Sponsored Fair Exchange

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.**

***Index Terms*—Fair exchange, smart contract**

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.

CIMA.Science started a long-term project in which one of the goals is to develop a costless autonomous transaction platform.[1] This would render the smart contract technology more accessible and so favor economic inclusion. Within this frame, 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. 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 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 can also consider storage services where we would pay $\$y$ to retrieve a file x characterized by its hash d.

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, 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 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 discuss more about it in Section 7.2. 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.

The contribution of this paper is twofold. We improve the existing OptiSwap protocol [14] to allow arbitrary de-

1. https://cima.science/

scription 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.

Problem statement. The aim is to enable the fair exchange of a digital item x against a payment $\$y$ between two participants V and B by using a smart contract as a trusted third party. We want security, fairness, low cost, and to accommodate x and desc of any size or structure. In addition to this, we want V and B to have no expense if the exchange fails, and V to have nothing to pay during the process. We rely on extra participants (sponsors) to cover the fees but we want to extend the fairness to them too.

The protocol includes an "optimistic transaction phase", with bounded (low) cost, and a more expensive "dispute resolution phase" which is assumed to be rare (occurring only in the presence of malicious participants). The dispute resolution happens when participants disagree on whether the exchange should complete or cancel. The goal is to reach a consensus on this point: to determine whose claim is fair, which would be the claim of the "winning party" while the other would be the "losing party". Here, fairness implies that an honest party must be winning.

As detailed below, the protocol involves other participants called "Sponsors". We have three types of sponsors: the optimistic transaction sponsor, the dispute sponsor of V, and the dispute sponsor of B. Sponsors could be anyone (including V and B). They all invest on the transaction and pay some fees, with the incentive of possible profits by getting a reward in the end. Compared to the optimistic transaction sponsor, the dispute sponsors take no risk at all: they make profit if they are honest and they lose for sure if they play against an honest participant. Actually, dispute sponsors can be fully automated and make profits at no risk.

We informally define our requirements as follows.

- **Fairness.** Either the protocol completes to the satisfaction of every honest participant, or it is canceled with no loss from the viewpoint of every honest participant.
- **Inclusivity.** The technology must be accessible by everyone with no entrance obstacle. At the moment, we only assume that participants have an account on the underlying blockchain. We do not assume any registration. Furthermore, transaction fees must not be an obstacle, and there must be no financial penalty for participants when the protocol is canceled.
  B may deposit some completion fee tip_o, which is returned in full if the exchange is canceled, and go to the optimistic exchange sponsor if the exchange completes. Beyond that, we assume no fee.
- **Privacy.** Item x should remain private to honest V and B, in the sense that only they may see x if they follow the protocol. It actually remains private to honest V when the exchange is canceled.

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 [16]. To overcome that impossibility, designers added a trusted third party.

At CCS 1997, Franklin and Reiter presented a simple fair exchange protocol [19]. In the $\mathbf{Z}_N$ ring where N has unknown factorization, the protocol allows to exchange the square root of a public value against a square root of another public value by using a trusted third party. This third party is semi-trusted in the sense that they cannot learn anything about the exchanged square roots.

Bürk and Pfitzmann noticed that there can be protocols in which the trusted third party is involved only in rare dispute cases [8]. Later on, the notion of *optimistic* fair exchange was 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. In 1999, Asokan et al. [3] proposed an optimistic fair exchange protocol. In some specific applications like certified emails, the optimistic fair exchange ends up being very simple, as shown by Micali [28].

The impossibility result was revisited. In 1999, Gartner, Pagnia, and Vogt proved that we can build a consensus protocol from fair exchange [29]. Then, they invoke the FLP impossibility result of consensus by Fischer, Lynch, and Paterson [18].

The FLP impossibility result applies in non-synchronized and deterministic distributed system, hence does not hold when we participants are synchronized nor when we introduce randomness in the process. As a matter of fact, consensus was shown to be practically feasible with the blockchain technology. Hence, we can assume an infrastructure making consensus already solved and consider fair exchange again.

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 receive any private information.

One example of elementary smart contract is the "Hash Time-Locked Contract" (HTLC) [30]. 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.

HTLC was used to implement atomic swap. This is a fair exchange of a payment in cryptocurrency against a payment in another cryptocurrency (for instance, bitcoins against ethers). The atomic swap is done using two HTLC: the party holding bitcoins sets an HTLC with the bitcoins and the hash of a random key of their choice. The other party can then set another HTLC with the ethers to be released with a preimage of the same hash. Then, the first party sends the selected key to the second HTLC, and gets the ethers. The other party can use the key in the first HTLC to get the bitcoins.

An HTLC can be replaced using an adaptor signature by a bonding contract where a bonded payment is taken by using a signature of both participants [15]: B pays to a bonding contract and gives a pre-signed payment to V; V converts the pre-signature into a real signature to post it on the bonding contract to get the payment; and the buyer takes the converted signature to extract 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 [23], Cross-Channel [20], Smart ZKCP [24], Fair Data Exchange (FDE) [33], zkMarket [26].

As a zero-knowledge proof that an encrypted item satisfies the expected description is expensive, the protocols FairSwap [13] and OptiSwap [14] push the optimistic approach further by starting the optimistic part with no proof and keeping the proof (not 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 which describes the item x to be sold.

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.

The SmartJudge architecture by Wagner et al. [35] generalizes the structure of FairSwap and OptiSwap with the notion of *mediator* (the optimistic phase) and of *verifier* (the dispute solver) as separate smart contracts. It extends to applications beyond fair exchange.

In blockchains such as Arbitrum [25], 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. The newly implemented EIP-7702 seems to reduce the problem by allowing a user to post transactions which can act as ephemeral smart contracts the gas of which being sponsored by another smart contract. This corresponds to our setting assumption: that a sponsor can prepay some deposit from which transaction fees will be taken.

Nearly all fair exchanges based on blockchains are vulnerable against temporal censorship [34]. This is an attack by which a malicious party would bribe the miner to delay the actions of the other party until a timeout is reached. For instance, in the case of HTLC, the malicious buyer could delay the action of the honest vendor when sending the hash

preimage to the HTLC, until the buyer can get their money back after timeout.

Rapidash [9] was proposed to defeat this attack. It works by essentially allowing an option by which the deposits would be lost forever. More precisely, the *intention* to cancel in an HTLC must be done by submitting the preimage of a specific cancel hash. Cancelation does not happen immediately but after some period of time, unless someone submits the two hash preimages at the same time in order to get a small reward and to make the rest of the deposit to be burned. This new termination state is called a *bomb*. The idea is that an honest miner who saw that the vendor tried to submit their preimage but was censored and who saw the intention of the buyer to get refunded should activate this bomb option, while the strategy of the buyer and of the malicious miner to continue to censor the vendor during the delay is most likely to fail. Ending on the bomb case is essentially unfair, as the vendor reveals the hash preimage and is not paid in this case. Somehow, allowing censorship degrades the notion of fairness. To prevent the miner-buyer coalition from preferring burning the money instead of paying the vendor, Rapidash assumes that the adversary is *rational* and some additional *collateral* deposit is made by the buyer and is always refunded, except in the bomb case. Hence, preferring to lose the payment and the collateral instead of just paying the vendor is non-rational. Rapidash offers security against a rational coalition which may include miners. This is the so called *CSP-fairness*. However, it increases the amount of seed money which is needed by the buyer and is not secure against rational adversaries receiving some external incentives. For instance, a coalition willing to spend a lot of money for the purpose of ruining a specific vendor by getting all their items without paying them can do that. As another example, a coalition could have bet on the exchange being unfair to the vendor.

FileBounty [22] proposes to exchange a file x with iterated hash d with a price proportional to the number of correct blocks received, which is yet another model. The dispute involves a single block hash on the smart contract but is done through a zkSNARK for privacy. It also requires the blocks to be verified in reverse order so that they can be challenged on the fly.

He et al. [21] propose another protocol where they specifically separate the role of the seller and of the content delivery system. Hence, the exchange is between three participants and the notion of fairness is more complex.

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 as we discuss in Section 7.2. Contrarily, the fees in a dispute are much higher but sponsoring is riskless. The dispute resolution sponsors are guaranteed to get a reward tip_d if they verify well the dispute arguments before engaging. Hence, dispute resolution happens only in the presence of malicious participants. Consequently, disagreements would normally resolve offchain without running a dispute, hence without fees.

Unfairness could remain if an honest party does not have the seed money to sponsor themselves (at no risk) and would not succeed to find a sponsor for dispute resolution. 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.

Second contribution. Besides, we bring improvements to the OptiSwap protocol. In OptiSwap, the dispute resolution solver follows the structure of a hardcoded balanced circuit which encodes the Merkle hash computation on x. 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. 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 includes the circuit of desc), V encrypts x into ct and 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

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 $\|x\| = 128$KB: V:89sec, B:34sec	n/a
OptiSwap	optimistic	Merkle-hash	952 939 gas (deploy) + 101 307 gas (run)	1 962 992 gas (deploy) for $\|x\| = 1$GB: 6 412 569 gas (run)
SOX	optimistic	SHA256	1 463 686 gas (deploy) + 219 540 gas (run) for $\|x\| = 4$MB: V:783ms, B:601ms	2 356 567 gas (deploy + steps 5–6) for $\|x\| = 4$MB: 7 039 703 gas (run) V:14.531sec, B:12.828sec

when they are right and that a participant who is right cannot lose in the dispute. The goal of the dispute is to make the smart contract decide without having to process or look at the x material who is right and should get the payment.

We can compare our SOX protocol with OptiSwap [14] (another optimistic protocol) and FDE [33] (a protocol in the ZKCP family) in Table 1. They all offer strong security. Only SOX is adapted to a standard SHA256 hash. FDE uses the KZG polynomial commitment while OptiSwap uses Merkle hash. By looking at the three protocols (limited to the optimistic phase for OptiSwap and SOX), the gas cost and privacy guarantee are pretty similar. FDE requires lots of offchain computation in all cases (it would require 3 days of computation for the buyer to sell a 1GB file), while 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.[2] Our implementation indicates that SOX and OptiSwap are comparable when the circuit is balanced, and SOX is much better in other cases.[3] The ratio between the costs of OptiSwap and SOX would grow as fast as $\frac{m}{\log_2 m}$ for x of size m. For applications where B wants to retrieve a file x by only knowing its SHA256 hash, this is a problem. In general, for applications where x has a complex description function desc, we cannot expect desc to be always programmed into a balanced circuit.

As for SmartJudge [35], we believe that our dispute resolution could find applications beyond fair exchange.

Structure of this paper. We start in Section 2 with the definition of accumulators, which we use as key building blocks. Then, Section 3 formally defines the input/output interface of our protocol and the security notions. Our protocol is specified in Section 4 and our formal result is stated and proven in Section 5 and Appendix. We discuss about implementation in Section 6 and the limitations of our approach in Section 7. Some variants are proposed in Appendix. Notations are summarized in Table 2.

2. Incremental Accumulator

Accumulator. We use a static accumulator Acc/Prove/Verify. The algorithm $h = \mathsf{Acc}(\mathsf{val})$ takes a vector val of any length ℓ as input and returns a hash. The algorithm $\pi = \mathsf{Prove}(\mathsf{val}, (i_1, \ldots, i_a))$ returns a proof for the values $\mathsf{val}(i_1), \ldots, \mathsf{val}(i_a)$. (By convention, $\pi = \emptyset$ if $a < 1$.) The algorithm $\mathsf{Verify}(h, \ell, (i_1, \ldots, i_a), (v_1, \ldots, v_a), \pi)$ verifies that the proof is correct and that each v_j is the correct value $v_j = \mathsf{val}(i_j)$. It is a multiproof as it proves a values at the same time. Correctness implies that for any vector val and any $(i_1, \ldots, i_a)$, if $h = \mathsf{Acc}(\mathsf{val})$ and $\pi = \mathsf{Prove}(\mathsf{val}, (i_1, \ldots, i_a))$, then

$$\mathsf{Verify}(h, \ell, (i_1, \ldots, i_a), (\mathsf{val}(i_1), \ldots, \mathsf{val}(i_a)), \pi) = \mathsf{true}$$

We assume a setup of an (implicit) public key to make collision-resistance possible.

The accumulator must be collision-resistant, i.e. such that for any PPT adversary $\mathcal{A}$, $\Pr[\mathsf{CR} \to 1]$ is negligible with the following game:

Game CR:
1: $\mathcal{A} \to ((v_1, \ldots, v_\ell), (v'_1, \ldots, v'_a), \pi, (i_1, \ldots, i_a), j)$
2: **if** $i_j > \ell$ or $v_{i_j} = v'_j$ **then return** 0
3: $h \leftarrow \mathsf{Acc}(v_1, \ldots, v_\ell)$
4: **if** $\neg\mathsf{Verify}(h, \ell, (i_1, \ldots, i_a), (v'_1, \ldots, v'_a), \pi)$ **then return** 0
5: **return** 1

TABLE 2. NOTATIONS

V, B, S	vendor, buyer, optimistic sponsor
SV, SB	dispute resolution sponsor of vendor and buyer
ID_P	blockchain identity of a participant P
desc	description function
desc_d	description predicate
x	digital item to sell
y	price for item x
k	symmetric encryption/decryption key
Δ	balance of a participant during the protocol
p	probability that the exchange completes
$\bar{\Delta}$, $\bar{p}$	estimates for Δ and p
θ	target average benefit for S
tip_o	tip to reward S
tip_d	tip to reward the winning dispute resolution sponsor
fee_o	maximum blockchain fee during the optimistic phase
fee_d	maximum blockchain fee during the dispute resolution
contract	committing parameters for the agreed contract between V, B, S
circuit[valid]	circuit implementing the valid predicate
argue	proposed argument to find a dispute resolution sponsor
$h\|\mathsf{rnd}$	concatenation of h and rnd
$\|x\|$	length of x
$\|\mathsf{desc}_d\|$	size of the circuit defining desc_d
1_C	1 if $C = \mathsf{true}$ and 0 if $C = \mathsf{false}$
$\perp$	error symbol

2. With the exception of the honest-V malicious-B case which would end in cancel as we do not want B to learn any information about x in this case. However, this case cannot occur in our protocol (Th. 5).

3. The dispute solver deployment is a bit more expensive because we accommodate any desc function.

Essentially, $\mathcal{A}$ is building an accumulator $h = \mathsf{Acc}(v_1, \ldots, v_\ell)$ and proves some values $(v'_1, \ldots, v'_a)$ at indices $(i_1, \ldots, i_a)$ with a divergent one $v_{i_j} \neq v'_j$.

Note that this notion is weaker than vector commitment as it is not binding in the vector elements. The CR security implies that the commitment h is honestly constructed (in Line 3 of the CR game) from a vector. However, a malicious construction could insert two different values at the same position and remain CR-secure.

Incremental accumulator. We use an incremental accumulator. This is a static accumulator Acc/Prove/Verify with additional algorithms ProveExt/VerifyExt. With $\rho = \mathsf{ProveExt}(v_1, \ldots, v_i)$, $w_{i-1} = \mathsf{Acc}(v_1, \ldots, v_{i-1})$, and $w_i = \mathsf{Acc}(v_1, \ldots, v_i)$, having the algorithm $\mathsf{VerifyExt}(i, w_{i-1}, w_i, v_i, \rho)$ to return true proves that adding v_i in the accumulator w_{i-1} leads to the accumulator w_i. In addition to the properties of the static accumulator, correctness implies that this property is always satisfied when ρ, w_{i-1}, and w_i are computed this way.

By convention, we write $\mathsf{Acc}() = \emptyset$ to address the $i = 1$ case.

The accumulator must be CR-resistant and also CRext-resistant, i.e. such that for any PPT adversary $\mathcal{A}$, $\Pr[\mathsf{CRext} \to 1]$ is negligible with the following game:

Game CRext:

1: $\mathcal{A} \to (i, w_{i-1}, w_i, w'_i, v_i, \rho, \rho')$
2: **if** $w_i = w'_i$ **then return** 0
3: **if**$\neg\mathsf{VerifyExt}(i, w_{i-1}, w_i, v_i, \rho)$ **then return** 0
4: **if**$\neg\mathsf{VerifyExt}(i, w_{i-1}, w'_i, v_i, \rho')$ **then return** 0
5: **return** 1

Essentially, the adversary shows that updating an accumulator of hash w_{i-1} with value v_i results in two different accumulator hashes w_i and w'_i. This property is always satisfied whenever Acc builds w_i as a deterministic function of w_{i-1} and v_i, i.e. by *incrementally* adding v_i.

As an incremental accumulator, we can use a Merkle tree [27] as detailed in Appendix B.2. The length of h is constant. The length of π is logarithmic in the size of val, as well as the complexity of Verify. The complexity of Acc and Prove are linear in the size of val. The Verify algorithm will be run in a smart contract, making it of low complexity is important.

In Appendix B.3, we construct another accumulator based on RSA [6], [7] with larger complexity but constant-size proofs.

3. Formal Definition

We define our protocol frame as follows.

Definition 1 (SOX). A sponsored optimistic exchange (SOX) protocol is an interactive protocol between a buyer V, a vendor B, a sponsor S, two dispute sponsors SV and SB, and a blockchain. The blockchain manages assets of participants, which are identified by a public key $\mathsf{ID_V}$, $\mathsf{ID_B}$, $\mathsf{ID_S}$, $\mathsf{ID_{SV}}$, $\mathsf{ID_{SB}}$. Each participant has input and output to the protocol, as well as an account balance which represents what the participant earns during the protocol. We also use some seed money at the beginning of the protocol. We use the following input/output notations for honest participants:

participant	V	B
input	x, y, desc_d	y, desc_d
output		x'

Here, desc_d is a predicate. We assume that V and B agreed on y and desc_d prior to the protocol and that $\mathsf{desc}_d(x)$ is true (in the honest-V case). Some other protocol parameters are predetermined: tip_o, tip_d, fee_o, and fee_d. Participants start with a deposit and terminate with a balance as follows:

participant	V	B	S	SV / SB
deposit	0	$y + \mathsf{tip}_o$	fee_o	$\mathsf{fee}_d + \mathsf{tip}_d$
balance	Δ_V	Δ_B	Δ_S	Δ_SV / Δ_SB

The protocol has two phases: an "optimistic phase" and a "dispute resolution". Dispute resolution may be triggered during the optimistic resolution phase and the seed money for the dispute resolution is only needed when we start the dispute resolution phase. The fees paid to the blockchain are denoted Δ_o and Δ_d for the optimistic and dispute phases, respectively. If there is no dispute resolution, then $\Delta_\mathsf{SV} = \Delta_\mathsf{SB} = \Delta_d = 0$. The protocol is a zero-sum game in the sense that $\Delta_\mathsf{V} + \Delta_\mathsf{B} + \Delta_\mathsf{S} + \Delta_o + \Delta_\mathsf{SV} + \Delta_\mathsf{SB} + \Delta_d = 0$. The paid fees for the blockchain operations are covered by deposits fee_o and fee_d such that $0 \leq \Delta_o \leq \mathsf{fee}_o$ and $0 \leq \Delta_d \leq \mathsf{fee}_d$. We define some properties of the protocol below.

- **Termination**. Every honest participant eventually terminates the protocol. It ends with a state "complete" or "cancel".
- **Correctness**. The protocol is *correct* in the sense that if V, B, S are honest, then their final states are "complete" and we have $x' = x$ and $\Delta_\mathsf{V} = +y$.
- **Optimism**. The protocol is *optimistic* in the sense that if V and B are honest, then there is no dispute.
- **Riskless**. The protocol is *riskless* for honest participants (except S) in the sense that there is a guaranteed balance for any honest V, B, SV, and SB, as detailed below.

	Δ_V	Δ_B
Cancel	0	0
Complete	y	$-y - \mathsf{tip}_o$

  When there is a dispute resolution, we have $\Delta \geq \mathsf{tip}_d$ except with negligible probability for an honest dispute resolution sponsor.
  A risk remains only for an honest S: the risk is to lose up to fee_o but the hope is to earn at least $\mathsf{tip}_o - \mathsf{fee}_o$.
- **Disincentivized disputes**. The decision to join for SV and SB is taken from information provided by V and B. If there is a dispute resolution, we have $\max(\Delta_\mathsf{SV}, \Delta_\mathsf{SB}) \geq \mathsf{tip}_d$ and $\min(\Delta_\mathsf{SV}, \Delta_\mathsf{SB}) \leq -\Delta_d - \mathsf{tip}_d$.

(Note that this implies that honest SV and SB never both decide to join the dispute resolution.)

Adversarial model. Our adversarial model is formalized by the oracles in Fig. 1. We have a set of participants P which includes a special participant $\mathsf{P} = \mathsf{blockchain}$. The adversary can take control of any subset of participants (who are then called malicious) but blockchain which remains honest by design. In particular, sponsors may be corrupted. The set of non-corrupted participants is denoted by Honest.

Communication between honest participants is private, but the adversary learns their sizes. A message m from P to Q is inserted into a mailbox $\mathsf{inbox}_\mathsf{Q}$ as a pair (P, m) when being sent by P, or by the adversary through the OSend oracle.

Communication to the blockchain is also done through a mailbox $\mathsf{inbox}_\mathsf{blockchain}$, which we can see as being a mempool. However, this communication is not private and always leaks to the adversary. Running the blockchain means flushing all waiting messaged and posting it as a new block. The adversary controls when the blockchain is run by the Oblockchain oracle, but must respect that every honest participant has seen the last block before making a new one. This is done by using the flags $\mathsf{hold\_on}_\mathsf{P}$ which indicates that P has not seen it yet. This makes sure that no participant can miss a deadline.

An honest participant P is associated to an algorithm that we also denote by P. It takes as input a non-final state $\mathsf{state}_\mathsf{P}$ (i.e. $\mathsf{state}_\mathsf{P} \notin \{\mathsf{Cancel}, \mathsf{Complete}\}$), the set of incoming messages $\mathsf{inbox}_\mathsf{P}$, and the last block inserted in the blockchain. It gives as an output the new state $\mathsf{state}_\mathsf{P}$, the set msgs of messages that P wants to send, and the final output o_P if the state is final, i.e. $\mathsf{state}_\mathsf{P} \in \{\mathsf{Cancel}, \mathsf{Complete}\}$. The adversary also controls when each participant is run and learns what leaks from msgs: the destinators, the size of each message, and the messages sent to a malicious participant or to the blockchain. Note that messages from P to the blockchain cannot be intercepted and are ensured to appear in the next block in our model.

Definition 2 (Consensus). A SOX has consensus if the final state of every honest participant is the same. More precisely, for any PPT adversary $\mathcal{A}$ and any $(\mathsf{Honest}, x, \mathsf{desc}_d, y)$, the game $\mathsf{Consensus}(\mathsf{Honest}, x, \mathsf{desc}_d, y)$ in Fig. 1 returns 1 with negligible probability.

Privacy. Ideally, we want to keep x and desc_d private. If V and B are honest, we want it to remain private in any case. Clearly, we cannot keep private whatever a malicious participant receives. Hence, if V is malicious, x and desc_d cannot remain private. If B is malicious, desc_d cannot be protected, and x can neither remain private when the protocol completes. In other cases, we do not want desc_d and x to leak. Specifically, we want x to remain private if the exchange is canceled. We formalize below the notion of privacy for x.

As for the price y, we decide not to consider it private for two reasons. First of all, it really depends on the blockchain if the transaction price can be maintained private or not. Second, S may want to set tip_o as a percentage of the price, so S needs to know.

In the privacy games in Fig. 1, we formalize what leaks by default about x and desc_d with the leak subroutine. Essentially, if V is malicious, everything leaks. Otherwise, the size of x leaks and the leakage about desc_d depends on B. If B is malicious, desc_d leaks. Otherwise, only the size of its circuit leaks.

Definition 3 (Privacy). Consider a SOX with an honest V. The SOX enforces *privacy* if for any PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that for any (x, desc_d, y), $\mathcal{S}(\mathsf{Honest}, \mathsf{leak}(x, \mathsf{desc}_d), y)$ and the game $\mathsf{Privacy}(\mathsf{Honest}, x, \mathsf{desc}_d, y)$ in Fig. 1 produce computationally indistinguishable outputs in the two following cases:
Honest-B privacy: $\mathsf{Honest} = \{\mathsf{V}, \mathsf{B}\}$.
Malicious-B privacy: $\mathsf{Honest} = \{\mathsf{V}\}$.

Clearly, if V terminates with state Complete, it implies that $\Delta_\mathsf{V} = y$, so V was paid to disclose x to B. If B is malicious, this means that x can leak. In that case, $\mathcal{A}$ may learn x as a feature, not a leakage.

Fairness. We want that for an honest V, we have (1) $\Delta_V = y$ in the Complete case, (2) x is not disclosed in the Cancel case, and for an honest B, (3) $\mathsf{desc}(x')$ is true in the Complete case and (4) $\Delta_\mathsf{B} = 0$ in the Cancel case. Conditions (1) and (4) are achieved in Def. 1 (riskless condition) and (2) is achieved in Def. 3 (malicious-B case). To complete the notion of fairness, what we only need is the condition (3).

Definition 4 (Fairness). Consider a SOX which enforces consensus and privacy. The SOX enforces *fairness* if it enforces that when B is honest and terminates in a Complete state, then $\mathsf{desc}_d(x')$ is true. More precisely, for any PPT adversary $\mathcal{A}$ and any $(\mathsf{Honest}, x, \mathsf{desc}_d, y)$, the game $\mathsf{Fairness}(\mathsf{Honest}, x, \mathsf{desc}_d, y)$ in Fig. 1 returns 1 with negligible probability.

to summarize, SOX has two possible termination states. In Complete, honest B pays $\$(y + \mathsf{tip}_o)$ and receives x' satisfying $\mathsf{desc}_d(x')$, while honest V receives $\$y$. In Cancel, honest B pays nothing, while honest V receives no payment and leaks no information about x (except the bit $\mathsf{desc}_d(x)$, the length $|x|$, and desc_d).

Adjusting the parameters. To make the SOX protocol work, we must have

1) $\mathsf{fee}_o \geq \Delta_o$ and $\mathsf{fee}_d \geq \Delta_d$ so the the smart contract does not run out of money;
2) $\mathsf{tip}_o > \Delta_o$ and $\mathsf{tip}_d > 0$ to maintain an incentive for sponsors;
3) Δ_o as low as possible since it is the worst case loss of S.

Game Privacy(Honest, x, desc_d, y):
1: init
2: $\mathcal{A}^{\mathsf{oracles}}(\mathsf{leak}(x, \mathsf{desc}_d), y) \to z$
3: **if** B $\notin$ Honest $\wedge$ $\mathsf{state}_\mathsf{V} \neq$ Cancel **then return** 0
4: **return** z

Game Fairness(Honest, x, desc_d, y):
5: **if** B $\notin$ Honest **then return** 0
6: init
7: $\mathcal{A}^{\mathsf{oracles}}(\mathsf{leak}(x, \mathsf{desc}_d), y)$
8: **if** $\mathsf{state}_\mathsf{B} \neq$ Complete $\vee$ $\mathsf{desc}_d(o_\mathsf{B})$ **then return** 0
9: **return** 1

Game Consensus(Honest, x, desc_d, y):
10: init
11: $\mathcal{A}^{\mathsf{oracles}}(\mathsf{leak}(x, \mathsf{desc}_d), y)$
12: final_states $\leftarrow \{\mathsf{state}_\mathsf{P}; \mathsf{P} \in \mathsf{Honest}\}$
13: **if** {Complete, Cancel} $\subseteq$ final_states **then return** 1
14: **return** 0

Subroutine init
15: $\mathsf{state}_\mathsf{V} \leftarrow (x, \mathsf{desc}_d, y)$
16: $\mathsf{state}_\mathsf{B} \leftarrow (\mathsf{desc}_d, y)$
17: clear all $\mathsf{inbox}_\mathsf{P}$ and last_block
18: lower all $\mathsf{hold\_on}_\mathsf{P}$
19: **return**

Subroutine leak
20: **if** V $\notin$ Honest **then return** (x, desc_d)
21: **if** B $\notin$ Honest **then return** $(|x|, |\mathsf{desc}_d|)$
22: **return** $(|x|, |\mathsf{desc}_d|)$

Oracle OSend(P, Q, m) ▷ malicious P sends m to Q
23: **if** P $\notin$ Honest **then** add (P, m) in $\mathsf{inbox}_\mathsf{Q}$
24: **return**

Oracle OP: ▷ run a honest P, P $\neq$ blockchain
25: lower $\mathsf{hold\_on}_P$
26: $\mathsf{P}(\mathsf{state}_\mathsf{P}, \mathsf{inbox}_\mathsf{P}, \mathsf{last\_block}) \to (\mathsf{state}_\mathsf{P}, \mathsf{msgs}, o_\mathsf{P})$
27: clear $\mathsf{inbox}_\mathsf{P}$ and out
28: **for** each $(\mathsf{Q}, m) \in$ msgs **do**
29: add (P, m) in $\mathsf{inbox}_\mathsf{Q}$
30: **if** Q $\in$ Honest $\wedge$ Q $\neq$ blockchain **then**
31: add $(\mathsf{Q}, |m|)$ in out ▷ private message m
32: **else**
33: add (Q, m) in out
34: **end if**
35: **end for**
36: **return** out

Oracle Oblockchain ▷ post a new block
37: **if** there exists P st $\mathsf{hold\_on}_\mathsf{P}$ is raised **then return**
38: last_block $\leftarrow \mathsf{inbox}_{\mathsf{blockchain}}$ ▷ consensus on last_block
39: clear $\mathsf{inbox}_{\mathsf{blockchain}}$
40: **for** each P $\in$ Honest, raise $\mathsf{hold\_on}_P$
41: **return** last_block

Figure 1. Privacy and Fairness Games

4. 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 $\mathsf{desc}_d(x) \iff \mathsf{desc}(x) = d$.

The complete pseudocode of the SOX protocol (after the premilinary phase) is provided in Fig. 3.

4.1. 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). 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;
- the description predicate desc_d of x (the input x must be such that $\mathsf{desc}_y(x)$ 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 version reference.

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 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 version. Hence, the compiler must be compatible with version.

The choice of algorithms includes a one-time symmetric encryption scheme Enc/Dec, a commitment scheme Com/Open, an accumulator

Acc/Prove/Verify, and also an incremental accumulator Acc/Prove/Verify/ProveExt/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

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

The variable k represents a symmetric key while ct represents a ciphertext. The predicate valid is "compiled" into a circuit circuit[valid]. We assume that the agreed desc_d, m, n, params, and version in the preliminary phase are enough for V and B to build circuit[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 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 $\mathsf{circuit[valid]} = G = (g_1, \ldots, g_n)$ is put in an accumulator. In other words, circuit[valid] is considered as the input vector of Acc with coordinates g_i. Thus, we compute $h_{\mathsf{circuit}} = \mathsf{Acc}(\mathsf{circuit[valid]})$. For each gate g_i, it is possible to compute $\pi = \mathsf{Prove}(\mathsf{circuit[valid]}, i)$ 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 version number. The smart contract must be compatible with version.

Example: media as hash preimage. To take an example, we assume that x is a media file which is characterized by its SHA256 hash $d = \mathsf{SHA256}(x)$ [1]. A typical Blu-ray disk would give x the length of 25GB. If x was one song, it would typically have 4MB. If we split ct in blocks of 512 bits, we obtain $m = 2^{29}$ blocks for Blu-ray or $m = 2^{16}$ blocks for a song. We easily define a circuit using $n = 2m + 1$ gates using the compression function Comp which defines SHA256. We use one-time AES-CTR encryption [12]. For this, we assume that the instruction set defines a gate $\oplus\mathsf{AES}_{k,i_1,i_2,i_3,i_4}$ with one son. It uses the parameters $i_1, \ldots, i_4$ and the input variable k. It evaluates to $u \oplus (\mathsf{AES}_k(i_1) \| \cdots \| \mathsf{AES}_k(i_4))$ when the son evaluates to u. We assume another gate Comp with one or two sons. With only one son evaluating to u, it evaluates to $\mathsf{Comp}(u, \mathsf{IV})$ with IV defined by SHA256. With two sons evaluating to u and v, it evaluates to $\mathsf{Comp}(u, v)$. Finally, we assume a gate Cmp_d with one son which evaluates to true if and only if $u = d$ when the son evaluates to u. The circuit circuit[valid] is as follows.

- m gates of form $(\oplus\mathsf{AES}_{k,4i-3,4i-2,4i-1,4i}, -i)$ to decrypt ct_i;
- 1 gate of form $(\mathsf{Comp}, 1)$ to start compressing x_1;
- $m-1$ gates of form $(\mathsf{Comp}, i, m+i-1)$ to iterate the SHA256 compression with x_i;
- 1 gate of form $(\mathsf{Cmp}_d, 2m)$ to compare the SHA256 hash with d.

For the Blu-ray with $m = 2^{29}$, a proof of membership in the Merkle tree accumulator would consist of 31 hashes. For the song with $m = 2^{16}$, a proof would consist of 18 hashes.

Note that the depth of the circuit is $2 + m$, which is linear in the size $n = 2m+1$. OptiSwap would be inefficient here, with a worst case complexity of $\mathcal{O}(m)$ compared to our $\mathcal{O}(\log m)$. In their paper, the authors of OptiSwap take as an example a media with a hash d obtained by a Merkle tree instead of the standard SHA256 [14]. The Merkle hash defines a circuit of logarithmic depth, contrarily to SHA256. This is how they obtain good performances. Instead, we use standard hash, but Merkle trees in accumulators to prove the correct/incorrect evaluation of a circuit no matter the structure of the circuit defined by the use-case.

4.2. Optimistic Phase

In this phase, V and B have as common input $\mathsf{ID_V}$, $\mathsf{ID_B}$, y, tip_o, tip_d, desc_d, m, n, circuit[valid], params, version. The vendor V has a private input x and the buyer B has an asset $\$(y + \mathsf{tip}_o)$.

Engagement phase. In an engagement phase, V prepares the committing information (a manifest that we denote by contract) to be inserted in the smart contract as follows. There, the V interface needs to process x.

1) V computes $h_{\mathsf{circuit}} = \mathsf{Acc}(\mathsf{circuit[valid]})$ using the static accumulator.
2) V picks a random symmetric key k.
3) V computes $\mathsf{ct} = \mathsf{Enc}_k(x)$, the encryption of x under key k, using the one-time symmetric encryption function Enc.
4) V computes $h_{\mathsf{ct}} = \mathsf{Acc}(\mathsf{ct}_1, \ldots, \mathsf{ct}_m)$ using the static accumulator Acc.
5) V computes $(c, o) \leftarrow \mathsf{Com}(h_{\mathsf{circuit}}, h_{\mathsf{ct}})$ using the cryptographic commitment scheme Com, getting the commit value c and the opening value o.
6) V computes contract as

   $\mathsf{contract} =$
   $(\mathsf{ID_V}, \mathsf{ID_B}, y, m, n, c, \mathsf{tip}_o, \mathsf{tip}_d, \mathsf{params}, \mathsf{version})$
7) V sends (through the secure channel) ct, contract, o to B.
8) B computes h_{circuit} and h_{ct} like V (in Step 1 and 4 above).
9) B takes c from contract.

10) B verifies $(h_{\mathsf{circuit}}, h_{\mathsf{ct}}) = \mathsf{Open}(c, o)$ when opening the commitment c.
11) B verifies that elements in contract are correct / as agreed.
12) B sends contract to S who decides to sponsor or not.

Note that desc_d is encoded in $\mathsf{circuit}[\mathsf{valid}]$ which is committed by c as h_{circuit} in contract. Similarly, ct is committed by c. The reference to c will be useful in the case of a dispute.

Smart contract execution. This phase has a bounded $\mathcal{O}(1)$ gas complexity. The sponsor S covers the bounded cost of this phase (but hopes to collect $\$\mathsf{tip}_o$ if the transaction completes). We let fee_o denote the maximum blockchain fee to be covered, which is what S "invests". In this phase, the optimistic transaction sponsor S launches a smart contract which is described below.

During its execution as explained below, the smart contract will collect payments by different participants. It collects the deposit from the sponsor S which is used to cover the blockchain fees until a dispute phase. It also collects deposits from "dispute sponsors" which are pooled into a "dispute deposit". Finally, it collects $\$y$ and $\$\mathsf{tip}_o$ by B. Hence, the smart contract maintains the following balances:

- S-deposit: the deposit from S (originally, fee_o);
- B-payment: the deposit from B (originally, $y+\mathsf{tip}_o$);
- dispute deposit: the pooled deposits from SV and SB ($\mathsf{fee}_d + \mathsf{tip}_d$ from each sponsor when they join the protocol).

The smart contract defines exactly what input to expect from which participant and the blockchain fees for each correct input are covered by the S-deposit.[4]

The smart contract can terminate by two manners.

- Cancel: Return the B-payment to B. Return the leftover S-deposit to S. If applicable, return the leftover dispute deposit to SB.
- Complete: On the B-payment, pay $\$y$ to V and $\$\mathsf{tip}_o$ to S. Return the leftover S-deposit to S. If applicable, return the leftover dispute deposit to SV.

The honest V and B terminate in the same state.

The smart contract runs as illustrated on Fig. 2. We specify it below in straight fonts and add comments in italic font.

Step 1. S sets up a smart contract, compatible with version, with input contract, and deposits some money $\$\mathsf{fee}_o$. The smart contract sets deadlines to complete Step 2, 3, and 4. S tells V and B that the smart contract is ready.
V and B verify that smart contract is correct and that contract is as agreed during engagement.
In the next steps, blockchain fees for moves of the smart contract are taken on the deposit of $\$\mathsf{fee}_o$ *from* S.

4. If the blockchain does not allow that, the smart contract can simply refund the participant for their fees as soon as the input is there.

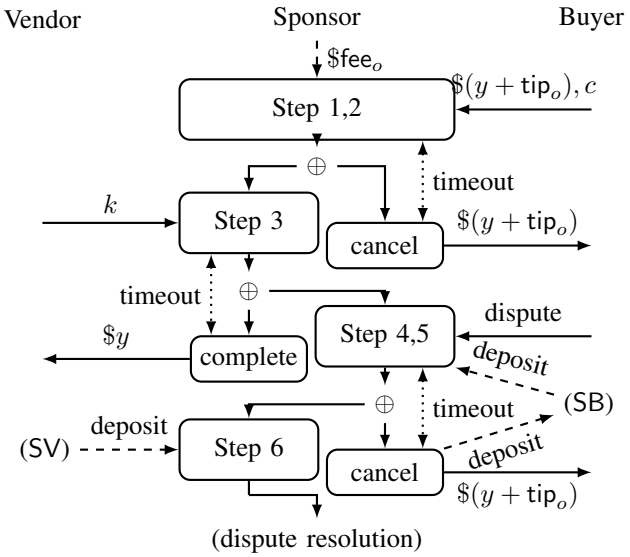

Figure 2. Evolution of the smart contract during the optimistic Phase. Dashed arrows are to or from a sponsor. Dotted arrows represent a time interval after which the contract terminates if nothing happened. The $\oplus$ symbol denotes an exclusive branch in two possible cases. The complete termination returns tip_o to the main sponsor. All termination cases return the unspent deposit from the main sponsor.

Step 1′. (Following Step 1 but after the deadline for Step 2.) Upon activation by anyone, the smart contract terminates with Cancel as B did not follow up.

Step 2. (Following Step 1.) Receive a payment by $\$(y+\mathsf{tip}_o)$ by B. Nothing else is considered.

Step 2′. (Following Step 2 but after the deadline for Step 3.) Upon activation by anyone, the smart contract terminates with Cancel as V did not follow up.

Step 3. (Following Step 2.) Receive a key k from V. No other message is considered.
At the deadline for Step 3 (or before if this step is done earlier), B takes k and verifies that $\mathsf{Dec}_k(\mathsf{ct}) = x'$ is such that $\mathsf{desc}_d(x')$ is true. If it is the case, B terminates with output x'. If it is not the case, B must jump to Step 4 before the deadline.

Step 3′. (Following Step 3 but after the deadline for Step 4.) Upon activation by anyone, the smart contract terminates with Complete as B implicitly accepted k.

Step 4. (Following Step 3.) Receive from B a trigger for dispute.
The smart contract gives deadlines to complete Step 5 and 6.
B and V prepare their arguments and look for dispute sponsors.

Step 4′. (Following Step 4 but after the deadline for Step 5.) Upon activation by anyone, the smart contract terminates with Complete as B did not find a sponsor SB to support the dispute.

Step 5. (Following Step 4.) Receive from a new participant SB a deposit $\mathsf{fee}_d + \mathsf{tip}_d$.
Before that, SB *is supposed to be convinced that*

the argument by B *is valid and supporting* B *(as explained in the next paragraph).*

Step $5'$. (Following Step 5 but after the deadline for Step 6.) Upon activation by anyone, the smart contract terminates with Cancel as V did not find a sponsor SV to support the dispute.

Step 6. (Following Step 5.) Receive from a new participant SV a deposit $\mathsf{fee}_d + \mathsf{tip}_d$.
Before that, SV *is supposed to be convinced that the argument by* V *is valid and supporting* V *(as explained in the next paragraph).*

After that, the smart contract proceeds with the dispute resolution. The blockchain fees of all those 6 steps are covered by S. As we can see, the smart contract had nearly nothing to do so far, except collecting payments, storing information, and keeping up on step transitions. Hence, gas fees are very low and bounded by a constant.

Call for dispute sponsor phase. Upon triggering the dispute resolution in Step 4, V and B must make their argument and use it to look for a sponsor. An argument consists of the entire circuit $\mathsf{circuit}[\mathsf{valid}]$ and the ciphertext ct which are committed by c in contract. We let

$$\mathsf{argue} = (\mathsf{circuit}[\mathsf{valid}], \mathsf{ct}, o)$$

We consider argue relative to contract and k which are in the smart contract. We denote $(h_{\mathsf{circuit}}, h_{\mathsf{ct}}) = \mathsf{Open}(c, o)$ where c is defined in contract.

- We say that argue relative to contract is *valid* if it is the case that $\mathsf{Acc}(\mathsf{ct}) = h_{\mathsf{ct}}$ and $\mathsf{Acc}(\mathsf{circuit}[\mathsf{valid}]) = h_{\mathsf{circuit}}$.
- We say that argue relative to contract and k *supports* B if $\mathsf{desc}_d(\mathsf{Dec}_k(\mathsf{ct}))$ is false, and that it *supports* V if it is true.

Clearly, any verifier can easily verify these properties by reading information in the smart contract and in argue.

By construction, there can be only one valid argument relative to contract unless the commitment scheme or the static accumulator is broken. This valid argument clearly shows which party should be supported in the dispute. A honest dispute sponsor accepts to sponsor a party if and only if the party presents a valid argument which is supporting themselves and the smart contract is currently waiting for the dispute sponsor.

Later on, given an argument argue, we define the circuit evaluation map val. This is a vector of n values, mapping each gate index to a value. We define it by induction, by

$$\forall i > 0 \quad \mathsf{val}(i) = \mathsf{op}(k, \mathsf{val}(s_1), \ldots, \mathsf{val}(s_a)) \qquad (1)$$

where $g_i = (\mathsf{op}, s_1, \ldots, s_a)$ is the ith gate in $\mathsf{circuit}[\mathsf{valid}]$. For convenience, we denote $\mathsf{val}(-i) = \mathsf{ct}_i$ for $i = 1, \ldots, m$. Note that $\mathsf{val}(n) = \mathsf{desc}_d(\mathsf{Dec}_k(\mathsf{ct}))$ by construction of $\mathsf{circuit}[\mathsf{valid}]$. We also denote by hpre the vector of partial accumulators defined by

$$\mathsf{hpre}(i) = \mathsf{Acc}(\mathsf{val}(1), \ldots, \mathsf{val}(i)) \quad i = 1, \ldots, n \qquad (2)$$

by using the incremental accumulator. (By convention, $\mathsf{hpre}(0) = \emptyset$.) Hence, each party has their own version of val and hpre, depending on their respective argument. The vector hpre is the key idea to our improved dispute resolution.

4.3. Dispute Resolution Phase

The dispute resolution consists of having V to spot a flaw in the argument by B. If no error in the proof is found by V, we assume that B is correct. The blockchain fees are covered by the dispute deposit ($\mathsf{fee}_d + \mathsf{tip}_d$ for each sponsor).

To commit to intermediate computations, the evaluations of the gates in the circuit are incrementally hashed. Namely, $w_i = \mathsf{hpre}(i)$ is hashing the evaluation of all gates until g_i. To spot the error, V applies a cut-and-choose search to find the first gate g_i on which the participants disagree. Then, the smart contract verifies this gate based on the agreed previous ones and determines who is right in the dispute.

Resolving dispute in an argument. After both dispute sponsors made their deposits, the dispute resolution starts.

Step 7. (Following Step 6.) The smart contract sets the variables $a = 1$ and $b = n + 1$ and sets deadlines to activate the several iterations of Step $7b$ and $7c$. Then, the protocol iterates through the following steps.

Step 7a. (Following Step 7 or $7c$.) The smart contract sets $i = \lfloor \frac{a+b}{2} \rfloor$ and waits for B to respond to challenge i.

Step $7a'$. (Following Step $7a$ but after the deadline for Step $7b$.) Upon activation by anyone, V wins as B did not activate Step $7b$, and it continues with Step 9.

Step $7b$. (Following Step $7a$.) B submits the response $w_i = \mathsf{hpre}(i)$ to the smart contract.

Step $7b'$. (Following Step $7b$ but after the deadline for Step $7c$.) Upon activation by anyone, B wins as V did not activate Step $7c$, and it continues with Step 9.

Step $7c$. (Following Step $7b$.) V says "left" or "right" to the smart contract. If V disagrees that $w_i = \mathsf{hpre}(i)$, V says "left". Otherwise, V admits that $w_i = \mathsf{hpre}(i)$ and says "right'.
If "left", the smart contract discards any stored w_j with $j > i$ and sets $b = i$. If "right", the smart contract discards any stored w_j with $j \leq i$ and sets $a = i + 1$.
If we have $a \neq b$, the protocol iterates with Step $7a$ again. Otherwise, V must submit more information as specified in the next step.

Step 8. (Following Step $7c$.) V sets i to the value of $a = b$ in the smart contract and proceeds to the following step, depending on the case.

Step $8a$. (Case $1 < i \leq n$ following Step 8.)

V/B input: $\mathsf{ID_V}, \mathsf{ID_B}, y, \mathsf{tip}_o, \mathsf{tip}_d, \mathsf{desc}_d, m, n, \mathsf{circuit[valid]}$
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 contract $= (\mathsf{ID_V}, \mathsf{ID_B}, y, m, n, c, \mathsf{tip}_o, \mathsf{tip}_d, \mathsf{params}, \mathsf{version})$
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 Eq. (1) and (2)
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 Step $8a$, $8b$, $8c$ in Section 4.3
34: verify the information from V as detailed in Step $8a$, $8b$, $8c$ in Section 4.3
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

Figure 3. The SOX Protocol

V takes the g_i gate of the circuit, sets $g_i = (\mathsf{op}, s_1, \ldots, s_a)$, $L = \{\ell; s_\ell < 0\}$, and computes

$$
\begin{aligned}
v_\ell &= \mathsf{val}(s_\ell) \qquad (\ell = 1, \ldots, a) \\
w'_i &= \mathsf{hpre}(i) \\
\pi_1 &= \mathsf{Prove}(\mathsf{circuit[valid]}, i) \\
\pi_2 &= \mathsf{Prove}(\mathsf{ct}, (-s_\ell)_{\ell \in L}) \\
\pi_3 &= \mathsf{Prove}((\mathsf{val}(1), \ldots, \mathsf{val}(i-1)), (s_\ell)_{\ell \notin L}) \\
\rho &= \mathsf{ProveExt}(\mathsf{val}(1), \ldots, \mathsf{val}(i))
\end{aligned}
$$

V submits

$$(o, g_i, v_1, \ldots, v_a, w'_i, \pi_1, \pi_2, \pi_3, \rho)$$

to the smart contract.
The smart contract computes $(h_{\mathsf{circuit}}, h_{\mathsf{ct}}) = \mathsf{Open}(c, o)$ and the value $v = \mathsf{op}(k, v_1, \ldots, v_a)$ following g_i, then verifies

$$
\begin{aligned}
&\mathsf{Verify}(h_{\mathsf{circuit}}, n, i, g_i, \pi_1) \\
&\mathsf{Verify}(h_{\mathsf{ct}}, m, (-s_\ell)_{\ell \in L}, (v_\ell)_{\ell \in L}, \pi_2) \\
&\mathsf{Verify}(w_{i-1}, i, (s_\ell)_{\ell \notin L}, (v_\ell)_{\ell \notin L}, \pi_3) \\
&\mathsf{VerifyExt}(i, w_{i-1}, w'_i, v, \rho)
\end{aligned}
$$

$$w_i \neq w'_i$$

If verification passes, V wins. Otherwise, B wins. It continues with Step 9.

Step $8b$. (Case $i = 1$ following Step 8: V said "left" all the time to disagree on every hpre; there is no w_{i-1} defined.)
V takes the g_i gate of the circuit, sets $g_i = (\mathsf{op}, s_1, \ldots, s_a)$, and computes

$$\begin{aligned} v_\ell &= \mathsf{ct}_{-s_\ell} \qquad (\ell = 1, \ldots, a) \\ w'_i &= \mathsf{hpre}(i) \\ \pi_1 &= \mathsf{Prove}(\mathsf{circuit}[\mathsf{valid}], i) \\ \pi_2 &= \mathsf{Prove}(\mathsf{ct}, (-s_1, \ldots, -s_a)) \\ \rho &= \mathsf{ProveExt}(\mathsf{val}(1)) \end{aligned}$$

V submits

$$(o, g_i, v_1, \ldots, v_a, w'_i, \pi_1, \pi_2, \rho)$$

to the smart contract.
The smart contract computes $(h_{\mathsf{circuit}}, h_{\mathsf{ct}}) = \mathsf{Open}(c, o)$ and the value $v = \mathsf{op}(k, v_1, \ldots, v_a)$ following g_i, then verifies

$$\begin{aligned} &\mathsf{Verify}(h_{\mathsf{circuit}}, n, i, g_i, \pi_1) \\ &\mathsf{Verify}(h_{\mathsf{ct}}, m, (-s_1, \ldots, -s_a), \\ &\quad (v_1, \ldots, v_a), \pi_2) \\ &\mathsf{VerifyExt}(i, \emptyset, w'_i, v, \rho) \\ &w_i \neq w'_i \end{aligned}$$

If verification passes, V wins. Otherwise, B wins. It continues with Step 9.

Step $8c$. (Case $i = n + 1$ following Step 8: V said "right" all the time to agree on every hpre.)
V computes $\pi_3 = \mathsf{Prove}(\mathsf{val}, n)$. V submits π_3 to the smart contract.
The smart contract verifies $\mathsf{Verify}(w_{i-1}, i, n, \mathsf{true}, \pi_3)$.
If verification passes, V wins. Otherwise, B wins. It continues with Step 9.

Step 9. (Following Step $7a'$, $7b'$, $8a$, $8b$, or $8c$.) Let $\mathsf{P} \in \{\mathsf{V}, \mathsf{B}\}$ be the winning party and Q be the other. If we are reaching this step for the first time or for the second time and P has changed since the first time, and if $\mathsf{Q} \neq \mathsf{SQ}$, then replace Q by SQ and go back to Step 7. *(This is a dispute resolution challenge.)*
If $\mathsf{P} = \mathsf{V}$, terminate with Complete. Otherwise, terminate with Cancel.

Clearly, the smart contract makes almost no computation except in the the final Step 8: in the worst case, the smart contract iterates $3 \log_2 n$ times and each iteration requires to store some w_i (and erase others). Additionally, it finishes by verifying one proof in h_{circuit}, one multiproof in h_{ct}, one multiproof in w_{i-1}, and one proof of update. The largest multiproof is for a values. Assuming that the maximum arity a is bounded by a constant, the total gas complexity is $\mathcal{O}(\log n)$.

Dispute resolution challenge phase. There is a *dispute resolution challenge* (in Step 9) when a dispute sponsor who lost a dispute resolution felt that the participant that they support did not follow the protocol. Actually, it may happen that both V and B are malicious and that they set up a dispute theater to make one specific participant P win. For the hypothetical case that Q had a valid argument which convinced a honest SQ, we must leave the opportunity to SQ to challenge the result. For that, SQ restarts the dispute and plays the role of Q against P.

It may also be the case that SQ is malicious too and runs a second dispute theater to lose. For the hypothetical case where it is P who had a valid argument and SP who is honest, we must leave the opportunity for SP to challenge a flipped winning decision. Hence, the dispute resolution could be run 3 times in total, just because we have two pairs of participants instead of two participants.

Our dispute resolution is such that a running honest participant cannot lose. So, the honest participant cannot be replaced in a challenge. The victory of the honest participant may be challenged by the adversarial party but the honest participant would run the dispute again and win again for good.

Interestingly, in each pair of participant, it does not matter who runs the dispute first and who can challenge. For instance, in the $\{\mathsf{B}, \mathsf{SB}\}$ pair, the first dispute resolution could be run as V against SB, but then B would challenge a losing decision.

5. Formal Result

We prove that our protocol satisfies the conditions of our formal definition Def. 1–4 through Th. 2–9 in Appendix. We summarize them 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 and BIND-secure, the static accumulator is CR-secure, the incremental accumulator is CR and CRext-secure.

6. Implementation

We implemented our protocol on Ethereum but without the full sponsoring options as we intended, since it is really new in Ethereum and was not available yet when we started. Hence, every participant pays for their messages to the smart contract. Sponsors mostly pay for the deployment of smart contracts. We could have the contract to pay back for their fees.

For the accumulators, we use Merkle trees as specified in Appendix B.2.

We designed two smart contracts and some libraries of smart contracts. The optimistic smart contract is deployed by S. The dispute resolution smart contract is deployed by a library which is called by the first smart contract after SV makes their deposit. We developed clients in Javascript. They are run by Firefox. Our results were obtained on Intel Core i7-10510U at 1.8GHz.

We chose to compare with OptiSwap [14] because we built on it and it is the most mature optimistic protocol based on smart contracts, and to compare with FDE [33] because it is the most recent and probably the most efficient competitor in the ZKCP family.

Compared to a plain (unfair) exchange protocol, the overhead for V and B is to encrypt/decrypt x using AES and to hash all computations using SHA256 to compute h_{circuit} and h_{ct}. We can rely on optimized implementations of AES and SHA256. Our circuit uses block compression gates instead of Boolean gates like in ZK techniques. Protocols such as OptiSwap [14] and ours have nearly no computational overhead in the optimistic phase.

Protocols such as FDE [33] always need cryptographic operations (typically, verifiable encryption for circuits at the bit/field level) which are quite expensive. They could be amortized by a vendor selling multiple copies of x but not by the buyer verifying the computation. The target application of FDE is the archive servers which provide data which is "stored on a blockchain" against payment, while it is only the commitment on the data which appears on the blockchain. In our setting, this means that the description function for x is the coherence with the commitment. When the data consists in 4096 field elements (that is, x of 128KB), the reported running time is of 89 seconds for V and of 34 seconds for B [33], using KZG commitment instead of SHA256. By extrapolation, buying 1GB of data x would take more than 3 days of computation for B for the sole verification of the ciphertext.

The optimistic phase of our protocol involves a smart contract storing and executing a constant number of values and operations. The more complex operations consist of verifying that the inputs come from the right sender, which is the same as verifying a digital signature of a transaction, what a blockchain typically does all the time. Contracts like FDE [33] or the optimistic phase of OptiSwap [14] are very similar. FDE has a gas cost estimated to 300 000 gas in total while OptiSwap (optimistic phase) has a gas cost of about 100 000 gas. Following the computation made for FDE with a gas at 14 GWei and ETH at US$2302.35, they obtain a cost of about US$10 on Ethereum for FDE, so three times less for OptiSwap.[5] We should however add the deploy gas cost which was estimated to 950 000 gas for OptiSwap and not mentioned for FDE. Our optimistic protocol has a gas cost in the same order of magnitude: 1 463 686 gas for deployment and 219 540 gas for running (Steps 1–3′).[6] We reported in Table 1 the running time of the Javascript clients for the engagement phase. For the smart contract execution, the running time is negligible compared to the interaction with the blockchain. As we can see, the clients need less than a second for $|x| = 4$MB. (This running time is linear.) By extrapolation, it should take 2 minutes for 1GB.[7]

The advantage of FDE is in involving no dispute resolution. As already mentioned, OptiSwap has a complexity which is linear in the depth of the circuit which characterizes x. Our protocol has a complexity which is logarithmic in its size, which is the same when the circuit is balanced but strictly better in other cases.

As for the gas cost of the dispute resolution, we have similar results as OptiSwap, when the circuit is balanced.[8] They estimated the cost of the dispute when selling 1GB of data, i.e. $m = 2^{24}$ with a Merkle hash. The total running price for the dispute is of 6 400 000 gas. If we hash with SHA256, there would be a factor of $\frac{2^{24}}{24} \approx 700\,000$ for OptiSwap.[9] With SOX, we run a dispute for x of 4MB ($m = 2^{16}$) and SHA256 hash and had a running cost of $16 \times 109\,613 + 5\,285\,895 = 7\,039\,703$ gas (without challenge). Extrapolated to 1GB, this gives $24 \times 109\,613 + 5\,285\,895 \approx 8\,000\,000$ gas.

7. Discussion

7.1. Enforcing Honest Behaviors

Our protocol can spot some malicious behaviors: any participant who has lost any iteration of the dispute resolution cannot be honest. In addition to this, any V who participated in a transaction which has passed Step 2 but ended up in a Cancel must be malicious. They could be banned. Hence, it is easy for S to decline sponsoring a new transaction with these participants.

7.2. Risk for the Optimistic Sponsor

Any participant could be malicious against an honest S. As mentioned several times, S is taking risks in sponsoring. This is inevitable if we want that V and B pay no fees even when the exchange aborts. In OptiSwap or FDE, B must pay some fees when making the deposit of $\$y$ in the smart contract, and this payment is lost even though the exchange

5. At the time of writing, 1 gas is 0.17 GWei and 1 ETH is US$3701 so gas is about 50 times cheaper.

6. The increase compared to OptiSwap comes from that we have extra steps for sponsoring and need to keep three types of deposits. Without sponsoring, the cost would be smaller.

7. At the time of writing, our implementation is done in JavaScript for a browser and the browser cannot handle a file as large as 1GB. For that, we are implementing another client which will be faster and without this limitation. This is why we only report the cost for a 4MB file.

8. The figures in Table 1 actually shows that we cost more gas. This is because we wanted to have a universal dispute resolution to handle any circuit, given as input. Instead, we could have hard-coded the circuit in the dispute solver and have no longer need for h_{circuit} and for the proof π_1 of the gate g_i. This obviously saves some gas cost. This is the approach which was adopted in OptiSwap, which explains the difference.

9. The depth of the circuit would be 2^{24} instead of 24.

can be canceled. There is no perfect solution about this. However, we can encourage economic inclusion by shifting the risks to an organization such as S. We can also propose partial solutions to limit the risks as explained below.

A malicious B could decide not to go to Step 2 after S has paid for Step 1. One way to fix that would be to have a smart contract ready to run Step 1 on behalf of S when presented with a valid signature from S. Then, B can trigger Step 1+2 by presenting the signature from S, contract, and the $\$(y+\mathsf{tip}_o)$ payment. The smart contract would then run Step 1+2. This way, a malicious B is of no harm to S.

A malicious V could run an exchange protocol and make it cancel (by not posting k or by posting a rogue one). They would not earn or lose anything from this strategy but S would lose Δ_o corresponding to setting up the contract and paying the fees for at least the payment by B. One way to fix this is to require $\mathsf{S} = \mathsf{V}$. This would still be sponsoring from the viewpoint of B (but not for V who would now need seed money to run the protocol).

The risk of S facing a malicious V can however be mitigated by appropriate risk analysis and filters. S can keep a white list based on subscription and/or reputation. They can keep black lists of malicious $\mathsf{ID_V}$ (as discussed in Section 7.1). A malicious vendor would harm only once for each identity $\mathsf{ID_V}$ but could create new identities. S can also limit the rate of transactions for vendors with no history (for instance, to one transaction at a time), or require a proof of work, or require to watch an advertisement. Such systems can operate independently of our SOX protocol.

Given that Δ_o is small and that S keeps control on the decision to sponsor, we believe that the risk is sustainable. Such attack can keep a marginal existence in a large volume of sponsoring. Below, we give a way to set up fees so that S should make profits when there is a large volume of transactions.

We let p be the probability that the exchange completes. We assume that S can make an estimate $\bar{p}$ of p by some market analysis or simply by a learning process. We also assume that S can make an estimate $\bar{\Delta}_o$ of Δ_o based on the current gas price. The expected income is $E(\Delta_\mathsf{S}) = p \cdot \mathsf{tip}_o - \Delta_o$. Hence, S can set $\mathsf{tip}_o = \frac{\bar{\Delta}_o+\theta}{\bar{p}}$ for a target value θ and get an expected income of

$$E(\Delta_\mathsf{S}) = \frac{p-\bar{p}}{\bar{p}}(\bar{\Delta}_o+\theta) + (\bar{\Delta}_o - \Delta_o) + \theta \qquad (3)$$

So, if the relative error of $\bar{p}$ is small and the error of $\bar{\Delta}_o$ is small, the expected income is close to θ. By summing over a large number N of transactions, $\frac{1}{N}\sum \Delta_\mathsf{S}$ quickly converges towards θ.

7.3. No Budget to Sponsor (Unfairness to the Poor)

What may happen is that an honest participant, engaged in a dispute with a malicious one, finds no sponsor while the other participant found one malicious sponsor. Assuming that the honest participant is "rich enough" to sponsor themselves by paying the expenses up until the winning case, this case is resolved. Otherwise, we may end up with unfair situations where honest participants are too poor to defend themselves and find no one willing to.

We stress that the problem may be nonexistent, because sponsoring a dispute comes with no risk and may be quite profitable. Furthermore, the dispute sponsor can easily be automated: the participant looking for one presents their argument; the potential sponsor verifies and decides to sponsor if correct. As there is no risk in sponsoring a participant with valid arguments, the decentralized autonomous organization service to sponsor disputes makes perfect sense. Investors should happily contribute. Hence, we believe that finding a dispute sponsor is not a problem.

The decision to sponsor can be done by an offchain server to which participants can submit their role $\mathsf{P} \in \{\mathsf{V}, \mathsf{B}\}$, $\mathsf{circuit}[\mathsf{valid}]$, and ct (i.e. argue without o) in an anonymous manner. By doing this, we reduce the problem of possible discrimination. Then, the server can issue a signed token to run a dispute to support P for h_{circuit} and h_{ct}. This token can be submitted to a smart contract SP with o and the reference to the optimistic smart contract.

7.4. Incentive for the Dispute Sponsor of the Buyer

Somewhat related to the problem of the honest B finding no sponsor in a dispute, we can observe that the dispute sponsor for B must be the first to be announced, and if the dispute resolution never starts (because SV is not announced), then SB is refunded with no profit. Since this must be the most common situation, we can see it as a lack of incentive for SB. However, the benefit when the dispute resolution is engaged can be important. Namely, if the argument of B is correct, SB is promised for $\Delta_{\mathsf{SB}} = \mathsf{fee}_d + \mathsf{tip}_d - \Delta_d$ when the dispute resolution starts, following the "winner-takes-all" approach. Since SB is quickly refunded when the dispute resolution does not start, we consider it as a lossless investment.

7.5. Too High Fees for Low Transactions

When y is small, we still need to pay for fee_o so to have $\mathsf{tip}_o > \mathsf{fee}_o$. This tip_o may render the transaction too expensive, compared to y. However, V and B could group many transactions between them into a single one to amortize the fees.

7.6. Limitation of Privacy

Avizheh et al. [4], [5] identified a privacy limitation in FairSwap and OptiSwap which also exists in our protocol: the dispute resolution may reveal some information about x and about the contract (namely, the description desc_d and price y). An honest participant P who engages in a shameful contract with a malicious participant Q can be blackmailed as follows: P is asked by Q to deviate from the contract (either by paying more than the price if $\mathsf{P} = \mathsf{B}$ or by accepting a lower payment if $\mathsf{P} = \mathsf{V}$), otherwise, Q would force the protocol to continue to the dispute resolution (although Q

would fail in the dispute) to disclose the shameful contract. Avizheh et al. [4] solve this problem in pOptiSwap by using an encoding of the circuit. This however does not hide desc_d and blows up the size of the circuit. We first need to have an arithmetic circuit (preventing us to have more elaborate gates such as AES block encryption or SHA256 compression gates) then use a secret sharing of each value into enough shares so that the number of values revealed by the dispute resolution would disclose nothing about x.

We believe that the problem of blackmail in shameful transactions remains no matter the privacy protection. Indeed, a malicious Q does not need to run a dispute resolution in order to disclose the terms of a transaction with P. The only question is whether the protocol leaks evidence of the transaction or maintains a plausible deniability by P. The pOptiSwap protocol does not solve this problem and neither do we. In our protocol, the opening of the commitment c in contract can always be disclosed by Q to reveal desc_d, y, and ct as a digital evidence. Then, the key k would disclose x. Making the fair exchange deniable is left to future work.

7.7. Side-Contracts and Censorship

Clearly, a buyer-miner coalition can intercept the attempt by V to submit k in Step 3 and stall it until timeout for the transaction to cancel. However, B would get k and deduce x without paying, which is unfair. Similarly, a vendor-miner coalition, after submitting a bogus k in Step 3, can intercept the attempt of B to trigger a dispute in Step $4b$ and stall it until timeout for the transaction to complete. However, vendor would get the payment $\$y$ without giving x, which is unfair.

We could have introduced the bomb option like in Rapidash [9] and proved CSP-fairness. However, we feel that CSP-fairness is weaker than our notion and still prone to coalition with external incentives.

Another approach is to use encrypted mempools, which are currently under development.

Censorship should rather be solved at the blockchain level. Essentially, the censorship model assumes that one single miner is elected to mine the transaction. The miner is not known in advance but corrupting several eligible miners ends up in a probabilistic censorship. The blockchain should make this probability negligible.

7.8. Simplification by Covering the Dispute

Assuming that the blockchain fees in the dispute resolution are low, they cannot be an obstacle for the optimistic exchange sponsor to engage. Hence, the protocol can be simplified by removing the two dispute sponsors and tip_d, and by letting the dispute fees covered by the S-deposit. As this removes the need to seek for dispute sponsors, posting argue is no longer necessary. Similarly, challenging the dispute resolution result should be removed. The additional risk for S is to lose Δ_d more than before (hence up to $\mathsf{fee}_o + \mathsf{fee}_d$). It is more severe as any malicious party could always force to go to the dispute to make S earn less, as there is no need to look for a dispute sponsor any more. If potential sponsors S are willing to take the risk, such simplification could be interesting.

8. Conclusion

We formalized the notion of optimistic sponsored exchange with no entrance fee and no risk. We achieve fairness and privacy of the exchange against polynomial-time adversaries, including random drop-out. 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 and Philip Hamelink for their contribution in the implementation.

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] Sepideh Avizheh, Preston Haffey, Reihaneh Safavi-Naini. Privacy-Enhanced OptiSwap. In *Proceedings of the 2021 on Cloud Computing Security Workshop (CCSW 2021)*, Virtual event, pp. 39–57, ACM, 2021. https://doi.org/10.1145/3474123.3486756

[5] Sepideh Avizheh, Preston Haffey, Reihaneh Safavi-Naini. Privacy-Preserving FairSwap: Fairness and Privacy Interplay. *Proceedings on Privacy Enhancing Technologies*, vol. 2022 (1), pp. 417–439, 2022. https://doi.org/10.2478/popets-2022-0021

[6] Niko Barić, Birgit Pfitzmann. Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees. In *Advances in Cryptology EUROCRYPT'1997*, Konstanz, Germany, Lecture Notes in Computer Science 1233, pp. 480–494, Springer-Verlag, 1997.

[7] Josh Benaloh, Michael de Mare. One-Way Accumulators: A Decentralized Alternative to Digital Signatures. In *Advances in Cryptology EUROCRYPT'1993*, Lofthus, Norway, Lecture Notes in Computer Science 765, pp. 274–285, Springer-Verlag, 1994.

[8] 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

[9] Hao Chung, Elisaweta Masserova, Elaine Shi, Sri AravindaKrishnan Thyagarajan. Rapidash: Foundations of Side-Contract-Resilient Fair Exchange. Presented at the Workshop on Cryptographic Tools for Blockchains (CTB 2024).
Eprint 2022/1063 https://eprint.iacr.org/2022/1063
Also: to appear at Financial Cryptography 2025 with title: Rapidash: Atomic Swaps Secure under User-Miner Collusion.

[10] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, Filippo Valsorda. Privacy Pass: Bypassing Internet Challenges Anonymously. *Proceedings on Privacy Enhancing Technologies*, vol. 2018 (3), pp. 164–180, 2018. https://doi.org/10.1515/popets-2018-0026

[11] Whitfield Diffie, Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, vol. IT-22, pp. 644–654, 1976. https://doi.org/10.1109/TIT.1976.1055638

[12] 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.

[13] 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

[14] 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

[15] Andreas Erwig, Sebastian Faust, Kristina Hostáková, Monosij Maitra, Siavash Riahi. Two-Party Adaptor Signatures From Identification Schemes. In *Public Key Cryptography'2021*, Virtual event, Lecture Notes in Computer Science 12710–41271, pp. 451–480, Springer-Verlag, 2021. https://doi.org/10.1007/978-3-030-75245-3_17
Eprint 2021/150. https://eprint.iacr.org/2021/150

[16] Shimon Even, Yacov Yacobi. Relations Among Public Key Signature Systems. Technical Report 175. Technion, Haifa, Israel, 1980.

[17] Amos Fiat, Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology CRYPTO'86*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 263, pp. 186–194, Springer-Verlag, 1987.

[18] Michael J. Fischer, Nancy A. Lynch, Michael S. Paterson. Impossibility of Distributed COnsensus with One Faulty Process. *Journal of the ACM*, vol. 32 (2), pp. 374–382, 1985.

[19] Matthew K. Franklin, Michael K. Reiter. Fair Exchange with a Semi-Trusted Third Party. In *4th ACM Conference on Computer and Communications Security*, Zürich, Switzerland, pp. 1–6, ACM Press, 1997. https://doi.org/10.1145/266420.266424

[20] 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

[21] Songlin He, Yuan Lu, Qiang Tang, Guiling Wang, Chase Qishi Wu. Fair Peer-to-Peer Content Delivery via Blockchain. In *Computer Security – ESORICS 2021*, Darmstadt, Germany, Lecture Notes in Computer Science 12972, pp. 348–369, Springer-Verlag, 2021.

[22] Simon Janin, Kaihua Qin, Akaki Mamageishvili, Arthur Gervais. FileBounty: Fair Data Exchange. In *IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, Genoa, Italy, pp. 357–366, IEEE, 2020. https://doi.org/10.1109/EuroSPW51379.2020.00056
ArXiv 2008.11362 https://doi.org/10.48550/arXiv.2008.11362

[23] 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 fflexible 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

[24] 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

[25] 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

[26] Seungwoo Kim, Semin Han, Seongho Park, Kyeongtae Lee, Jihye Kim, Hyunok Oh. zkMarket : Privacy-preserving Digital Data Trade System via Blockchain. Eprint 2024/1775. https://eprint.iacr.org/2024/1775

[27] Ralph Merkle. A Certified Digital Signature. In *Advances in Cryptology CRYPTO'89*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 435, pp. 218–238, Springer-Verlag, 1990.

[28] Silvio Micali. Simple and Fast Optimistic Protocols for Fair Electronic Exchange. In *Proceedings of the Twenty-Second Annual Symposium on Principles of Distributed Computing (PODC 2003)*, Boston MA, USA, pp. 12–19, ACM, 2003. https://doi.org/10.1145/872035.872038

[29] Henning Pagnia, Felix C. Gärtner. On the Impossibility of Fair Exchange without a Trusted Third Party. Technical Report TUD-BS-1999-02, Darmstadt University of Technology, 1999. https://www.cs.utexas.edu/~shmat/courses/cs395t_fall04/pagnia.pdf

[30] Joseph Poon, Thaddeus Dryja. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. 2016. https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts

[31] Claus P. Schnorr. Efficient Identification and Signature for Smart Cards. In *Advances in Cryptology CRYPTO'89*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 435, pp. 235–251, Springer-Verlag, 1990.

[32] Claus P. Schnorr. Efficient Identification and Signature for Smart Cards. *Journal of Cryptology*, vol. 4, pp. 161–174, 1991.

[33] Ertem Nusret Tas, István András Seres, Yinuo Zhang, Márk Melczer, Mahimna Kelkar, Joseph Bonneau, Valeria Nikolaenko. Atomic and Fair Data Exchange via Blockchain. https://dl.acm.org/doi/proceedings/10.1145/3658644
Eprint 2024/418. https://eprint.iacr.org/2024/418

[34] Fredrik Winzer, Benjamin Herd, Sebastian Faust. Temporary Censorship Attacks in the Presence of Rational Miners. In *IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, Stockholm, Sweden, pp. 357–366, IEEE, 2019. https://doi.org/10.1109/EuroSPW.2019.00046
Eprint 2019/748 https://eprint.iacr.org/2019/748

[35] Eric Wagner, Achim Völker, Frederik Fuhrmann, Roman Matzutt, Klaus Wehrle. Dispute Resolution for Smart Contract-based Two-Party Protocols. In *IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, Seoul, South Korea, pp. 422–430, IEEE, 2019. https://doi.org/10.1109/BLOC.2019.8751312

Appendix A. Detailed Proofs

We first discuss the case of a malicious sponsor S which is easy. Actually, the only action of S is to set up a smart contract and pay at the beginning. If there is no honest participant, we actually do not care about what happens. Hence, there must be one honest participant. Clearly, the first honest participant who needs to engage would see any misbehavior from S when they check the smart contract and would not engage in the protocol. Hence, we can assume S to be honest in what follows.

The optimistic sponsor S sets up a smart contract which is following the protocol. Thanks to the assumed blockchain property, every honest participant sees the smart contract in the same state, due to consensus. We assume the blockchain to be available for read with no delay. To add a transaction in the blockchain (i.e. to give an input to the smart contract), we will explicitly say when we assume an always available write access with no delay by saying that there is no censorship.

A.1. SOX Properties

Theorem 2 (SOX properties and consensus). Our protocol terminates with consensus, is correct, optimistic, with disincentivized disputes. It is riskless for V and B. As for S, we have $\Delta_{\mathsf{S}} = -\Delta_o$ in cancel case, and $\Delta_{\mathsf{S}} = \mathsf{tip}_o - \Delta_o$ in complete case. resolution.

This shows that our protocol satisfies the SOX conditions in Def. 1,2, except for the risk of the dispute sponsor which will be covered in Th. 4 and Th. 6.

Termination and Consensus. Our protocol aborts after a timeout when messages do not come as expected. When they do, we clearly see that their number is strictly bounded. So, the protocol eventually terminates for the honest participants. The final state Complete or Cancel is decided by the smart contract. Due to consensus in the blockchain, every honest participant ends with that same state.

Correctness and Optimism. Clearly, our protocol satisfies the *correctness* and *optimistic* notions, which involve honest V, B, and S. They end without dispute on the same state, and it is Complete if and only if they start with matching inputs.

Disincentivized disputes. The cumulated seed money for the dispute resolution is of $2\mathsf{fee}_d + 2\mathsf{tip}_d$ while the cost is of $\Delta_d \leq \mathsf{fee}_d$. The losing sponsor gets nothing in the end and is paying $\mathsf{fee}_d + \mathsf{tip}_d$. Hence, $\min(\Delta_{\mathsf{SV}}, \Delta_{\mathsf{SB}}) = -\mathsf{fee}_d - \mathsf{tip}_d \leq -\Delta_d - \mathsf{tip}_d$ and $\max(\Delta_{\mathsf{SV}}, \Delta_{\mathsf{SB}}) = \mathsf{fee}_d + \mathsf{tip}_d - \Delta_d \geq \mathsf{tip}_d$.

Balance of the honest V, B, S. V does not pay anything and is being paid only in complete cases, for an amount of $\$y$. Hence, $\Delta_{\mathsf{V}} = 0$ in cancel cases and $\Delta_{\mathsf{V}} = y$ in complete cases.

B has to pay $\$(y + \mathsf{tip}_o)$, and is being paid only in cancel cases, for a full refund. Hence, $\Delta_{\mathsf{B}} = 0$ in cancel cases and $\Delta_{\mathsf{B}} = -y - \mathsf{tip}_o$ in complete cases.

S has to pay $\$\mathsf{fee}_o$ as a deposit and is rewarded tip_o in complete cases, and is being returned $\mathsf{fee}_o - \Delta_o$ in the end. Hence, we deduce Δ_{S} in the two cases.

Balance of the honest dispute sponsor. Due to Th. 4 and Th. 6 below, based on assumptions, an honest dispute sponsor does not lose, except with negligible probability. Hence, the outcome of the protocol is to refund the dispute sponsor from the seed money $\mathsf{fee}_d + \mathsf{tip}_d$ and to reward with at least tip_d. Hence, $\Delta \geq \mathsf{tip}_d$.

A.2. Honesty Pays

We want to prove that an honest participant cannot lose a dispute, except with negligible probability. For that, we consider all possible roles.

First of all, the case of the optimistic sponsor S is trivial as there is no notion of losing in a dispute. In all other cases, we observe that we need the no censorship assumption. Otherwise, the honest participant may be censored when trying to plead their case.

Case of an honest B. If B is honest, they triggered the dispute because they realized that $\mathsf{desc}_d(\mathsf{Dec}_k(\mathsf{ct})) = \mathsf{false}$. Hence, they must have made a valid argument which is supporting B. Below, we use val and hpre as defined by B.

During dispute resolution, B is asked to commit to some elements of hpre in their argument. The loop in Step 7 ends with B committing on $w_{i-1} = \mathsf{hpre}(i-1)$ and $w_i = \mathsf{hpre}(i)$, except in the corner cases $i = 1$ (only w_i is provided) and $i = n + 1$ (only w_{i-1} is provided).

V provides some input o. Due to the BIND-security of the commitment, this opens to the right h_{circuit} and h_{ct} in the smart contract, except with negligible probability. In the $0 < i \leq n$ case (Step $8a$ and $8b$), V provides g_i and a proof π in the accumulator $h_{\mathsf{circuit}} = \mathsf{Acc}(\mathsf{circuit}[\mathsf{valid}])$. Due to the CR-security of the static accumulator, this gives the correct gate to the smart contract, except with negligible probability. Similarly, the elements $v_1, \ldots, v_a$ must be correct in the smart contract: either proven with π_2 based on $h_{\mathsf{ct}} = \mathsf{Acc}(\mathsf{ct})$, or based on π_3 based on $w_{i-1} = \mathsf{Acc}(\mathsf{val}(1), \ldots, \mathsf{val}(i-1))$. Hence, the smart contract computes the right value $v = \mathsf{val}(i)$ by evaluating the gate.

In the $1 < i \leq n$ case (Step $8a$), we know that $w_{i-1} = \mathsf{Acc}(\mathsf{val}(1), \ldots, \mathsf{val}(i-1))$ is updated into $w_i = \mathsf{Acc}(\mathsf{val}(1), \ldots, \mathsf{val}(i))$ by the value v. So, it can be proven by V, thanks to correctness. Hence, due to the CRext-security of the incremental accumulator, V cannot make another proof ρ that w_{i-1} is updated into w'_i by the value v, with $w_i \neq w'_i$, except with negligible probability.

In the $i = 1$ case (Step $8b$), the same argument holds by replacing w_{i-1} by $\emptyset$.

In the $i = n+1$ case (Step $8c$), we know that the last element in $w_{i-1} = \mathsf{Acc}(\mathsf{val}(1), \ldots, \mathsf{val}(n))$ is $\mathsf{val}(n) = \mathsf{false}$. Hence, the probability that V succeeds to prove the opposite is negligible, due to the CR-security of the incremental accumulator.

Even if the outcome is challenged, B can proceed again and win, except with negligible probability. The probability for B to lose is negligible. We obtain the following result.

Theorem 3 (Honest* B *wins dispute). We consider the execution of a session of the protocol, assuming no censorship in the blockchain, that the commitment is BIND-secure, that the static accumulator is CR-secure, that the incremental accumulator is CR-secure and CRext-secure, that any participant takes the role of SB, and an honest B. The probability that B loses in dispute is negligible.

Case of an honest SB**.** If the dispute sponsor SB is honest, they must have been shown a valid argument which is supporting B. In the event that a malicious B sabotages the dispute resolution and loses, SB can challenge the result and run the dispute resolution using the argument like in the honest B case. As shown before, SB can only win, except with negligible probability. Even if the outcome is challenged again, SB can proceed again and eventually wins. The probability to lose is negligible. We obtain the following result.

Theorem 4 (Honest* SB *wins dispute). We consider the execution of a session of the protocol, assuming no censorship in the blockchain, that the commitment is BIND-secure, that the static accumulator is CR-secure, that the incremental accumulator is CR-secure and CRext-secure, and an honest SB. The probability that SB loses in dispute is negligible.

Case of an honest V**.** If V is honest, we have $\mathsf{desc}_d(\mathsf{Dec}_k(\mathsf{ct})) = \mathsf{true}$. Hence, they must have made a valid argument supporting V. Below, we use val and hpre as defined by V.

We prove by induction that at each iteration of Step 7, the smart contract has the commitment by B to some value $w_{a-1} = \mathsf{hpre}(a-1)$ if $a > 1$, and to some $w_i \neq \mathsf{hpre}(i)$ for $a \leq i \leq b$ if $b \leq n$. Clearly, the loop ends with $a = b$. In the end, we have an index i, some value w_{i-1} if $i > 1$, and some value w_i if $i \leq n$. These values are such that $w_{i-1} = \mathsf{hpre}(i-1)$ and $w_i \neq \mathsf{hpre}(i)$.

In the $1 < i \leq n$ case (Step $8a$), it is clear that V can prove what is required and win, due to the correctness of the commitment and of the accumulators. In the $i = 1$ case (Step $8b$) is similar with $w_{i-1} = \emptyset$. In the $i = n+1$ case (Step $8c$), it is clear that V can prove that the last element in w_n is true. Hence, V always wins.

Even if the outcome is challenged, V can proceed again and win. The overall probability to lose is null. We obtain the following result.

Theorem 5 (Honest* V *wins dispute). We consider the execution of a session of the protocol, assuming no censorship in the blockchain, that any participant takes the role of SV, and an honest V. The probability that V loses in dispute is null.

Case of an honest SV**.** If the dispute sponsor SV is honest, they have been shown a valid argument by V which is supporting V. So, the proof goes the same as for the honest V.

If malicious V sabotages the dispute resolution and loses, SV can challenge the outcome and redo the proof to isolate the error in the argument and spot it. The overall probability to lose is null. We obtain the following result.

Theorem 6 (Honest* SV *wins dispute). We consider the execution of a session of the protocol, assuming no censorship in the blockchain and an honest SV. The probability that SV loses in dispute is null.

A.3. Privacy

Privacy in the case of a dispute is counterintuitive, because V and B reveal everything, at least to potential sponsors, by posting argue. It should be clear that we reach this step (Step 4) when either V or B is malicious. If V is malicious, x and desc_d are already known by a malicious party so privacy is pointless. The remaining case is when V is honest and B is malicious. In that case, since B must have already paid for getting x to reach Step 4, there is no privacy concern any more: the purpose was to sell x to the adversary and the money cannot escape from V due to our fairness result in Th. 5.

Honest-B **privacy.** We assume an honest B.

Theorem 7 (Honest-*B *privacy). Assuming that communication between V and B during the preliminary phase and the engagement phase is private and that Com is HIDE-secure, the protocol enforces privacy for the honest B.

When V and B are honest, there is no possible dispute. As neither V nor B can miss a deadline and that the adversary cannot interfere with the communications, the game works as if the adversary was passive. Hence, the adversary learns only the communications to the smart contract, which is the content of contract and a random key. In the contract, only the commitment could reveal information.

We change the Privacy game to a game where B does not make any verification, except that the smart contract is correct. Hence, B accepts any contract and any k. The two games produce the same output. Then, we change the game by having c made by V replaced by a random string in the contract. Assuming that Com *is hiding*, this new game produces indistinguishable results.

Now, the adversary receives no information about x and desc_d, except about their lengths. Clearly, a simulator can now perfectly simulate what oracles answer to $\mathcal{A}$.

Malicious-B privacy. We assume a malicious B.

Theorem 8 (Malicious-B privacy). Assuming no possible censorship in the blockchain, that any participant takes the role of SV, that Enc is INDCPA-secure, the protocol enforces privacy with a malicious B.

In the case that V is honest but B is malicious, the adversary in the Privacy game receives ct which is the encryption of x. As soon as V releases the encryption key k, assuming that there is no censorship, we know that the protocol is never canceled due to Th. 5. Hence, we can modify the game by stopping and returning 0 as soon as V activates Step 3, with no effect on the output of the game. What remains is a game where ct is released and k is not disclosed. Due to INDCPA-security, we can replace x by a random string and have a game producing indistinguishable output. Then, the entire game becomes simulatable using $|x|$, desc_d, and y only.

A.4. Fairness

Theorem 9 (Fairness). Under the assumptions of Th. 2–8, our protocol enforces fairness.

We assume an honest B. If B does not trigger any dispute, it means $\mathsf{desc}_d(x')$ is true as B verified it in Step $4b$. If B triggers a dispute, since there is *no censorship*, the smart contract activates the dispute. We further assume that a dispute sponsor SB agrees to support B and no censorship when registering. Hence, to end up with a Complete decision, we must go to the dispute resolution and have B to lose. We show in Th. 3 that an honest B loses a dispute resolution with negligible probability. So, the Complete cases with dispute are negligible, and other Complete cases have $\mathsf{desc}_d(x')$ true in the Fairness game.

Appendix B. More Building Blocks

B.1. Cryptographic Primitives

One-time symmetric encryption. We use a variable-length symmetric encryption scheme Enc/Dec. The algorithm $\mathsf{ct} = \mathsf{Enc}(k, x)$ takes a key k and a plaintext x to return a ciphertext ct. We assume that this operation is length-preserving. The decryption algorithm $x = \mathsf{Dec}(k, \mathsf{ct})$ takes a key k and a ciphertext ct to return the plaintext x. Correctness implies that for any x and k, if $\mathsf{ct} = \mathsf{Enc}(k, x)$, then $\mathsf{Dec}(k, \mathsf{ct}) = x$.

The encryption uses a key only once. Hence, we can use AES in CTR mode with a fixed zero-nonce.

The encryption must be such that for any PPT adversary $\mathcal{A}$, $\Pr[\mathsf{INDCPA}_1 \to 1] - \Pr[\mathsf{INDCPA}_0 \to 1]$ is negligible with the following game:

Game INDCPA_b:
1: $\mathcal{A} \to (x_0, x_1, \mathsf{st})$
2: **if** $|x_0| \neq |x_1|$ **then return** 0
3: pick a key k at random
4: $\mathsf{ct} \leftarrow \mathsf{Enc}_k(x_b)$
5: $\mathcal{A}(\mathsf{st}, \mathsf{ct}) \to z$
6: **return** z

Essentially, $\mathcal{A}$ cannot distinguish from ct if it was x_0 or x_1 which was encrypted.

Commitment scheme. We use a commitment scheme Com/Open. The algorithm $(c, o) \leftarrow \mathsf{Com}(h)$ takes an input h and returns a commit value c and an opening key o. Opening is done by $\mathsf{Open}(c, o)$ which returns either some h or $\perp$. Correctness implies that for any h and any outcome $(c, o) \leftarrow \mathsf{Com}(h)$, we have $\mathsf{Open}(c, o) = h$.

As a commitment scheme, we can use what follows. To run $(c, o) \leftarrow \mathsf{Com}(h)$, we pick a random rnd, set $o = h \| \mathsf{rnd}$, and $c = H(o)$. To run $\mathsf{Open}(c, o)$, we return $\perp$ if $c \neq H(o)$ and $\mathsf{Open}(c, o) = h$ otherwise (after parsing $o = h \| \mathsf{rnd}$).

The commitment scheme must be hiding, i.e. such that for any PPT adversary $\mathcal{A}$, $\Pr[\mathsf{HIDE}_1 \to 1] - \Pr[\mathsf{HIDE}_0 \to 1]$ is negligible with the following game:

Game HIDE_b:
1: $\mathcal{A} \to (h_0, h_1, \mathsf{st})$
2: $(c, o) \leftarrow \mathsf{Com}(h_b)$
3: $\mathcal{A}(\mathsf{st}, c) \to z$
4: **return** z

Essentially, $\mathcal{A}$ cannot distinguish from c if it was h_0 or h_1 which was committed.

The commitment must also be binding, i.e. such that for any PPT adversary $\mathcal{A}$, $\Pr[\mathsf{BIND} \to 1]$ is negligible with the following game:

Game BIND:
1: $\mathcal{A} \to (c, o_0, o_1)$
2: $\mathsf{Open}(c, o_0) \to h_0$
3: **if** $h_0 = \perp$ **then return** 0
4: $\mathsf{Open}(c, o_1) \to h_1$
5: **if** $h_1 = \perp$ **then return** 0
6: **if** $h_0 = h_1$ **then return** 0
7: **return** 1

Essentially, $\mathcal{A}$ is issuing a commitment to be opened in two different manners.

B.2. Merkle-Based Accumulator

We define our Merkle accumulator as follows. For domain separation, we use two hash functions H and H'. Given a list of ℓ values $v_1, \ldots, v_\ell$, construct a Merkle tree by we first computing the depth $\delta = \lceil \log_2 \ell \rceil$. Bitstrings $\mathsf{str} = \cdots b_1 b_0$ of length up to δ represent a node in the Merkle tree. The empty string $\emptyset$ represents the root. Leaves have length δ while inner nodes have a smaller length. We define $\mathsf{toint}(\mathsf{str}) = \sum_j 2^j b_j$. Inner nodes str have one son str0 or two sons str0 and str1. We require that there are exactly ℓ leaves. The value v_i is assigned to the leave str such that $\mathsf{toint}(\mathsf{str}) = i - 1$. Finally, to each node str we assign a value which is recursively computed as follows: for a leaf str, we assign $\mathsf{val}_{\mathsf{str}} = H'(v_{\mathsf{toint}(\mathsf{str})+1})$. For an inner node with one son $\mathsf{val}_{\mathsf{str}} = H(\mathsf{val}_{\mathsf{str}0})$. For an inner node with two sons $\mathsf{val}_{\mathsf{str}} = H(\mathsf{val}_{\mathsf{str}0}, \mathsf{val}_{\mathsf{str}1})$.

We define $\mathsf{Acc}(v_1,\ldots,v_\ell) = \mathsf{val}_\emptyset$. The Merkle tree is computed with complexity $\mathcal{O}(\ell)$. Once computed, Acc takes complexity $\mathcal{O}(1)$: the time to read $\mathsf{val}_\emptyset$.

Given $i_1,\ldots,i_a$, we define the set $\mathsf{ancestors}(i_1,\ldots,i_a)$ of all the prefixes of the δ-bit strings str for which there exists j such that $\mathsf{toint}(\mathsf{str}) = i_j - 1$. We further define the set $\mathsf{missing}(i_1,\ldots,i_a)$ of strings str which are not element of $\mathsf{ancestors}(i_1,\ldots,i_a)$ but represent an existing sibling of an element of it in the tree. The size of $\mathsf{missing}(i_1,\ldots,i_a)$ is $\mathcal{O}(a\log\ell)$.

We define $\mathsf{Prove}((v_1,\ldots,v_\ell),(i_1,\ldots,i_a))$ as the tuple π of all $\mathsf{val}_{\mathsf{str}}$ for $\mathsf{str} \in \mathsf{missing}(i_1,\ldots,i_a)$. Once the Merkle tree is computed, the size of π is of $\mathcal{O}(a\log\ell)$ elements and the complexity of Prove is to read them.

The predicate $\mathsf{Verify}(h,\ell,(i_1,\ldots,i_a),(v'_1,\ldots,v'_a),\pi)$ recomputes the Merkle tree with available data: the number of leaves is ℓ, which allows to get δ, the leaves str such that $\mathsf{toint}(\mathsf{str}) = i_j - 1$ for some j are assigned to the $H'(v'_j)$ values, and the nodes str in $\mathsf{missing}(i_1,\ldots,i_a)$ are assigned to $\mathsf{val}_{\mathsf{str}}$ as specified in π. This is enough to compute $\mathsf{val}_\emptyset$ and to compare it with h. The complexity of Verify is $\mathcal{O}(a\log\ell)$.

This accumulator can be extended as an incremental accumulator. Interestingly, the Merkle tree of $(v_1,\ldots,v_i)$ is always a subtree of the Merkle tree of $(v_1,\ldots,v_\ell)$ for $i \le \ell$, with the same values. Hence, once the Merkle tree of $(v_1,\ldots,v_\ell)$ is computed, the complexity of $\mathsf{Acc}(v_1,\ldots,v_i)$ is $\mathcal{O}(\log\ell)$ and the one of $\mathsf{Prove}((v_1,\ldots,v_i),(i_1,\ldots,i_a))$ is $\mathcal{O}(a\log\ell)$.

We define $\mathsf{ProveExt}(v_1,\ldots,v_i) = \mathsf{Prove}((v_1,\ldots,v_i),i)$. Once the Merkle tree of $(v_1,\ldots,v_\ell)$ is computed, the complexity of $\mathsf{ProveExt}(v_1,\ldots,v_i)$ is $\mathcal{O}(\log\ell)$, as well as the size of ρ.

For $\mathsf{VerifyExt}(i,w_{i-1},w_i,v_i,\rho)$, we actually recompute w_{i-1} and w_i from (i,v_i,ρ) and verify that it matches the inputs w_{i-1} and w_i. The complexity is $\mathcal{O}(\log\ell)$.

B.3. RSA-Based Accumulator

We can build an RSA-based accumulator [6], [7]. First, we need to map index-value pairs (i,v) to accumulate to different prime numbers $\mathsf{map}(i,v)$. The accumulator is set with an RSA modulus N of unknown factorization and an initial value h_0. Then, $\mathsf{Acc}(v_1,\ldots,v_\ell) = h_0^{\mathsf{map}(1,v_1)\cdots\mathsf{map}(\ell,v_\ell)} \bmod N$. We have

$$\mathsf{Prove}(\mathsf{val},(i_1,\ldots,i_a)) = h_0^{\prod_{\ell\in L}\mathsf{map}(\ell,v_\ell)} \bmod N$$

where $L = \{1,\ldots,n\} - \{i_1,\ldots,i_a\}$ and

$$\begin{aligned} & \mathsf{Verify}(h,\ell,(i_1,\ldots,i_a),(v_1,\ldots,v_a),\pi) \\ \Leftrightarrow\quad & h = \pi^{\mathsf{map}(i_1,v_1)\cdots\mathsf{map}(i_a,v_a)} \bmod N \end{aligned}$$

The proof of update is void and $\mathsf{VerifyExt}(i,h,h',v,\rho)$ is equivalent to $h' = h^{\mathsf{map}(i,v)} \bmod N$. Proofs have constant size (say π of 2048 bits) but the complexity of map is high: each (i,v) should be mapped to a "random" unique prime (say of 270 bits) and verification should require a full exponential with a modulus of 2048 bits. Besides, verification would need to be implemented in a smart contract. We believe that the Merkle accumulator would be better. Implementation will say.

Appendix C. More Examples

C.1. Hash Preimage

We take as an example the case where x is a 256-bit string characterized as being the hash preimage of some value d. In this case, the function desc is actually the hash function H. We assume that the hash function is SHA256 and that x and d are two 256-bit keys. Using AES-CTR, we have

$$\mathsf{Dec}_k(\mathsf{ct}) = \mathsf{ct} \oplus (\mathsf{AES}_k(0)\|\mathsf{AES}_k(1))$$

With a so simple example, it could make sense to modify the protocol so that $(c,o) \leftarrow \mathsf{Com}(\mathsf{ct})$ and B directly submits o in Step 4 to launch a dispute. Then, the dispute consists of the smart contract opening c to get ct and verifying that $\mathsf{SHA256}(\mathsf{Dec}_k(\mathsf{ct})) = d$. There is no need for SV and SB.

C.2. Signature Service

As another example, we take the case where x is a signature on a given message M (known by V and B) which is valid for the given public key pk (also known by V and B). An application could be for a certificate authority to sell a certificate for M. It could also be to exchange a loan against a binding debt acknowledgement.

For an ECDSA signature, we have $x = (r,s)$ such that

$$r = \mathsf{xcoord}\left(\frac{H(M)}{s}G + \frac{r}{s}\mathsf{pk}\right)$$

We assume that r and s are represented on 256 bits. We split $\mathsf{ct} = \mathsf{ct}_l\|\mathsf{ct}_r$. We assume

$$r = \mathsf{ct}_l \oplus (\mathsf{AES}_k(0)\|\mathsf{AES}_k(1)) \qquad s = \mathsf{ct}_r \oplus (\mathsf{AES}_k(2)\|\mathsf{AES}_k(3))$$

We let $h = H(M)$. We can define the $\mathsf{circuit}[\mathsf{valid}]$ with 10 gates $(g_1,\ldots,g_{10})$ as follows.

- $g_1 = (\oplus\mathsf{AES}_{k,0,1},-1)$ is a gate evaluating to r with son g_{-1};
- $g_2 = (\oplus\mathsf{AES}_{k,2,3},-2)$ is a gate evaluating to s with son g_{-2};
- $g_3 = (\mathsf{invmod},2)$ is a gate evaluating to $\frac{1}{s}$ with son g_2;
- $g_4 = (\mathsf{mulmod}_h,3)$ is a gate evaluating to $\frac{h}{s}$ with son g_3;
- $g_5 = (\mathsf{scalmul}_G,4)$ is a gate evaluating to $\frac{h}{s}G$ with son g_4;
- $g_6 = (\mathsf{mulmod},1,3)$ is a gate evaluating to $\frac{r}{s}$ with sons g_1 and g_3;
- $g_7 = (\mathsf{scalmul}_{\mathsf{pk}},6)$ is a gate evaluating to $\frac{r}{s}\mathsf{pk}$ with son g_6;
- $g_8 = (+,5,7)$ is a gate evaluating to $\frac{h}{s}G + \frac{r}{s}\mathsf{pk}$ with sons g_5 and g_7;

- $g_9 = (\mathsf{xcoord}, 8)$ is a gate evaluating to xcoord with son g_8;
- $g_{10} = (=, 1, 9)$ is a gate evaluating to $\mathsf{desc}_d(\mathsf{Dec}_k(\mathsf{ct}))$ with sons g_1 and g_9;

We have $m = 2$ and $n = 10$. Using Merkle trees, a proof of membership in the Merkle accumulator consists of four hashes.

This circuit with 10 gates makes sense if we want to use dispute solver with a general-purpose set of instructions, as it could be used for other, or if we want to minimize the number of big number operations to be done by the smart contract circuits as well. We can have other strategies, such as using a $\mathsf{Verify}_{\mathsf{pk},h}$ gate taking two inputs r and s, or a $\mathsf{Verify}_{k,\mathsf{pk},h}$ taking two inputs ct_l and ct_r and verifying that their decryptions make a valid signature, or again a trivial circuit with two gates (a dummy gate ct and a hard-coded $\mathsf{valid}(k, .)$ gate), or an ad-hoc protocol.

C.3. Decryption Service

We could propose a pay-per-decryption service where there would be an established public key $\mathsf{pk} = \mathsf{sk} \cdot G$ and a secret key sk. The service V who holds sk would offer decryption in a form of a fair exchange: a client would submit a query Q and the contract would be to get R together with a proof that there exists sk verifying $\mathsf{pk} = \mathsf{sk} \cdot G$ and $R = \mathsf{sk} \cdot Q$ against a payment. This could be used as a decryption service for hybrid encryption based on the Diffie-Hellman protocol [11]. The proof is made by the Fiat-Shamir transform [17] of a Schnorr proof [31], [32] for discrete logarithm equality.

Namely, the item x would be $x = (R, \mathsf{ch}, \mathsf{resp})$ and $\mathsf{desc}_d(x)$ would be the predicate

$$\mathsf{ch} = H(G, \mathsf{pk}, Q, \mathsf{resp} \cdot G - \mathsf{ch} \cdot \mathsf{pk}, \mathsf{resp} \cdot Q - \mathsf{ch} \cdot R)$$

To generate x, V would compute $R = \mathsf{sk} \cdot Q$, select a random r, compute $A = r \cdot G$, $B = r \cdot Q$, $\mathsf{ch} = H(G, \mathsf{pk}, Q, A, B)$, and $\mathsf{resp} = r + \mathsf{ch} \cdot \mathsf{sk}$.

This decryption service can also be used as a service to issue one or multiple privacy pass tokens [10].

Defining the circuit is straightforward, subject to several strategic choices to trade the number of iterations in Step 7a against the amount of computations in the final Step 8.