Sponsored Fair Exchange (Extended Abstract)

Serge Vaudenay

EPFL, Lausanne, Switzerland

Abstract. We propose a costless platform for fair exchange based on smart contracts to favor economic inclusion. Our smart contract is minimal, as most of communication is done offchain. The smart contract costs are covered by incentivized sponsors. Our protocol is a knowledge-coin exchange: it allows to exchange a digital item, characterized by an automatically verifiable description, against a payment in cryptocurrency. The exchange is fair in the sense that either both parties receive what they expect (the exchange completes) or both parties lose nothing (the exchange is canceled). We ensure a costless transaction in cancelation cases, and a transaction with pre-determined fee if it completes. We also ensure privacy of the transaction. Our protocol offers an improvement compared to OptiSwap: we can work with any description function. The complexity is never higher and sometimes significantly smaller. Namely, the worst case complexity is logarithmic instead of being linear. Furthermore, we introduce sponsoring as an enabler for economic inclusion.

1 Introduction

The advent of blockchains is an important milestone towards decentralized economic systems. However, only a negligible fraction of people is using it. We believe that this is due to many obstacles. First of all, there are fees to pay for every transaction. Fees are essentially flat rate (but fluctuate), so they can be important when the transaction is small. Second, the decentralized nature opens new opportunities to malicious behaviors. Automation and anonymity somehow encourage malicious actors as they give a sense of impunity. In this paper, we set up tools to solve some of these problems.

We focus on the specific problem of knowledge-coin exchange. We consider a vendor V who knows some digital information x and who would like to sell it to a buyer B at a price of $\$y$. We want the transaction to be effortless, costless, and fair.

As it is usual in fair exchange [3], the digital item x is characterized by a description function desc which can be objectively evaluated by any participant. For instance, desc can be a cryptographic hash function. Item x is characterized by the property $\mathsf{desc}(x) = d$, where desc and d are known by B prior to the exchange. Our protocol is *universal* in the sense that it works with any desc function. The payment of $\$y$ is done in cryptocurrency on a smart contract-enabled blockchain. The exchange can complete or be canceled. In a complete case, V (if honest) receives $\$y$ and B (if honest) receives x' such that $\mathsf{desc}(x') = d$. In a cancel case, V (if honest) does not reveal x and B (if honest) does not pay.

Applications could be storage services where we would pay $\$y$ to retrieve a file x characterized by its hash d. We can also consider the market of media content (movies, songs, photographs), assuming that we can rely on a trusted "catalog" which provides the verifiable description of each file (such as its hash). Having a catalog committed on a consensus-based public ledger would establish such trust, based on reputation.

We want this exchange to be *fair* in the sense that an honest V will never end up in revealing information about x without being paid, and that an honest B will never pay without receiving an x' satisfying $\mathsf{desc}(x') = d$. This is the classical *fair exchange* problem, which is made possible by blockchain consensus.

To reach a better economic inclusion, we do not want any entrance fee for V and B: V has nothing to pay (namely: no gas expense at all), and B has no extra fee, except a negotiated tip to pay to a third party only if the transaction succeeds. This third party commits to paying the processing fees (gas costs) of the transaction. They act as a *sponsor* who takes risks of paying

fees without any advantage, but who would be rewarded with a tip if the transaction completes. The sponsor may also be the victim of malicious V and B making the transaction fail to make the sponsor lose money. However, we view the sponsor as a dedicated organization (corporate or not-for-profit) who can afford taking such risks over large volumes of transactions (given that fees are very small) and protect by other means. We note that this type of sponsor already exist in some form: In platforms such as Alchemy, bundlers sponsor gas costs on the blockchain. In Bitcoin, miners spend energy for free except in rare cases when they receive an award. However, in all those cases, sponsors may have an impact on fairness due to possible censorship attacks.

Finally, we want a high level of security for honest participants and not only a protection against rational adversaries. The costless property comes from the assumption that a sponsor exists and by the protocol design which accommodates sponsoring by any participant. The sponsor could actually be V or B. This sponsor could also be a decentralized autonomous organization. We finally stress that we need no trust assumption in the sponsor: a sponsor may be malicious.

The contribution of this paper is twofold. We improve the existing OptiSwap protocol [8] to allow arbitrary description functions at a logarithmic cost instead of a Merkle hash description function. Independently, we introduce the notion of sponsors and identify their roles in a fair exchange protocol.

Previous work. Fair exchange has been a long lasting research topic. It found relations with certified email (exchange the delivery of an email against a signed receipt), payment with receipt (exchange of a payment against a signed receipt), contract signing (exchange signatures on a joint document), atomic swap (exchange of — say — bitcoins against ethers), and many more.

In 1980, Even and Yacobi (informally) proved that an exchange protocol between two participants could not be perfectly fair [9]. To overcome that impossibility, designers added a trusted third party. Blockchains enable the notion of smart contract. This implements trustworthy and immutable processes which can be thought of as trusted third parties. One difficulty to use them, compared to regular trusted execution platforms, is that they are fully transparent and cannot keep secrets. So, they cannot sign and they should not process any private information.

One example of an elementary smart contract is the "Hash Time-Locked Contract" (HTLC) [16]. It is implemented in the bitcoin architecture and many others. Essentially, HTLC enables the fair exchange of some payment in cryptocurrency against the preimage of a specified hash. The party willing to buy the preimage sets up the HTLC and pays to the smart contract. Then, the other party sends the preimage to the smart contract to get the payment. The paying party can see the preimage in the HTLC and get it. This is the *complete* termination state. However, after a timeout during which no preimage was sent, the HTLC returns the payment to the payee. This is the *cancel* termination state. An HTLC solves some aspects of our problem. It restricts to a particular desc function: a hash function on a key space. It lets B pay the fee for the deployment of the HTLC and the payment, and V pay the fee to post the key.

In general, fair exchange (of a verifiable digital good against a payment) can be made using verifiable encryption and HTLC. In protocols such as Zero-Knowledge Contingent Payment (ZKCP) [2], the vendor encrypts the digital good with a new key. This key is hashed. The vendor sends to the buyer (offchain) the encrypted good, the hash of the key, and a zero-knowledge proof that what was encrypted satisfies the description and that it was encrypted with what was hashed. Then, B can set an HTLC by paying the agreed price, to be released with a preimage of the hashed key. Many protocols are based on this approach: ZKCPlus [14], Cross-Channel [10], Smart ZKCP [15], Fair Data Exchange (FDE) [17], zkMarket [13], DVVZ[1] [5], K-FDE[2] [12]. However, a zero-knowledge proof that an encrypted item satisfies the expected description can be very expensive depending on the description function.

In 1990, Bürk and Pfitzmann noticed that there can be protocols in which the trusted third party is involved only in rare dispute cases [4]. Later on, the notion of *optimistic* fair exchange was

[1] Della Monica et al. [5] proposed a protocol that we name DVVZ after the initials of the authors for convenience.

[2] We name this protocol K-FDE as it is an extension of FDE by Khabbazian.

proposed. This notion allows two parties to exchange in an optimistic manner, but in the event that it ends up in a dispute, they resolve it by implying the trusted third party. The protocols FairSwap [7] and OptiSwap [8] use the optimistic approach by starting the optimistic part with no zero-knowledge proof and keeping the proof (not necessarily zero-knowledge) in the dispute resolution. Instead of V proving their good behavior at the beginning, B proves bad behavior from V during dispute resolution. One advantage is that it needs not be done on encrypted data so it is much faster. OptiSwap improves upon FairSwap in further decreasing the cost of the dispute resolution by interaction. The complexity depends on the depth of the circuit implementing the description function.

In OptiSwap, the predicate verifying x is encoded as a circuit. The dispute resolution consists of identifying a gate where participants agree on the input but not on the output. It works as a descent in the circuit, which is why the complexity in linear in its depth.

In blockchains such as Arbitrum [11], smart contracts are not executed by nodes but by involved participants (called administrators) who sign that they agree on its state after execution. If a dispute occurs, they must identify where their views on the execution diverge. At each execution, they commit to the input/output states of the virtual machine and proceed in a cut-and-choose manner in complexity $\mathcal{O}(\log T)$, where T is the time complexity (the number of state transitions in the virtual machine) of the smart contract. When the divergence point is found, they provide a proof of a one-step execution. For that, they need to provide some parts of the state and prove consistency with the commitment. The consistency proof, based on a Merkle hash, has complexity $\mathcal{O}(\log M)$ where M is the memory complexity M of the smart contract. Hence, their complexity is in $\mathcal{O}(\log(TM))$. The advantage of their approach is that smart contract developers do not have to care about how it is executed or disputes are resolved. In our paper, we propose something in between OptiSwap and Arbitrum: like for OptiSwap, we use the circuit description so that the complexity does not depend on M, but we use a linear sequence to evaluate the gates and look for the first step of disagreement like in Arbitrum. Our complexity is in $\mathcal{O}(\log n)$, where n is the number of gates in the circuit.

Some blockchains such as Ethereum allow transaction fees to be sponsored by a third party, following ERC-2771, EIP-3074, or ERC-4337. Quite often, this sponsoring option is motivated by the ability to reduce the barrier of entry for new users. We push this further to accommodate users who wish to pay fees only if they can get what they expect. However, integrating this option in a critical protocol such as a fair exchange requires some careful analysis. As far as we know, sponsoring has not been considered in the fair exchange literature. For instance, ERC-2771 and EIP-3074 allow the sponsor (*relayer* or *delegate*) to easily cancel their support as they are not immutable. In our setting, this would be an avenue to censorship attacks and would break fairness. ERC-4337 allows gas sponsoring by a smart contract (*paymaster*) to another smart contract. There, the sponsor is now immutable, so can be programmed to make this withdrawal impossible, but the user is normally an externally owned account; not a smart contract. We rely on concurrent services (*bundlers*) which verify the validity of a signed order before submitting it to the *entrypoint*. It implies that bundlers pay some gas but are paid back by the entrypoint with the deposit from the paymaster. Such infrastructure is already deployed, for instance in Alchemy.[3] The transaction to the paymaster looks like coming from a bundler. With the newly implemented EIP-7702, a normal user can delegate a bundler to do a transaction to the paymaster by remaining the originator.

Our first contribution. We push the development of fair exchange further by having costless transactions. Indeed, we remove the cost by relying on sponsors. The optimistic transaction sponsor agrees to "invest" on supporting the exchange in hope to collect a reward tip_o if the transaction succeeds. The "investment" to sponsor a transaction is rather small, so we believe that the risk can be mitigated by simple mechanisms. Contrarily, the fees in a dispute are much higher but sponsoring a dispute by an honest sponsor is totally riskless. An honest dispute resolution sponsor

[3] `https://www.alchemy.com/docs/wallets/transactions/sponsor-gas`
The provided library would set the tip to \$0.001575 plus 80% of the gas fees.
`https://www.alchemy.com/overviews/what-is-a-bundler`

Table 1. Comparison between Knowledge-Coin Fair Exchange Protocols on Ethereum

protocol	type	description	cost of honest exchange	cost of dispute
FDE	ZKCP	KZG	?? (deploy), 306 498 gas (run) for $\|x\|$ = 128KB: V:89sec, B:34sec	n/a
DVVZ	ZKCP	img resize	3 087 908 (deploy+mint), 562 166 gas (run) for x 631 × 631 image: V:1min, B:2sec (V:18GB RAM)	1 633 536 gas (run)
K-FDE	ZKCP	KZG	for $\|x\|$ = 32MB: V:5sec, B:1sec	n/a
OptiSwap	optimistic	Merkle-hash	952 939 gas (deploy), 101 307 gas (run) for $\|x\|$ = 1GB: V:??, B:??	1 962 992 gas (deploy) for $\|x\|$ = 1GB: 6 412 569 gas (run)
SOX	optimistic	SHA256	2 668 072 gas (deploy), 270 291 gas (run) for $\|x\|$ = 1GB: V:49sec, B:49sec	2 721 171 gas (deploy + steps 5–6) for $\|x\|$ = 1GB: 24 991 674 gas (run) V:37sec, B:37sec

is guaranteed to get a reward tip_d if they verify well the dispute argument before engaging. Finally, any sponsor can act maliciously (be bribed or corrupted) without harming any honest participant.

Unfairness could remain for an honest party if two conditions are met: the party does not have the seed money to sponsor themselves and the party could not find a sponsor for dispute resolution (even though there is no risk). However, given that sponsoring a dispute is riskless and that it can generate important benefits, we believe that this is not a problem as this type of sponsoring will appear in the form of a decentralized autonomous organization.

Our second contribution. Besides, we bring improvements compared to the OptiSwap protocol. In OptiSwap, the dispute resolution follows the structure of a *hardcoded* balanced circuit which encodes the description function. Dispute resolution follows an interactive process, in which one participant reveals the value of the sons of a node in the circuit, and their counterpart says on which son they disagree with the value so that they can go down by one node and iterate. The circuit *must be balanced* to keep low complexity. So, OptiSwap favors a description function defined by a Merkle hash. In our protocol, the description desc is *arbitrary* and provided by a committed circuit which is not necessarily balanced. The smart contract which is resolving the dispute does not depend on desc and can be deployed before desc is known. Only the set of instructions (the gates which are used in the circuit) which allow programming the circuit needs to be known. By making use of a subtle encoding with an incremental accumulator, we make sure that the dispute cost is logarithmic in the size of the circuit of desc in the worst case, instead of being linear in its depth as in OptiSwap. This is a substantial improvement in the case of imbalanced circuits. For instance, when desc is defined by a SHA256 hash on a big file x, our complexity is logarithmic in the length of x while it is linear with OptiSwap.

We stress that it is important that users can decide on which desc function to use and that this choice may not be imposed by the protocol designers. Hence, we cannot assume that the implementation of desc will give a balanced circuit.

High-level overview. The high-level idea of our protocol is as follows: V and B agree on the terms of the exchange (which include the circuit of desc), V encrypts x into ct and privately sends ct to B but keeps the symmetric encryption/decryption key k. The terms of the exchange and ct are committed by both participants into a smart contract sponsored by a new participant S. Messages by participants to the smart contract are pre-paid by S. B deposits the payment into the smart contract. Then, V deposits the key k. Then, B verifies that k helps to get the expected x. If it is not the case, B triggers a dispute. Otherwise, the payment goes to V and a tip goes to S. To engage in a dispute, each participants must find a sponsor to prepay the fees of the dispute. The sponsor of the winning party gets their money back and the unspent leftover by their counterpart while the other sponsor loses the prepaid amount. This is a winner-takes-all principle. The protocol is such that both V and B can prove to their potential sponsor when they are right and that a participant who is right cannot lose in the dispute. The proof generation and verification are simple (non zero-knowledge), and made offchain, so costless. The goal of the dispute is to make

the smart contract decide onchain without having to process or look at the x material who is right and should get the payment. Like in OptiSwap, our protocol is privacy-preserving in the sense that a malicious participant does not learn more about x than what usually leaks: its size leaks in any case, the description function leaks if either V or B is corrupted, and x itself leaks if B is corrupted but pays $\$y$ to V. This holds even with corrupted sponsors.

We can compare our SOX protocol with OptiSwap [8] (another optimistic protocol), FDE [17], DVVZ [5], and K-FDE [12] (protocols in the ZKCP family) in Table 1.[4] They all offer strong security. Only SOX is adapted to a standard SHA256 hash. It can adapt to any other desc function at low cost. FDE and K-FDE use the KZG polynomial commitment while OptiSwap uses Merkle hash (or any desc with balanced circuit). DVVZ [5] can in theory be adapted to any description function but it was only tried with the function reducing an image x by half to advertise a lower resolution with images x of resolution up to 631×631 pixels (so x is much less than 1MB). By looking at the five protocols (limited to the optimistic phase for OptiSwap and SOX), the gas cost and privacy guarantee are pretty similar. OptiSwap and SOX require a dispute resolution in dispute cases only. Since dispute requires that either V or B is malicious, privacy during the dispute is pointless here.[5]

To compare SOX and OptiSwap, the figures in the table could look like SOX is more expensive but this would be a bit misleading. Some of the additional costs (both in the optimistic phase and in the dispute) come from that we handle the logic of sponsoring, which adds more participants and more transactions. In the case of the dispute, this also multiplies the worst case cost by three to enforce fairness to sponsors. Another reason is that we implemented the dispute resolution to be generic and accommodate any description function that can be defined by a circuit using our set of gates, instead of having a hard-coded description function in the dispute solver. This creates an extra cost for proving the correctness of a gate.

We believe that our dispute resolution could find applications beyond fair exchange.

2 The Protocol

The exchange needs a description function desc and a claimed description output $d = \mathsf{desc}(x)$ to characterize x. Actually, d and desc define a predicate $\mathsf{desc}_d(\cdot)$ such that $\mathsf{desc}_d(x) \iff \mathsf{desc}(x) = d$.

2.1 Preliminary Phase

In this phase, V and B interact by whatever communication means they prefer in order to agree on a "pre-contract" which defines the terms of the transaction. This communication channel is supposed to be private (e.g. encrypted with TLS). If it is done over a public channel, it typically starts by a key agreement protocol which defines a "context" which will be part of the pre-contract. This part is typically done in a browser using a web interface to connect V and B. The goal of this phase is to define the input of the optimistic protocol.

The participants agree on

- the blockchain public key of V and B;
- the price y;
- the offered completion reward tip_o and dispute reward tip_d to sponsors;

[4] Comparison with HTLC or adaptor signature protocols is out of the scope, as these protocols are *not* general-purpose knowledge-coin exchange protocols but rather key-coin exchange ones. They avoid the difficulty of general-purpose protocols by ignoring a ZK phase or a dispute phase that protocols in Table 1 solve. Essentially, all protocols listed in the table rely on a form of HTLC. Likewise, we cannot compare with Arbitrum [11] which is not a fair exchange protocol.

[5] Malicious-V cases are pointless since the adversary knows x and can give it away to anyone anyway. The honest-V malicious-B case is relevant because we do not want to reveal the x of an honest V in the case that the exchange aborts. However, we prove that the protocol always ends up in a complete state in this case, so reveals x to B as a feature after B has paid for it. Clearly, such a paying malicious B can give x to anyone as well so privacy is pointless.

– the description predicate desc_d of x (the input x must be such that $\mathsf{desc}_d(x)$ is true for V);
– the implementation of desc_d into a circuit (this implies determining the number $m = |x|$ of x pieces, the number n of gates, and the circuit itself);
– some parameters params for the protocol (including timeout delays and algorithm choices) and a $\mathsf{version}$ reference;
– $\mathsf{context}$ (if any) to authenticate a key agreement transcript (ephemeral public keys) by a hash.

The private item x is not needed to define the pre-contract (except an upper bound on its length). However, V promises that $\mathsf{desc}_d(x)$ is true at this stage.

The tip tip_o is determined by the policy of the potential optimistic sponsor S while the tip tip_d is determined based on the maximal gas cost of the dispute resolution (based on the agreed implementation of desc_d) and the forecasted gas price.

Agreeing on the implementation of desc_d does not require to exchange the full description of the circuit, which can be the result of a "compiler". For instance, if $\mathsf{desc} = \mathsf{SHA256}$, the participants only need to agree on the compiler version for $\mathsf{SHA256}$ and on an upper bound for the blocklength of x.

The circuit compiler is using a set of available instructions in the dispute solver. The dispute solver is specified by $\mathsf{version}$. Hence, the compiler must be compatible with $\mathsf{version}$.

The choice of algorithms includes a one-time symmetric encryption scheme $\mathsf{Enc}/\mathsf{Dec}$, a commitment scheme $\mathsf{Com}/\mathsf{Open}$, an accumulator $\mathsf{Acc}/\mathsf{Prove}/\mathsf{Verify}$, and also an incremental accumulator $\mathsf{Acc}/\mathsf{Prove}/\mathsf{Verify}/\mathsf{ProveExt}/\mathsf{VerifyExt}$.

Building up the circuit. V and B define a predicate $\mathsf{valid}(k, \mathsf{ct})$ which depends on two variables k and ct, by

$$\mathsf{valid}(k, \mathsf{ct}) \iff \mathsf{desc}_d\left(\mathsf{Dec}_k(\mathsf{ct})\right)$$

The variable k represents a symmetric key while ct represents a ciphertext. The predicate valid is "compiled" into a circuit which we denote $\mathsf{circuit}[\mathsf{valid}]$. We assume that the agreed desc_d, m, n, params, and $\mathsf{version}$ in the preliminary phase are enough for V and B to build $\mathsf{circuit}[\mathsf{valid}]$ consistently.

The circuit is defined as a list G of "gates". For $n = |G|$ gates, we have $G = (g_1, \ldots, g_n)$. We also assume that ct can be written as a tuple of m objects $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_m)$. A gate $g_i = (\mathsf{op}, s_1, \ldots, s_a)$ is defined by a function op (actually, a code from an instruction set which is defined by the $\mathsf{version}$ and implemented in the dispute solver) and a list $(s_1, \ldots, s_a)$ of integers either between 1 and $i-1$ or between $-m$ and -1.

For $i > 0$, the function op in g_i takes a variable number $a+1$ of inputs $\mathsf{op}(k, v_1, \ldots, v_a)$ and should be easy to compute by a smart contract with $\mathcal{O}(1)$ gas complexity. For $i < 0$, we define "dummy gates" g_i with $a = 0$ which evaluate to the variables ct_{-i}. Actually, the m dummy gates $g_{-1}, \ldots, g_{-m}$ represent the variables $\mathsf{ct}_1, \ldots, \mathsf{ct}_m$.

We say that g_{s_j} is the jth son of $g_i = (\mathsf{op}, s_1, \ldots, s_a)$ in the circuit. The son relation defines an acyclic structure with a unique gate g_n which is the son of no other gate. This gate is called the "root" of the circuit.

The circuit $\mathsf{circuit}[\mathsf{valid}] = G = (g_1, \ldots, g_n)$ is put in an accumulator. In other words, $\mathsf{circuit}[\mathsf{valid}]$ is considered as the input vector of Acc with coordinates g_i. Thus, we compute $h_{\mathsf{circuit}} = \mathsf{Acc}(\mathsf{circuit}[\mathsf{valid}])$. For each gate g_i, it is possible to compute $\pi = \mathsf{Prove}(\mathsf{circuit}[\mathsf{valid}], i)$ and to verify that gate g_i is correct by checking $\mathsf{Verify}(h_{\mathsf{circuit}}, n, i, g_i, \pi)$.

To ensure that used operations op can be evaluated in the smart contract, we refer to an instruction set specified by a $\mathsf{version}$ number. The smart contract must be compatible with $\mathsf{version}$.

Example: media as hash preimage. To take an example, we assume that x is a media file which is characterized by its $\mathsf{SHA256}$ hash $d = \mathsf{SHA256}(x)$ [1]. We use one-time AES-CTR encryption [6] to encrypt. We split ct in 512-bit blocks. A typical Blu-ray disk would give x the length of 25GB, so $m = 2^{29}$. If x was one song, it would typically have 4MB, so $m = 2^{16}$.

We easily define a circuit using $n = 2m + 1$ gates using the compression function of $\mathsf{SHA256}$. We first use m gates which would decrypt each 512-bit block of ct with a counter value (provided

as an op parameter). Then, m gates will iteratively compress each decrypted block (the first gate of arity 1 uses the initial value in SHA256; the other gates have arity 2). Finally, one gate compares the obtained hash with d (hard-coded in op) to give a boolean result.

Note that the depth of the circuit is $2 + m$. OptiSwap would be inefficient here, with a worst case complexity of $\mathcal{O}(m)$ compared to our $\mathcal{O}(\log m)$.

V/B input: $\mathsf{ID_V}$, $\mathsf{ID_B}$, y, tip_o, tip_d, desc_d, m, n, circuit[valid], context
V extra input: x
B extra input: $\$(y + \mathsf{tip}_o)$
S input: $\$\mathsf{fee}_o$
SV/SB input: $\$(\mathsf{fee}_d + \mathsf{tip}_d)$

Engagement phase
1: V and B both compute $h_{\mathsf{circuit}} \leftarrow \mathsf{Acc}(\mathsf{circuit[valid]})$
2: V picks k at random
3: V computes $\mathsf{ct} \leftarrow \mathsf{Enc}_k(x)$
4: V computes $h_{\mathsf{ct}} \leftarrow \mathsf{Acc}(\mathsf{ct}_1, \ldots, \mathsf{ct}_m)$
5: V computes $(c, o) \leftarrow \mathsf{Com}(h_{\mathsf{circuit}}, h_{\mathsf{ct}})$
6: V sets
$$\mathsf{contract} = (\mathsf{ID_V}, \mathsf{ID_B}, y, m, n, c, \mathsf{tip}_o, \mathsf{tip}_d, \mathsf{params}, \mathsf{version}, \mathsf{context})$$
7: V sends (securely) ct, contract, o to B
8: B computes $h_{\mathsf{ct}} \leftarrow \mathsf{Acc}(\mathsf{ct}_1, \ldots, \mathsf{ct}_m)$
9: B takes c from contract
10: B verifies $(h_{\mathsf{circuit}}, h_{\mathsf{ct}}) = \mathsf{Open}(c, o)$
11: B verifies that elements in contract match the input
12: B sends contract to S
13: S decides to sponsor or not

Smart contract phase (fees taken on S-deposit)
14: **Step 1**: S deploys a smart contract with contract and deposits $\$\mathsf{fee}_o$, V and B verify
15: **Step 2**: if before deadline, receive $\$(y + \mathsf{tip}_o)$ from B, otherwise (Step $1'$) go to Cancel
16: **Step 3**: if before deadline, receive k from V, otherwise (Step $2'$) go to Cancel
17: B takes k and compute $x' = \mathsf{Dec}_k(\mathsf{ct})$
18: if $\mathsf{desc}_d(x') = \mathsf{true}$, B terminates with output x'
19: **Step 4**: if before deadline, receive a dispute trigger from B, otherwise (Step $3'$) go to Complete
20: B and V to look for potential sponsors by showing $\mathsf{argue} = (\mathsf{circuit[valid]}, \mathsf{ct}, o)$ and the smart contract
21: SV and SB verify argue and the contract, and decide whether to sponsor accordingly
22: **Step 5**: if before deadline, receive $\$(\mathsf{fee}_d + \mathsf{tip}_d)$ from SB, otherwise (Step $4'$) go to Complete
23: **Step 6**: if before deadline, receive $\$(\mathsf{fee}_d + \mathsf{tip}_d)$ from SV, otherwise (Step $5'$) go to Cancel

Dispute resolution (fees taken on dispute deposit)
24: **Step 7**: set $a = 1$ and $b = n + 1$
25: V and B compute val and hpre as
$$\mathsf{val}(i) = \mathsf{op}(k, \mathsf{val}(s_1), \ldots, \mathsf{val}(s_a))$$
$$\mathsf{hpre}(i) = \mathsf{Acc}(\mathsf{val}(1), \ldots, \mathsf{val}(i))$$
26: **Step** $7a$: set $i = \lfloor \frac{a+b}{2} \rfloor$
27: **Step** $7b$: if before deadline, receive $w_i = \mathsf{hpre}(i)$ from B, otherwise (Step $7a'$) V wins and go to Step 9
28: **Step** $7c$: if before deadline, receive "left" (V disagrees with w_i) or "right" (V agrees with w_i) from V, otherwise (Step $7b'$) B wins and go to Step 9
29: if "left", set $b \leftarrow i$ and discards any w_j with $j > i$
30: if "right", set $a \leftarrow i + 1$ and discards any w_j with $j \leq i$
31: if $a \neq b$, go to Step $7a$
32: **Step 8**: set $i \leftarrow a$
33: receive from V (at the same time as the last "left/right"; if missing, B wins and go to Step 9) $(o, g_i, v_1, \ldots, v_a, w'_i, \pi_1, \pi_2, \pi_3, \rho)$, depending on the cases as detailed in the table below
34: verify the information from V as detailed in the table below
35: if correct, V wins, otherwise, B wins
36: **Step 9**: let $\mathsf{Q} \in \{\mathsf{V}, \mathsf{B}\}$ be the losing party
37: if Step 9 is reached for the first time, or for the second time but with a different Q, and if $\mathsf{Q} \neq \mathsf{SQ}$, set $\mathsf{Q} \leftarrow \mathsf{SQ}$ and go to Step 7
38: if V wins, go to Complete, otherwise, go to Cancel

39: **Cancel**: return the B-payment to B, the leftover S-deposit to S, the leftover dispute deposit to SB, and terminate

40: **Complete**: from the B-payment, pay $\$y$ to V, tip_o to S, return the leftover S-deposit to S and the leftover dispute deposit to SV, and terminate

case	V computation (line 33)	verification (line 34)
$1 < i \leq n$	$L = \{\ell; s_\ell < 0\}$ $v_\ell = \mathsf{val}(s_\ell) \quad (\ell = 1, \ldots, a)$ $w'_i = \mathsf{hpre}(i)$ $\pi_1 = \mathsf{Prove}(\mathsf{circuit[valid]}, i)$ $\pi_2 = \mathsf{Prove}(\mathsf{ct}, (-s_\ell)_{\ell \in L})$ $\pi_3 = \mathsf{Prove}((\mathsf{val}(1), \ldots, \mathsf{val}(i-1)), (s_\ell)_{\ell \notin L})$ $\rho = \mathsf{ProveExt}(\mathsf{val}(1), \ldots, \mathsf{val}(i))$	$L = \{\ell; s_\ell < 0\}$ $\mathsf{Verify}(h_{\mathsf{circuit}}, n, i, g_i, \pi_1)$ $\mathsf{Verify}(h_{\mathsf{ct}}, m, (-s_\ell)_{\ell \in L}, (v_\ell)_{\ell \in L}, \pi_2)$ $\mathsf{Verify}(w_{i-1}, i, (s_\ell)_{\ell \notin L}, (v_\ell)_{\ell \notin L}, \pi_3)$ $\mathsf{VerifyExt}(i, w_{i-1}, w'_i, v, \rho)$ $w_i \neq w'_i$
$i = 1$	$v_\ell = \mathsf{ct}_{-s_\ell} \quad (\ell = 1, \ldots, a)$ $w'_i = \mathsf{hpre}(i)$ $\pi_1 = \mathsf{Prove}(\mathsf{circuit[valid]}, i)$ $\pi_2 = \mathsf{Prove}(\mathsf{ct}, (-s_1, \ldots, -s_a))$ $\rho = \mathsf{ProveExt}(\mathsf{val}(1))$	$\mathsf{Verify}(h_{\mathsf{circuit}}, n, i, g_i, \pi_1)$ $\mathsf{Verify}(h_{\mathsf{ct}}, m, (-s_1, \ldots, -s_a), (v_1, \ldots, v_a), \pi_2)$ $\mathsf{VerifyExt}(i, \emptyset, w'_i, v, \rho)$ $w_i \neq w'_i$
$i = n + 1$	$\pi_3 = \mathsf{Prove}(\mathsf{val}, n)$	$\mathsf{Verify}(w_{i-1}, i, n, \mathsf{true}, \pi_3)$

Fig. 1. The SOX Protocol

2.2 Engagement Phase, Smart Contract Phase, Dispute Resolution

Following the preliminary phase, the protocol executes as specified in Fig. 1. We summarize here the three phases of this execution.

During the *engagement phase*, V encrypts x with a fresh random key k into $\mathsf{ct} = \mathsf{Enc}_k(x)$, then computes a binding hash h_{circuit} for the circuit, a binding hash h_{ct} for ct, and a binding commitment c for $(h_{\mathsf{circuit}}, h_{\mathsf{ct}})$. The value c is sent to B. The ciphertext ct and the opening key o of the commitment are *privately* sent to B. Then, B can check that c commits to the right h_{circuit} and h_{ct}. V and B agree on the terms of a *contract* which is a tuple

$$\mathsf{contract} = (\mathsf{ID}_\mathsf{V}, \mathsf{ID}_\mathsf{B}, y, m, n, c, \mathsf{tip}_o, \mathsf{tip}_d, \mathsf{params}, \mathsf{version}, \mathsf{context})$$

then appoint a sponsor S.

In the *smart contract phase*, S creates a smart contract with a deposit of $\$\mathsf{fee}_o$, then B deposits $\$(y+\mathsf{tip}_o)$. Following that, V deposits the key k on the smart contract. B can check if by decrypting ct with k they can obtain some item x' following the description predicate. When it is not the case, B prepares an argument consisting of $(\mathsf{circuit}, \mathsf{ct}, o)$ to convince a potential sponsor that they are entitled to launch a dispute. A potential sponsor can verify the argument by using what is stored on the smart contract (i.e. c and k). Hence, B can appoint a sponsor SB who deposits $\$(\mathsf{fee}_d+\mathsf{tip}_d)$. By seeing the dispute coming, V similarly prepares an argument and appoints a sponsor SV who also deposits $\$(\mathsf{fee}_d + \mathsf{tip}_d)$. Then, the execution moves on to the next phase. If any of those steps is not activated, after a given delay, the protocol ends either by a Cancel or by a $\mathsf{Complete}$ and pays each participant V, B, S, SV, SB with the deposited money accordingly.

In the *dispute resolution*, we have a dispute between two pairs of participants $(\mathsf{V}, \mathsf{SV})$ and $(\mathsf{B}, \mathsf{SB})$. One member of each pair runs the iterations of the protocol to identify the first gate of disagreement in the evaluation of the circuit. This gate is then evaluated by the smart contract to determine who wins. The participant of the losing pair who did not run the protocol takes over and the iterations restart. If the outcome is different, the losing participant who did not run the protocol so far can take over for a third and last run. The winning decision is then final and determines if the protocol completes or cancels.

3 Formal Result

We summarize our formal results as follows.

Theorem 1 (Main result). *Our protocol is a SOX enforcing consensus, privacy, and fairness. For this, we assume no censorship in the blockchain. We assume that communication between* V *and* B *during the preliminary phase and the engagement phase is private. We assume that an honest party in a dispute always finds a sponsor. We also assume security of the underlying cryptographic algorithms:* Enc *is* INDCPA*-secure,* Com *is* HIDE*-secure and* BIND*-secure, the static accumulator is* CR*-secure, the incremental accumulator is* CR*-secure and* CRext*-secure.*

4 Conclusion

We formalized the notion of optimistic sponsored exchange with no entrance fee. We achieve fairness and privacy of the exchange against malicious adversaries. We succeeded this at very low cost as we could have an optimistic phase with nearly no computation in the smart contract.

Acknowledgements. This work was done in cooperation with CIMA.Science. We would like to deeply thank Alfio Lanuto for providing the infrastructure and for encouraging. We are also grateful to Jonathan Poveda Colominas, Philip Hamelink, Hana El Moutaoukil, and Ryad Aouak for all their work in the implementation. Finally, we thank Microsoft Research for hosting a presentation of this work.[6]

[6] https://www.microsoft.com/en-us/research/video/knowledge-coin-fair-exchange/

References

1. Secure Hash Standard (SHS). *Federal Information Processing Standard* publication #180-4. U.S. Department of Commerce, National Institute of Standards and Technology, 2015.
2. Zero Knowledge Contingent Payment. Bitcoin wiki. 2018. https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment
3. N. Asokan, Victor Shoup, Michael Waidner. Optimistic Fair Exchange of Digital Signatures. In *Advances in Cryptology EUROCRYPT'1998*, Espoo, Finland, Lecture Notes in Computer Science 1403, pp. 591–606, Springer-Verlag, 1998.
   Eprint 1997/015 https://eprint.iacr.org/1997/015
4. Holger Bürk, Andreas Pfitzmann. Value Exchange Systems Enabling Security and Unobservability. *Computer & Security*, vol. 9 (8), pp. 715–721, 1990. https://doi.org/10.1016/0167-4048(90)90114-9
5. Pierpaolo Della Monica, Ivan Visconti, Andrea Vitaletti, Marco Zecchini. Decentralized Fait Exchange with Advertising. In *Cryptology and Network Security – CANS 2025*, Osaka, Japan, Lecture Notes in Computer Science 16351, pp. 358–381, Springer-Verlag, 2025.
   ArXiv 2503.10411 https://doi.org/10.48550/arXiv.2503.10411
6. Morris Dworkin. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. NIST Special Publication 800-38A. U.S. Department of Commerce, National Institute of Standards and Technology, 2001.
7. Stefan Dziembowski, Lisa Eckey, Sebastian Faust. FairSwap: How to Fairly Exchange Digital Goods. In *25th ACM Conference on Computer and Communications Security*, Toronto, Canada, pp. 967–984, ACM Press, 2018. https://doi.org/10.1145/3243734.3243857
   Eprint 2018/740 https://eprint.iacr.org/2018/740
8. Lisa Eckey, Sebastian Faust, Benjamin Schlosser. OptiSwap: Fast Optimistic Fair Exchange. In *27th ACM Conference on Computer and Communications Security*, virtual, pp. 543–557, ACM Press, 2020. https://doi.org/10.1145/3320269.3384749
   Eprint 2019/1330 https://eprint.iacr.org/2019/1330
9. Shimon Even, Yacov Yacobi. Relations Among Public Key Signature Systems. Technical Report 175. Technion, Haifa, Israel, 1980.
10. Yihao Guo, Minghui Xu, Dongxiao Yu, Yong Yu, Rajiv Ranjan, Xiuzhen Cheng. Cross-Channel: Scalable Off-Chain Channels Supporting Fair and Atomic Cross-Chain Operations. *IEEE Transactions on Computers*, vol. 72 (2023.11), pp. 3231–3244, 2023. https://doi.ieeecomputersociety.org/10.1109/TC.2023.3288765
    ArXiv 2212.07265 https://doi.org/10.48550/arXiv.2212.07265
11. Harry A. Kalodner, Steven Goldfeder, Xiaoqi ChenM S. Matthew Weinberg, Edward W. Felten. Arbitrum: Scalable, private smart contracts. In *USENIX Security Symposium (USENIX Security 2018)*, Baltimore, MD, USA, pp. 1353–1370, USENIX Association, 2018. https://www.usenix.org/conference/usenixsecurity18/presentation/kalodner
12. Majid Khabbazian. Fair Data Exchange at Near-Plaintext Efficiency. To appear in Financial Cryptography and Data Security (FC 2026). 2026.
    ArXiv 2506.14944 https://doi.org/10.48550/arXiv.2506.14944
13. Seungwoo Kim, Semin Han, Seongho Park, Kyeongtae Lee, Jihye Kim, Hyunok Oh. zkMarket: Privacy-preserving Digital Data Trade System via Blockchain. 2024. Eprint 2024/1775 https://eprint.iacr.org/2024/1775
14. Yun Li, Cun Ye, Yuguang Hu, Irving Morpheus, Yu Guo, Chao Zhang, Yupeng Zhang, Zhipeng Sun, Yiwen Lu, Haodi Wang. ZKCPlus: Optimized Fair-Exchange Protocol Supporting Practical and fllexible Data Exchange. In *28th ACM Conference on Computer and Communications Security*, virtual, pp. 3002–3021, ACM Press, 2021. https://doi.org/10.1145/3460120.3484558
15. Xuanming Liu, Jiawen Zhang, Yinghao Wang, Xinpeng Yang, Xiaohu Yang. SmartZKCP: Towards Practical Data Exchange Marketplace Against Active Attacks. Eprint 2024/941 https://eprint.iacr.org/2024/941
16. Joseph Poon, Thaddeus Dryja. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. 2016. https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts
17. Ertem Nusret Tas, István András Seres, Yinuo Zhang, Márk Melczer, Mahimna Kelkar, Joseph Bonneau, Valeria Nikolaenko. Atomic and Fair Data Exchange via Blockchain. In *30th ACM Conference on Computer and Communications Security*, Salt Lake City UT, USA, pp. 32227-3241, ACM Press, 2024. https://dl.acm.org/doi/proceedings/10.1145/3658644
    Eprint 2024/418 https://eprint.iacr.org/2024/418