Dispute Resolution for Smart Contract-based Two-Party Protocols

Eric Wagner, Achim Völker, Frederik Fuhrmann, Roman Matzutt, Klaus Wehrle
Communication and Distributed Systems
RWTH Aachen University
Aachen, Germany
{wagner,voelker,fuhrmann,matzutt,wehrle}@comsys.rwth-aachen.de

***Abstract*—Blockchain systems promise to mediate interactions of mutually distrusting parties without a trusted third party. However, protocols with full smart contract-based security are either limited in functionality or complex, with high costs for secured interactions. This observation leads to the development of protocol-specific schemes to avoid costly dispute resolution in case all participants remain honest. In this paper, we introduce SmartJudge, an extensible generalization of this trend for smart contract-based two-party protocols. SmartJudge relies on a protocol-independent mediator smart contract that moderates two-party interactions and only consults protocol-specific verifier smart contracts in case of a dispute. This way, SmartJudge avoids verification costs in absence of disputes and sustains interaction confidentiality among honest parties. We implement verifier smart contracts for cross-blockchain trades and exchanging digital goods and show that SmartJudge can reduce costs by 46–50% and 22 % over current state of the art, respectively.**

***Index Terms*—Ethereum, Bitcoin, smart contracts, two-party protocols, dispute resolution, cross-blockchain trades**

I. Introduction

Smart contracts enable trustworthy interactions between otherwise mutually distrusting parties by providing a technical replacement for intermediaries. Specifically, they provide a robust and use case-custom verification system combined with a transparent and immutable event log. However, these valuable building blocks for novel protocols and systems come with considerable drawbacks: First, complex verification within smart contracts is *costly* as each required operation inflicts monetary costs to protect the peers executing the smart contract from overloading. Secondly, users experience a *lack of privacy* since all interaction with a smart contract is being recorded on the underlying blockchain in order to achieve the desired transparency of the event log.

As a consequence of these drawbacks, we notice a shift of designs for smart contracts: A potpourri of recent works [1]–[5] either implicitly or explicitly propose to avoid expensive verification processes during two-party interactions unless there is a dispute whether or not both parties acted faithfully. This approach currently enables individual applications to reduce costs and increase privacy in the good case, as computations can be shifted off-chain and are only verified if one party disputes them. However, current approaches are tailored towards their respective application. Hence, while these works have a common pattern to improve their designs, seizing these saving potentials remains manual effort.

To remedy this situation, in this paper we propose to *generalize* conditional dispute resolution within smart contracts utilized by protocols executed by two mutually distrusting parties. We hence propose *SmartJudge*, an extensible framework for two-party protocols based on Ethereum smart contracts, which requires only minimal information and a security deposit from participating parties. At its core, SmartJudge consists of a general-purpose *mediator contract* that consults protocol-specific *verifier contracts* only in case of a dispute to protect a cheated participant. SmartJudge thereby enables protocol designers to minimize costs and maximize privacy against blockchain observers while providing full protection against misbehavior as long as the protocol generates a witness of its execution that is verifiable by a smart contract.

Costs can be minimized as honest parties can easily execute any protocol off-chain after an initial setup of preconditions and ultimately only have to acknowledge the other party's faithful execution on the blockchain. As long as both parties agree on the correctness of the execution, no on-chain verification is required. In our evaluation, we show that even protocols already using conditional conflict resolution can further profit when additionally using SmartJudge. Furthermore, SmartJudge maximizes achievable privacy against blockchain observers as details of the protocol execution only need to be disclosed in case of a dispute. If there is a dispute, i.e., either party claims to have been cheated on, then the mediator contract consults the protocol-specific verifier contract, which requires both participants to disclose cryptographic *protocol witnesses* to unambiguously judge which party misbehaved. The mediator contract then transfers the misbehaving party's security deposit, as well as a verifier-dependent fee, to the honest party as a penalty to disincentivize dishonest behavior.

We implement two use cases within SmartJudge. First, we realized cross-blockchain trades, which are currently being executed via hashed time-locked contracts (HTLCs) [6]. In contrast to HTLCs, SmartJudge performs potentially complex and thus expensive cryptographic checks only in case of a dispute. This way, we can reduce costs over HTLC-based approaches by 46–50%. Furthermore, we integrate FairSwap [3] into SmartJudge and thereby can reduce costs by roughly 22 %

Author manuscript.

 DOI: 10.1109/BLOC.2019.8751312

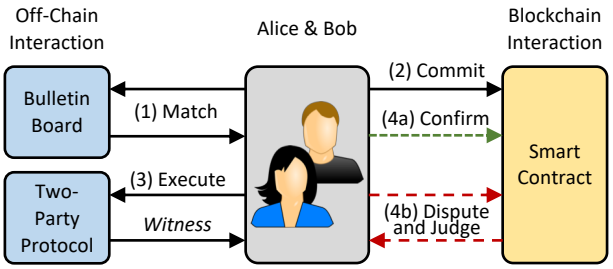

Fig. 1. Scenario for smart contract-based two-party protocols. Alice and Bob want to execute a specific, pre-negotiated protocol without necessarily knowing each other. As they are not entirely trusting each other, they want to rely on a smart contract to protect them in case a dispute arises. While this publicly records information on the blockchain, the smart contract resolves disputes fairly based on Alice's and Bob's protocol witnesses.

for honest parties, which shows that even two-party protocols that already perform conditional dispute resolution can benefit from using SmartJudge as an additional abstraction layer.

A. Main Contributions

In this paper, we make the following contributions to improve two-party protocols based on smart contracts:

- We *identify smart contract-based applications* that rely on two-party protocols, which can potentially benefit from further optimizations w.r.t. costs, privacy, and security.
- We *present SmartJudge*, our extensible framework for minimizing on-chain interaction for two-party smart contract-based protocols as a general-purpose solution to seize the identified potential for optimizations.
- We *implement verifier contracts* for cross-blockchain exchanges between Bitcoin and Ethereum as well as FairSwap [3] within our framework to show its feasibility.
- We *perform a cost evaluation* of our mediator contract as well as our two use cases to show that SmartJudge can both avoid expensive on-chain verification among honest parties and further reduce costs of protocols already performing protocol-specific conditional dispute resolution.

B. Paper Organization

The remainder of this paper is organized as follows. In Section II, we describe the assumed scenario for jointly executing smart contract-based two-party protocols. From the scenario, we derive a set of applications involving two-party protocols seizing the protection offered by smart contracts in Section III. Section IV subsequently introduces SmartJudge with a focus on the mediator contract at its core. We then describe the implementations and cost analyses for our verifier contracts for cross-blockchain exchanges in Section V and FairSwap in Section VI, respectively. In Section VII, we discuss related work and Section VIII concludes this paper.

II. Smart Contract-based Two-Party Protocols

In this paper, we consider protocols in which two parties, Alice and Bob, want to interact with each other over the Internet without necessarily knowing each other, e.g., to exchange digital goods. Hence, Alice and Bob do not necessarily trust each other. Bob can have an incentive to cheat during the interaction, e.g., as a musician he can try to lure Alice into digitally paying for his songs without ever giving her access to them. In general, we say that Alice and Bob want to jointly execute a *two-party protocol* in order to reach their respective goals, e.g., Alice wants to receive Bob's songs and Bob wants to be paid accordingly in our example.

Executing such a protocol in private proves risky for Alice and Bob as *disputes* may arise over whether or not the other party acted faithfully. When Alice and Bob communicate via a typical online service, then the service operator can resolve disputes by judging evidence provided by Alice and Bob, respectively. However, service providers are potentially biased and hence a fair and transparent dispute resolution cannot be guaranteed. In the most severe case, the service operator can even be Alice's counterpart of the two-party protocol, for instance when Alice buys goods from an online store.

Online users thus require means to resolve disputes in a *trustless manner*. Smart contracts promise to provide exactly this functionality and consequently smart contracts now constitute a well-used building block to act as a decentralized and thus independent mediator for two-party protocols. Examples comprise FairSwap [3], a protocol for exchanging digital goods, and CAIPY [4], a smart contract-based platform for processing car insurance claims. These and further related protocols such as Truebit [1] and Arbitrum [5] have in common that verification can become costly and are thus deferred to an *optional* dispute resolution phase instead of using built-in protocol-level protection via the utilized smart contracts.

From these observations, we derive the following general interaction pattern of Alice and Bob when jointly executing a smart contract-based two-party protocol, as shown in Figure 1. First, Alice and Bob *match* interests, i.e., they commit to jointly executing a specific protocol (Step 1). They typically negotiate the protocol to execute as well as necessary preconditions in private, for instance via an anonymous bulletin board (e.g., as in XIM [7]). After being matched, Alice and Bob both *commit* to executing the negotiated protocol via the smart contract (Step 2). In case of a dispute, this mutual commitment later on allows either party to prove that their counterpart agreed to perform the interaction as negotiated. Optionally, it can be required that Alice and Bob attach *security deposits* to their commitments to be deducted by the smart contract in case of misbehavior [8]. Subsequently, Alice and Bob execute the two-party protocol off-chain (Step 3). In order to enable the smart contract to resolve potential disputes, it is required that the protocol creates a *protocol witness* of its correct execution, i.e., Alice and Bob can only provide a sound protocol witness if they acted faithfully. After the execution, Alice and Bob either agree on their mutually faithful interaction or have a dispute. If both parties agree, they *confirm* this to the smart contract and attest that they will not contend their interaction in the future (Step 4a). Contrarily, in case of a dispute, i.e., one party misbehaves or stalls the protocol execution, the other party can contend the interaction and the smart contract has to resolve the dispute (Step 4b).

From now on, we consider this interaction pattern among Alice, Bob, and a mediating smart contract as the basic scenario for executing smart contract-based two-party protocols. In the next section, we derive applications involving such interactions and thus could benefit from improved mediation.

III. Potential Use Cases

Based on recent works in the field of smart contracts, we derive applications involving two-party protocols that potentially benefit from utilizing a smart contract-based mediator. Namely, potential use cases comprise cross-blockchain exchanges, trading digital goods, safety for Bitcoin transactions, two-party mixing of cryptocurrencies, off-chain execution of smart contracts, and physical-state verification.

Cross-Blockchain Exchanges. Over the past years, Bitcoin has inspired the creation of many other cryptocurrencies. At the time of writing, CoinMarketCap tracks the respective market capitalization of over 2000 cryptocurrencies [9]. To mitigate the fragmentation of cryptocurrency assets, cross-blockchain trading emerged. However, trading parties must ensure that neither party can receive funds without completing the exchange [10], especially when trading across cryptocurrency borders. Current approaches such as TierNolan's protocol [11] or BarterDEX [12] utilize HTLCs to achieve *atomic swaps*, i.e., payments are locked and inevitably redeemable as soon as the seller concluded their part of the trade. Alternatively, Coincer [13] relies on a peer-to-peer overlay to facilitate trustless exchanges.

Trading Digital Goods. Smart contracts have been deployed to control access to digital goods in a decentralized manner, e.g., as a platform for digital music [14]. In this setting, Alice wants to buy a digital good from Bob, where (i) Alice wants the guaranty that she can access her purchased good after her payment and (ii) Bob only grants access after receiving Alice's payment. While this fairness approach is well-studied [15]–[18], corresponding protocols had to rely on a trusted third party [19]. Introducing cryptocurrency deposits, however, allowed for secure, decentralized payment escrows [20], optionally with only conditional conflict resolution [21]. Finally, FairSwap [3] provides a general framework for the smart contract-based trading of digital goods.

Safety for Bitcoin Transactions. Bitcoin transactions need roughly one hour to be confirmed. While this is prohibitive for everyday transactions, Bitcoin remains the most widely accepted cryptocurrency for real-world payments [22]. If Alice's and Bob's two-party protocol simply consists of Alice paying her coffee in Bitcoin, Alice can additionally back up her payment via an Ethereum smart contract as this transaction is expected to be verified much faster. While this additional overhead seems counter-intuitive at first, it can provide additional safety for merchants in case of one-time customers, i.e., whenever micropayment channels [23] are infeasible to set up.

Two-Party Mixing. As evidenced by various analyses [24]–[26], cryptocurrencies without privacy-enhancing extensions do not provide sufficient anonymity against blockchain observers. Contrary to other distributed mixing approaches [27]–[30], XIM [7] proposes to mix bitcoins via a two-party protocol that allows Alice and Bob to securely and anonymously mix by a combination of binding on-chain interaction and private off-chain negotiation facilitated by Tor hidden services locally operated by both parties, respectively. Thereby, XIM charges additional transaction fees to incentivize Alice and Bob to act faithfully, but also relies on an on-chain protocol for fair exchange similar to HTLCs [10]. Two-party mixing follows the interaction pattern described in Section II and we see the potential to reduce the required on-chain operations.

Off-Chain Execution of Smart Contracts. A recent shift in managing smart contracts proposes to execute smart contracts off-chain and leave the result open for optional verification in case of a dispute [1], [5]. While this optional on-chain verification of individual computation steps of a smart contract today comes with significant overhead, it enables smart contracts to perform more complex and less redundant computations.

Physical-State Verification. To a limited extent, smart contracts can also be used to process real-world events. Recently, CAIPY [4] proposed to use tamper-resistant sensors in cars to confidentially record information about insurance-related events on the Ethereum blockchain. The respective smart contract can then also perform rudimentary verification of the soundness of reported events. Similarly, Slock.it [31] has the vision to connect arbitrary smart devices to the blockchain. We thus believe that physical devices have the potential to assume a crucial role in opening up further classes of secure two-party protocols as long as these devices can produce verifiable protocol witnesses as described in Section II.

The wide range of potential applications motivates our work to minimize costs for two-party protocols. In the following, we thus propose a general framework to achieve this goal.

IV. A Mediation Framework for Ethereum

We now present SmartJudge, our extensible framework for smart contract-based two-party protocols as defined in Section II to minimize protocol overhead via conditional dispute resolution without forfeiting the security of the protocol. We first state our design goals (Section IV-A) and then give an overview of SmartJudge's design (Section IV-B). Next, we describe SmartJudge's core element, the mediator contract (Section IV-C), and finally discuss its costs (Section IV-D).

A. Design Goals

We identify the following design goals for extending two-party protocols with efficient, optional dispute resolution:

Decision Correctness. Dispute resolution must detect and punish dishonest behavior by either party at any time.

Cost Minimization. We seek to minimize costs for honest parties even if a dishonest party necessitates dispute resolution.

Protocol Confidentiality. The details of Alice's and Bob's negotiation process and protocol execution shall remain private as long as both parties act honestly.

Extensibility. Any framework for dispute resolution must remain extensible to cope with different and novel protocols such as the potential applications outlined in Section III.

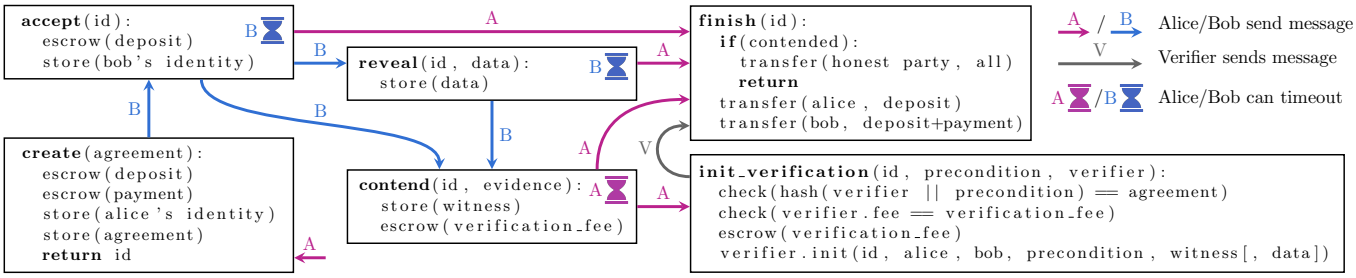

Fig. 2. Overview of SmartJudge's mediator contract. Alice creates a new agreement, submits her security deposit, and optionally escrows a payment for Bob. Bob commits the agreement by accepting it, simultaneously submitting his security deposit. Then Bob provides his service off-chain (optionally with on-chain proof). Now, either Alice finishes the interaction or Bob can contend the interaction in case he provided the service without Alice's acknowledgement. If Alice disagrees with the contention, she can invoke the initially agreed-upon verifier contract to enforce a decision in a trustless manner.

In the following, we detail how our proposed framework SmartJudge achieves these design goals.

B. Design Overview of SmartJudge

At the core of SmartJudge resides the *mediator contract*, a smart contract to handle on-chain interactions between Alice and Bob independently from their agreed-upon two-party protocol. First, Alice and Bob negotiate the protocol they want to execute and its respective preconditions. They commit to these terms by submitting a confidentiality-protecting *negotiation witness* and a *security deposit* to the mediator contract. This witness includes the address of a *verifier contract*, a protocol-specific smart contract to unambiguously judge disputes. This way, SmartJudge remains extensible to mediate the execution of novel two-party protocols. The verifier contract relies on the creation of *protocol witnesses* that it can evaluate in order to prove a party's faithful execution of protocol-specific actions outside the scope of the mediator contract (cf. Section II).

If Alice and Bob agree that both of them executed the chosen protocol faithfully, they publish a *confirmation* to the mediator contract, which then returns their security deposits. Otherwise, the execution is contended and the mediator contract consults the agreed-upon verifier contract. Alice and Bob then have to submit their protocol witnesses to the verifier contract. The verifier contract will reveal the dishonest party based on these witnesses and then instructs the mediator contract to reimburse the honest party with *both* security deposits while simultaneously punishing the dishonest party.

C. The Mediator Contract

The *mediator contract* is the core component of SmartJudge. The mediator contract is a general-purpose smart contract on the Ethereum blockchain that manages on-chain interaction between Alice and Bob when they seek to jointly execute a two-party protocol in a secure manner. By abstracting away any protocol-specific structure from the mediator contract, we enable Alice and Bob to record a transcript of their interaction irrevocably on the blockchain at minimal costs and information leakage. As long as both parties remain honest, the mediator contract only confidentially records which verifier contract must be consulted in case of a dispute and it records an immutable witness of Alice's and Bob's negotiated parameters. When committing to executing a protocol, the mediator contract requires Alice and Bob to submit a *security deposit* that is transferred back to the participants after successful interaction. If a party contends the interaction, then the mediator contract consults the specified verifier contract and reimburses the cheated party with *both* security deposits and verifier-dependent fees based on its decision.

Figure 2 shows the interaction with the mediator contract in greater detail. As described in Section II, we assume that Alice and Bob negotiated a two-party protocol they want to execute and its required parameters in advance, e.g., via a bulletin board. Without loss of generality, we now assume that Alice instigates on-chain interaction with the mediator contract. Furthermore, we assume in this section that Alice's and Bob's two-party protocol involves that Bob has to provide a service that Alice pays for in ether. By allowing Bob to submit a higher security deposit than Alice, SmartJudge can also generalize to arbitrary two-party protocols without payment by Alice, in which she is compensated if Bob misbehaves.

Alice first creates a new *agreement*, in which she publishes a *negotiation witness* to the mediator contract. The negotiation witness contains a hash value over the verifier contract's address and further protocol-specific parameters from her perspective. Storing only a hash value prevents blockchain observers from deducing parameters from the negotiation witness, e.g., the volume of a trade between Alice and Bob.

Bob then commits to the agreement by accepting it. Thereby, Bob escrows the same security deposit as Alice and accepts Alice's protocol parameters. This way, Alice and Bob publicly record that they agreed on the terms of their interaction without disclosing any private information.

Next, Bob must provide his service to Alice as specified in Alice's negotiation witness. Once Alice recognizes that Bob fulfilled his part of the agreement, she can gracefully conclude their interaction by publishing a confirmation to the mediator contract. If the agreed-upon protocol requires on-chain evidence of Alice's and Bob's interaction, e.g., Bob permanently transfers a digital good to Alice [3], Bob can also reveal the required data as further proof before Alice confirms the interaction's conclusion.

However, up to now the mediator contract only covers the good case in which Alice and Bob both act honestly. In this

case, the mediator contract shall minimize costs and maximize privacy. However, both parties choose to interact involving a smart contract in the first place because they do not entirely trust each other and choose to rely on the mediator contract for security in case of a dispute. The mediator contract must ensure that an honest party can always progress towards a graceful conclusion of the interaction, or that the verifier contract identifies and punishes the dishonest party. It can either happen that Bob refuses to provide his service, or that Alice refuses to acknowledge that Bob did provide the service.

If Bob refuses to provide his service as agreed, Alice will not conclude the interaction. This either forces Bob to reveal a fake protocol witness, which can easily be debunked as Alice can subsequently invoke the verifier contract, or he never responses. In the former case, Alice has to disclose the address of the verifier contract and the agreed-upon parameters publicly in order to enable verification. In the latter case, Alice can finish the interaction after an agreed-upon timeout and thereby collect Bob's security deposit.

If Bob provided his service, but Alice refuses to acknowledge this, Bob can prove his faithfulness by publishing his protocol witness. Afterwards, Alice has only the options to either confirm Bob's protocol witness or to contend it and thereby initiate its on-chain verification. After verifying Bob's correct protocol witness, the verifier contract reimburses Bob with both security deposits. Bob can further collect Alice's security deposit after a timeout period in case she attempts to stall the verification process.

We deliberately accept that verifier contracts can be very complex and thus costly. In Sections V and VI, we discuss verifier contracts for two use cases and their associated costs, respectively. The main principle of SmartJudge is that honest parties can bypass *all* costs for securing their interaction while still being able to rely on full verifiability if deemed necessary. Yet, interaction with the verifier contract can cause asymmetric additional costs. To account for this, both parties need to deposit a verifier-dependent verification fee once the verifier contract shall be involved. Insufficient verification fees are detected once the verifier contract is revealed, causing the underpaying party to lose the dispute.

Having Alice and Bob agree on a specific verifier contract to back their interaction makes SmartJudge highly extensible. We only require that all verifier contracts share a common interface. However, in order to keep extremely complex verification operations off-chain, Alice and Bob can also involve a mutually accepted trusted third party to overlook their interaction within the same framework as on-chain verification. This can be achieved by releasing a verifier contract that accepts human decisions by a dedicated authority such as a notary, potentially in exchange for a monetary reward.

Concluding, Alice and Bob can be sure that their interaction is overlooked properly due to their agreed-upon verifier contract. Furthermore, the mediator contract's design allows Alice and Bob to interact at protocol-independent low costs and according to confidential terms in case that Alice and Bob remain honest as they can avoid costly verification. Finally, negotiating the verifier contract keeps SmartJudge extensible. Hence, we argue that SmartJudge fulfills our design goals set out in Section IV-A. In the following, we quantify the actual costs introduced by using SmartJudge's mediator contract in the good case where Alice and Bob are honest.

TABLE I
COSTS OF INTERACTING WITH THE MEDIATOR CONTRACT

Function	**Caller**	**Costs**		
		Gas Costs	$\mathcal{A}$ [USD]	$\mathcal{C}$ [USD]
`create`	Alice	135 445	0.20	0.18
`create_reuse`	Alice	55 703	0.08	0.07
`abort`	Alice	36 080	0.05	0.05
`accept`	Bob	43 321	0.06	0.06
`reveal`	Bob	66 112*	0.10	0.09
`finish`	Alice	43 720	0.07	0.06
`contend`	Bob	97 193*	0.15	0.13
`init_verif.`	Alice	121 192	0.18	0.16
`timeout`	Alice/Bob	36 112	0.05	0.05
`reg._verif.`	anybody	69 686	0.10	0.09

* Split gas costs — $\mathcal{A}$: average model; $\mathcal{C}$: current model

D. Evaluation of Mediator Contract Costs

To evaluate SmartJudge, we implemented the mediator contract for Ethereum[1] and analyze its costs.

Methodology. We measure the costs for using SmartJudge (in USD) to assess the efficiency of our framework. To account for price volatility, we calculate costs w.r.t. two different price models, the *average model* $\mathcal{A}$ and the *current model* $\mathcal{C}$. In $\mathcal{A}$, we fix an ether price of 500 USD and a gas price of 3 GWei. These costs roughly correspond to the average recommendations during the summer of 2018 and enable us to compare our costs with the analysis of [3] in Section VI. In the *current model* $\mathcal{C}$, we use the recent ether price of 94.74 USD [9] and gas costs of 13.8 GWei [33] as of 12/19/2018.

Cost Analysis. The mediator contract can easily be deployed to the Ethereum blockchain costing 1 947 000 gas ($\mathcal{A}$ / $\mathcal{C}$: 2.92 / 2.55 USD, limit per block: 8 000 000 gas). The costs for interacting with the mediator contract are shown in Table I. Single-use interaction of honest Alice and Bob costs them 222 000 gas ($\mathcal{A}$ / $\mathcal{C}$: 0.33 / 0.29 USD). These costs mostly stem from agreement creation. Hence, we further reduce the costs by allowing Alice and Bob to reuse previously allocated storage, reducing the total costs of the good case to 143 000 gas ($\mathcal{A}$ / $\mathcal{C}$: 0.21 / 0.19 USD). If Bob needs to transfer data on-chain, this costs 66 000 gas ($\mathcal{A}$ / $\mathcal{C}$: 0.10 / 0.09 USD) for 32 B of data. In total, contending an interaction can inflict costs of up to 338 000 gas ($\mathcal{A}$ / $\mathcal{C}$: 0.51 / 0.44 USD) on Alice and 125 000 gas ($\mathcal{A}$ / $\mathcal{C}$: 0.19 / 0.16 USD) on Bob, which motivates an initial security deposit of 400 000 gas for both.

Our analysis shows that the mediator contract only inflicts low costs. To account for price volatility, the security deposits are scaled based on the current gas price to ensure that honest parties are fully reimbursed even when exchange rates peak.

V. TRUSTLESS EXCHANGE OF DIGITAL CURRENCIES

In this section, we describe and evaluate our verifier contract for *secure cross-blockchain trades* using trades be-

[1] Code available at: https://github.com/COMSYS/smartjudge

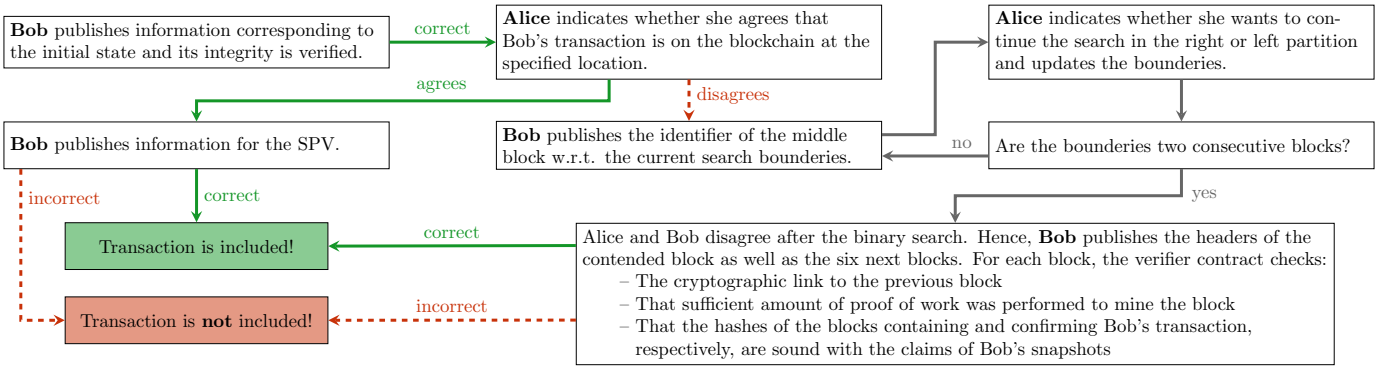

Fig. 3. Simplified interactive process to verify the existence of a Bitcoin transaction via an Ethereum smart contract. If Alice and Bob agree that the block of interest is part of Bitcoin's blockchain, the verifier contract can perform Simple Payment Verification (SPV) [32] to verify Bob's transaction on that block. Otherwise, Alice and Bob perform a binary search on Bob's view of the Bitcoin blockchain to derive the block containing the transaction. The verifier contract then verifies that this block is confirmed according to Bitcoin's rules and judges in favor of Bob if this is the case, and in favor of Alice otherwise.

tween Ethereum and Bitcoin as an example. Our approach is, however, extensible to other cryptocurrencies as long as an Ethereum smart contract can decide whether or not a specific transaction is included in the remote blockchain. In principle, this approach also enables trading between two remote cryptocurrencies in a trustless manner by escrowing ether, e.g., to trade cryptocurrencies without original support for cross-blockchain trades.

We first describe our verifier contract for the existence of transactions on the Bitcoin blockchain (Section V-A) and then argue how our approach brings improvements over using HTLCs to escrow funds (Section V-B).

A. An Efficient Verifier Contract for Bitcoin Transactions

In this section, we describe the design and implementation of our verifier contract that decides whether or not a given transaction exists on the Bitcoin blockchain. In this use case, Alice wants to trade her ether for Bob's bitcoins. The key challenge here is that such a verifier contract cannot interact with the Bitcoin blockchain directly. Instead, if there is a dispute among Alice and Bob whether Bob correctly sent his bitcoins to Alice, our verifier contract relies on claims by both parties reflecting their *individual views* on the Bitcoin blockchain. These claims constitute our protocol witnesses and can only be faked by parties that can attack Bitcoin mining.

Figure 3 shows how the verifier contract works and which interaction it requires from Alice and Bob. Our verifier contract first determines whether the block allegedly containing Bob's transaction is accepted by Alice and then uses either simple payment verification (SPV) [32] to verify that this block contains Bob's transaction or a binary search to identify who is dishonest about the state of the blockchain. This way, we aim to minimize the number and size of messages Alice and Bob have to send to the verifier contract.

Before engaging in their trade, Alice and Bob negotiate its conditions as well as a *snapshot* of Bitcoin's blockchain and submit a hash value of the snapshot to the mediator contract as part of their commitment. The conditions specify Alice's Bitcoin address and how many bitcoins Bob must send her. The snapshot consists of the identifier of a recent Bitcoin block as well and a lower bound of its mining difficulty[2]. Similarly, after his transaction was added to Bitcoin's blockchain, Bob shares the hash value of another snapshot with the mediator contract to prove *confirmation* of his transaction. This snapshot contains the identifier of the block that confirms Bob's Bitcoin transaction (i.e., the sixth-next block) and the number of blocks mined since the initial snapshot. In case of a dispute, this snapshot constitutes evidence that his Bitcoin transaction has been confirmed based on the agreed-upon initial snapshot.

If Alice refuses to acknowledge Bob's transaction, then he contends the interaction (cf. Section IV) and publishes his protocol witness to prove that his transaction has been confirmed based on the initial snapshot. To this end, he releases (i) the full information for the initial and later snapshot and (ii) the identifier of the block containing his transaction.

If Alice agrees that Bob's claimed block is on the blockchain, the verifier contract immediately checks that Bob's transaction is part of this block using SPV. Bob then has to publish his transaction, the block's header, and the hash values of the block's Merkle tree needed to perform SPV. The verifier contract then verifies that Bob's information is sound w.r.t. the agreed-upon trade terms and decides the dispute accordingly.

If, however, Alice claims that Bob submitted an invalid block identifier, the verifier contract must decide whether or not Bob's claim was correct. At this point, the verifier contract knows a valid block from the initial snapshot and can verify the relative validity of the confirming block stated in Bob's later snapshot. However, there must be some block in between that Alice and Bob do not agree on. Hence, Alice and Bob perform an interactive binary search via the verifier contract to efficiently find this point of disagreement. Alice challenges Bob to upload new block identifiers until the binary search terminates. Bob then publishes the block headers of the determined and the six following blocks to the verifier contract. The verifier contract can then verify whether or not Bob's submitted later snapshot is sound by checking the integrity and proof of work of his submitted block headers. To limit the number of verification steps, we cap the maximum

[2]The verifier contract uses this lower bound to validate that the transaction was confirmed using sufficient proof of work

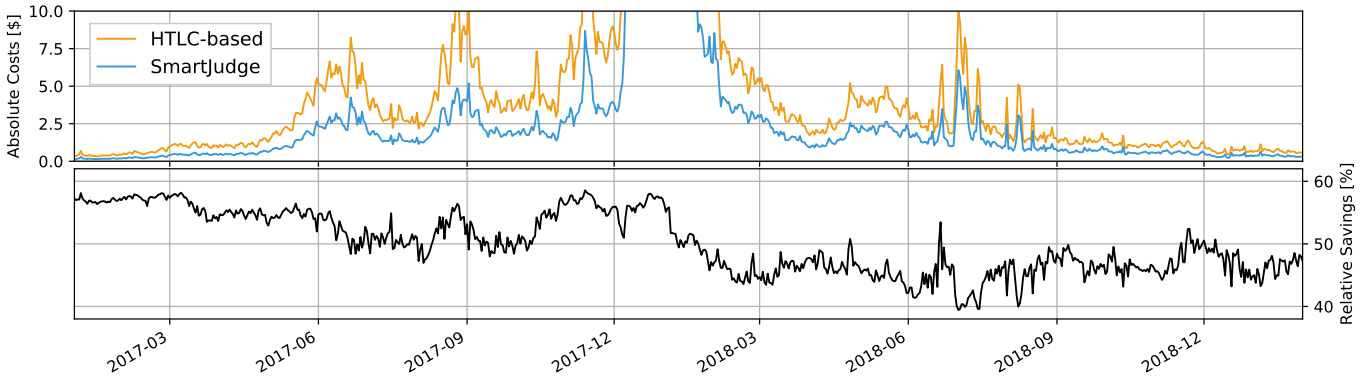

Fig. 4. Cost savings of SmartJudge over HTLC-based atomic swaps. The top graph shows the absolute costs according to average exchange rates and fees of Ethereum [33] and Bitcoin [34], respectively. For readability purposes, we omitted the peaks around January 2018 (28.82 USD and 68.17 USD), but additionally show the relative cost savings of SmartJudge over HTLC-based atomic swaps over the whole considered duration in the bottom graph.

time between the initial and Bob's later snapshots to two weeks (roughly 2016 blocks), which results in worst-case verification costs of 1 343 000 gas ($\mathcal{A}$ / $\mathcal{C}$: 2.01 / 1.76 USD).

While the required interaction by Alice and Bob seem infeasible at first, the interactive binary search only inflicts moderate costs and the honest party will be reimbursed accordingly. As the verifier contract will unveil dishonest behavior, however, trading partners are incentivized to remain honest, which allows bypassing *all* verification overhead.

B. Comparison of SmartJudge and HTLC-based Approaches

As discussed in Section III, cross-blockchain trades are mainly performed via HTLC-based atomic swaps. Hence, we evaluate our verifier contract for Bitcoin transactions w.r.t. costs, address linkability, fairness, and versatility.

Cost Analysis. As discussed in Section IV-D, honest parties can conclude an interaction at costs of 143 000 gas ($\mathcal{A}$ / $\mathcal{C}$: 0.21 / 0.19 USD). Still, for cross-blockchain trades we also have to take costs on the remote blockchain, i.e., Bitcoin, into account. Similarly to our approach in Section IV-D, we assume a bitcoin price of 6500 USD and fees of 10 sat/B for the average model $\mathcal{A}$ and a price of 3215.21 USD with fees of 13 sat/B for the current model $\mathcal{C}$ as of 12/19/2018 [9], [35].

We now compare these costs with HTLC-based approaches, specifically as proposed by BIP 199 [6]. The hash-locking and unlocking transactions there have an average size of 475 B. As they have one input and output, respectively, we mimic the same behavior when considering Bob's standard Bitcoin transaction, which we fixed at a size of 191 B and which would cost fees of 0.12 USD ($\mathcal{A}$) respective 0.08 USD ($\mathcal{C}$). Considering the typical output for change only accounts to a constant overhead for both approaches. To simulate HTLCs in Ethereum, we use the smart contract of Altcoin.io [36]. As concluding a trade in our framework then yields costs of 232 000 gas ($\mathcal{A}$ / $\mathcal{C}$: 0.35 / 0.30 USD), we conclude that we can reduce the costs for cross-blockchain trades by around 50 % over current HTLC-based approaches.

To further investigate the relations between HTLC-based approaches and SmartJudge, we also consider the long-term evolution of Ethereum's and Bitcoin's market prices since 2017 in Figure 4. Again, SmartJudge can reduce costs over HTLC-based approaches by 50 % on average, with a slight decrease to a 46 % cost reduction in 2018. This is a result of Ethereum's increased popularity relative to Bitcoin, which we identify as an important factor to further judge the saving potential of our approach for cross-blockchain trades. In fact, cost savings of SmartJudge over HTLC-based approaches are maximized if the remote blockchain has higher exchange rates compared to Ethereum. However, SmartJudge vastly outperforms HTLC-based approaches even though we observed the opposite trend during our price analysis.

Address Linkability. HTLC-based approaches enable linking the involved parties' addresses as they cryptographically link the transactions across blockchains with a distinctive pattern. Using SmartJudge, contrarily, Bob only has to announce a standard transaction on the remote blockchain. As long as both parties remain honest, Alice and Bob do not have to reveal any information about the executed protocol.

Fairness. HTLC-based atomic swaps are asymmetric in that only Alice can back out of an agreed-upon trade. Hence, only she can base her decision to conclude or abort a trade based on the development of exchange rates. Using SmartJudge, both parties can back out of the trade until Bob confirms its conditions. After both parties committed to the trade, SmartJudge will punish non-compliance. Hence, SmartJudge achieves fairness for cross-blockchain trades w.r.t. price volatility.

Versatility. To use HTLC, the bridged cryptocurrencies need to support the same primitives. Although this is also beneficial for SmartJudge, SmartJudge remains more resilient to situations where the verifier contract has to emulate features of the remote blockchain. While we expect such emulation to quickly become expensive, SmartJudge again incentivizes parties to be honest and thus avoid such costs.

VI. Selling Digital Goods via SmartJudge

Another use case for smart contract-based two-party protocols is to immutably manage access to digital goods such as music via the blockchain. As outlined in Section III, Alice wants to buy some digital good from Bob, but neither party may be able to abort the transaction prematurely to rip off their counterpart. A recent solution to this problem is FairSwap [3]. By integrating FairSwap into our general-purpose framework

SmartJudge (Section VI-A), we can show that SmartJudge can even reduce costs for protocols that already deploy protocol-specific conditional conflict resolution (Section VI-B).

A. Integration of FairSwap Into SmartJudge

In FairSwap [3], Bob offers to sell a set of files to interested parties. If Alice is interested in buying a file from Bob, he sends her an encrypted version of the file and constructs a Merkle tree over the file and sends its root to Alice. This constitutes the initial protocol witness based on which Bob can later prove that he sent Alice the correct file. Bob submits this information to the smart contract and Alice accepts this by escrowing her payment. Furthermore, Bob publishes metadata for the file as well as a cryptographic commitment which enables him to later prove that he sent Alice the correct key.

SmartJudge can cover this general protocol flow with only minor adjustments. In fact, SmartJudge can act as an abstraction layer for FairSwap since Alice and Bob only publish hash values over their negotiated terms. Once Bob accepted Alice's request via the mediator contract, he sends her the required decryption key off-chain. If Alice refuses to conclude the trade, Bob reveals the decryption key as his protocol witness to prove his faithfulness in combination with the negotiated terms and the encrypted file. Hence, Alice either has to conclude or contend the trade, which would determine Bob's faithfulness.

B. Advantages of Using SmartJudge

Introducing SmartJudge as a layer of abstraction on top of FairSwap has several adavantages. First, SmartJudge inflicts much lower costs if Alice and Bob remain honest. The authors report costs of $1.60\,\mathrm{USD}$ for using FairSwap [3]. Based on the provided prototype implementation, we obtained respective costs of $174\,000\,\mathrm{gas}$ ($\mathcal{A}$ / $\mathcal{C}$: 0.26 / 0.23 USD) for using FairSwap. Despite these vastly lower costs, SmartJudge still reduces costs by $22\,\%$ over vanilla FairSwap. Even though our integration of FairSwap into our framework yields worst-case costs of $271\,000\,\mathrm{gas}$ ($\mathcal{A}$ / $\mathcal{C}$: 0.41 / 0.35 USD) when invoking the verifier contract, parties are incentivized to avoid this case and a honest party would be reimbursed. Secondly, SmartJudge can handle multiple trades in parallel, which would require multiple deployments of FairSwap. Thirdly, incentives to remain honest have been discussed as a future extension of FairSwap and are a core feature of SmartJudge. Finally, the biggest advantage might be that SmartJudge allows honest parties to trade digital goods confidentially without revealing any information about the terms or even that a trade took place (in case that multiple verifiers are deployed).

VII. Related Work

Several other works consider conditional dispute resolution to reduce the costs of smart contract-based verification. CAIPY [4] proposes to integrate smart contracts judging event readings of tamper-proof sensors into car insurance-related processes to avoid costly manual inspection where possible. While CAIPY's approach is highly application-specific, the anticipation of verifier contracts relying on tamper-proof data readings promises a further design space for SmartJudge-based protocols. FairSwap [3] applies a similar pattern as ours to the use case of general trading of digital goods. In Section VI, we showed that costs for using FairSwap can be further reduced by relying on SmartJudge as an additional abstraction layer. Truebit [1] is more closely related to SmartJudge. Truebit also proposes to use verifier contracts, albeit in the form of web assembly code that is partially interpreted on-chain in case of a dispute. While this approach allows for more complex verification, it comes with significant overhead. Furthermore, Truebit does not allow for user-submitted protocol witnesses, which limits the scope of realizable use cases. While the limited complexity of SmartJudge's verifier contracts may also limit its applicability, we believe that Truebit and SmartJudge can complement each other nicely. Finally, Arbitrum [5] outsources the execution of smart contracts to special virtual machines whose execution traces can be verified on-chain. A small set of trusted managers can investigate the state of a virtual machine to ease verification. We envision that SmartJudge can also complement Arbitrum's approach with a suitable verifier contract. While being only suitable for verifying single computation steps, SmartJudge maintains its low costs for honest parties, increases achievable privacy by not revealing the initial setup and has simpler integration of interaction with the participants during the protocol execution.

VIII. Conclusion

We presented SmartJudge, a general framework for the efficient and secure moderation of two-party protocols using Ethereum smart contracts. Our design involves minimal interaction by honest users to reduce costs and to preserve confidentiality of the negotiated terms of the respective protocols. Furthermore, SmartJudge incentivizes parties to remain honest by consulting protocol-specific smart contracts in case of a dispute to identify and punish any dishonest party.

To showcase the applicability of SmartJudge, we implemented and evaluated two use cases. First, we replaced hashed time-locked contracts with SmartJudge for cross-blockchain trades between Ethereum and Bitcoin. Our rationale is that current solutions cryptographically protect also trades among honest users and, as a consequence, inflict avoidable costs on them. Our evaluation shows that SmartJudge can reduce costs by 46–50% on average compared to current state of the art. Secondly, we integrated FairSwap, a protocol for trading digital goods, into SmartJudge. Even though FairSwap already does conditional conflict resolution we still reduced costs by $22\,\%$ when combining it with SmartJudge.

Because of these promising results, we believe that SmartJudge can provide a valuable building block for further applications, parts of which we already outlined in this paper.

Acknowledgments

This work has been funded by the German Federal Ministry of Education and Research (BMBF) under funding reference numbers 16KIS0443 and 16DHLQ013. The responsibility for the content of this publication lies with the authors.

References

[1] J. Teutsch, L. Luu, and C. Reitwiessner, "A scalable verification solution for blockchains," 2017, last accessed: 03/12/2019. [Online]. Available: https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf

[2] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2017.

[3] S. Dziembowski, L. Eckey, and S. Faust, "FairSwap: How To Fairly Exchange Digital Goods," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CSS)*. ACM, 2018.

[4] L. Bader, J. C. Bürger, R. Matzutt, and K. Wehrle, "Smart Contract-based Car Insurance Policies," in *Proceedings of the 2018 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2018.

[5] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, "Arbitrum: Scalable, Private Smart Contracts," in *Proceedings of the 27th USENIX Security Symposium (USENIX Security)*. USENIX Association, 2018.

[6] S. Bowe and D. Hopwood. (2017) BIP 199: Hashed Time-Locked Contract Transactions. Last accessed: 03/12/2019. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0199.mediawiki

[7] G. Bissias, A. P. Ozisik, B. N. Levine, and M. Liberatore, "Sybil-Resistant Mixing for Bitcoin," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2014.

[8] K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi, "Step by Step Towards Creating a Safe Smart Contract: Lessons and Insights from a Cryptocurrency Lab," in *Proceedings of Financial Cryptography and Data Security 2016 (FC)*. Springer, 2016.

[9] CoinMarketCap. (2013) Cryptocurrency Market Capitalizations. Last accessed: 03/12/2019. [Online]. Available: https://coinmarketcap.com

[10] S. Barber, X. Boyen, E. Shi, and E. Uzun, "Bitter to Better—How to Make Bitcoin a Better Currency," in *Proceedings of Financial Cryptography and Data Security 2012 (FC)*. Springer, 2012.

[11] TierNolan. (2013) Alt chains and atomic transfers. Last accessed: 03/12/2019. [Online]. Available: https://bitcointalk.org/index.php?topic=193281.0

[12] "Komodo – Advanced Blockchain Technology, Focused on Freedom," 2018, last accessed: 03/12/2019. [Online]. Available: https://komodoplatform.com/wp-content/uploads/2018/06/Komodo-Whitepaper-June-3.pdf

[13] M. Zima, "Coincer: Decentralised Trustless Platform for Exchanging Decentralised Cryptocurrencies," in *Proceedings of the 11th International Conference on Network and System Security (NSS)*. Springer, 2017.

[14] ConsenSys AG. (2018) UJO Music. Last accessed: 03/12/2019. [Online]. Available: https://ujomusic.com

[15] H. Bürk and A. Pfitzmann, "Value Exchange Systems Enabling Security and Unobservability," *Computers & Security*, vol. 9, no. 8, 1990.

[16] B. Cox, J. D. Tygar, and M. Sirbu, "NetBill Security and Transaction Protocol," in *Proceedings of the 1st conference on USENIX Workshop on Electronic Commerce (WOEC)*. USENIX, 1995.

[17] J. Zhou and D. Gollman, "A Fair Non-Repudiation Protocol," in *Proceedings of the 1996 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1996.

[18] F. Bao, R. H. Deng, and W. Mao, "Efficient and Practical Fair Exchange Protocols With Off-line TTP," in *Proceedings of the 1998 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1998.

[19] H. Pagnia and F. C. Gärtner, "On the Impossibility of Fair Exchange Without a Trusted Third Party," Department of Computer Science, Darmstadt University of Technology, Tech. Rep., 1999.

[20] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," in *Proceedings of the 34th Annual Cryptology Conference (CRYPTO)*. Springer, 2014.

[21] D. Jayasinghe, K. Markantonakis, and K. Mayes, "Optimistic fair-exchange with anonymity for bitcoin users," in *Proceedings of the 2014 IEEE 11th International Conference on e-Business Engineering (ICEBE)*. IEEE, 2014.

[22] VirtualCoinSquad. (2018) Businesses that Accept Bitcoin, Litecoin, Dogecoins and other Altcoins in 2018. Last accessed: 03/12/2019. [Online]. Available: https://www.virtualcoinsquad.com

[23] P. McCorry, M. Möser, S. F. Shahandasti, and F. Hao, "Towards Bitcoin Payment Networks," in *Proceedings of the 21st Australasian Conference on Information Security and Privacy (ACISP)*. Springer, 2016.

[24] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A Fistful of Bitcoins: Characterizing Payments Among Men With No Names," in *Proceedings of the 2013 Conference on Internet measurement Conference*. ACM, 2013.

[25] P. Koshy, D. Koshy, and P. McDaniel, "An Analysis of Anonymity in Bitcoin Using P2P Network Traffic," in *Proceedings of Financial Cryptography and Data Security 2014 (FC)*. Springer, 2014.

[26] A. Biryukov and I. Pustogarov, "Bitcoin Over Tor Isn't a Good Idea," in *Proceedings of the 2015 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2015.

[27] G. Maxwell, "CoinJoin: Bitcoin privacy for the real world," 2013, last accessed: 03/16/2019. [Online]. Available: https://bitcointalk.org/index.php?topic=279249

[28] J. H. Ziegeldorf, F. Grossmann, M. Henze, N. Inden, and K. Wehrle, "CoinParty: Secure Multi-Party Mixing of Bitcoins," in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy (CODASPY)*. ACM, 2015.

[29] J. H. Ziegeldorf, R. Matzutt, M. Henze, F. Grossmann, and K. Wehrle, "Secure and Anonymous Decentralized Bitcoin Mixing," *Future Generation Computer Systems (FGCS)*, vol. 80, 3 2018.

[30] S. Meiklejohn and R. Mercer, "Möbius: Trustless Tumbling for Transaction Privacy," *Proceedings on Privacy Enhancing Technologies (PoPETs)*, vol. 2018, no. 2, 2018.

[31] slock.it. (2018) Incubed. Last accessed: 03/12/2019. [Online]. Available: https://slock.it/iotlayer.html

[32] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008, last accessed: 03/12/2019. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[33] Etherscan. (2018) Ethereum (eth) blockchain explorer. Last accessed: 03/12/2019. [Online]. Available: https://etherscan.io

[34] Bitcoin.com. (2018) Bitcoin core charts. Last accessed: 03/12/2019. [Online]. Available: https://charts.bitcoin.com/btc

[35] Earn.com. Bitcoin Fees for Transactions. Last accessed: 03/12/2019. [Online]. Available: https://bitcoinfees.earn.com

[36] Altcoin.io. (2017) Ethereum Atomic Swap – GitHub. Last accessed: 03/12/2019. [Online]. Available: https://github.com/AltCoinExchange/ethatomicswap