential multiparty registered edelivery protocol using blockchain," *IEEE access*, vol. 8, pp. 106 855–106 871, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9110560

[25] J. L. Ferrer-Gomilla, J. A. Onieva, M. Payeras, and J. Lopez, "Certified electronic mail: Properties revisited," *Computers & security*, vol. 29, no. 2, pp. 167–179, 2010. [Online]. Available: https://dx.doi.org/10.1016/j.cose.2009.06.009

[26] H. Vogt, H. Pagnia, and F. Gartner, "Modular fair exchange protocols for electronic commerce," in *Proceedings 15th Annual Computer Security Applications Conference (ACSAC'99)*. Phoenix, AZ, USA: IEEE, 1999, pp. 3–11. [Online]. Available: https://ieeexplore.ieee.org/document/816008

[27] National Institute of Standards and Technology, "SHA-3 standard: Permutation-based hash and extendable-output functions," National Institute of Standards and Technology, Tech. Rep. FIPS PUB 202, 2015.

[28] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ser. ITCS '12. New York, NY, USA: Association for Computing Machinery, 2012, pp. 326–349.

[29] J. Groth, "On the size of pairing-based non-interactive arguments," in *Advances in Cryptology – EUROCRYPT 2016*, ser. Lecture Notes in Computer Science, vol. 9666. Vienna, Austria: Springer, 2016, pp. 305–326.

[30] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in *Theory of Cryptography*, S. P. Vadhan, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 61–85.
[31] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for Zero-Knowledge proof systems," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2021, pp. 519–535.