Toward Fairness of Cryptocurrency Payments

Jian Liu | Aalto University
Wenting Li and Ghassan O. Karame | NEC Laboratories
N. Asokan | Aalto University

Given that blockchain participants typically do not trust each other, enabling fairness in existing cryptocurrencies is an essential but insufficiently explored problem. We explore the solution space for enabling the fair exchange of a cryptocurrency payment for a receipt.

The phenomenal success of Bitcoin[1] has fueled innovation in a number of application domains such as financial payments, smart contracts, and identity management. There are currently more than 500 alternate cryptocurrencies—most of which are simple variants of Bitcoin. One of the (many) reasons that led to the growing adoption of blockchain-based cryptocurrencies is their promise of low-cost global payments without the need for a bank account or a cumbersome registration process. However, although existing cryptocurrency schemes can reasonably ensure the security of payments (for example, double-spending resistance and integrity of payment), they do not provide any guarantee of fairness. Given that blockchain participants do not necessarily trust each other, we argue that fairness is an especially important property that should be preserved to ensure the growth of existing cryptocurrency exchanges.[2]

For instance, consider the example where a payer Alice makes a payment to a payee Bob in return for an expected good (digital or physical) or a service. This process is unfair toward Alice if her expectation is not met after Bob receives the payment. On the other hand, it is unfair toward Bob if he does provide the service but Alice later cancels or double-spends the payment. A fair payment scheme should ensure that Bob receives the payment if and only if Alice's expectations are met and vice versa. We can model this as a fair exchange of payment-for-receipt where the receipt is a digital signature, which can act as a proxy for a physical/digital good or real-world service.

While there is a wealth of literature on fair exchange in a general setting,[3,4] little attention has been paid to the problem of fair exchange involving cryptocurrencies.[2] In this article, we explore the solution space to achieve fair payment-for-receipt for cryptocurrencies. More specifically, we analyze how well-known fair exchange techniques can be adapted for use with existing cryptocurrencies, in particular by leveraging functionality from a blockchain. We investigate three blockchain-based fair payment-for-receipt solutions, and systematically compare them based on both theoretical analyses and prototype implementations. We identify pros and cons of different schemes and discuss scenarios where a given solution is likely to be preferable. We hope that this work motivates researchers to further investigate this largely unexplored area of fair cryptocurrency payments.

Blockchain and Smart Contracts

The notion of blockchains was originally introduced by the well-known hash-based proof-of-work (PoW) mechanism that confirms cryptocurrency payments in Bitcoin.[1] Bitcoin payments are performed by issuing

1540-7993/18/$33.00 © 2018 IEEE Copublished by the IEEE Computer and Reliability Societies

transactions that transfer Bitcoin coins from the payer to the payee. These entities are referenced in each transaction by means of pseudonyms, denoted by Bitcoin addresses. Each address maps to a unique public/private key pair; these keys are used to transfer the ownership of coins among addresses. *Miners* are entities that participate in the generation of Bitcoin blocks. These blocks are generated by solving a hash-based PoW scheme; more specifically, miners must find a nonce value that, when hashed with additional fields (for example, the Merkle hash of all valid transactions or the hash of the previous block), the result is below a given target value. If such a nonce is found, miners then include it in a new block, thus allowing any entity to verify the PoW. Because each block links to the previously generated block, the Bitcoin blockchain grows upon the generation of a new block in the network.

As such, the PoW-based blockchain ensures that all transactions and their order of execution are available to all blockchain nodes and can be verified by these entities. Consensus by the majority of participating miners is required for every transaction exchanged in the system. This inherently prevents double-spending attacks (where the payer attempts to spend the same coin twice) and ensures the correctness of all transactions confirmed in the blockchain as long as the majority of the network is honest.

To ensure that a payment in a cryptocurrency transaction is definitive, a payee needs to wait until a sufficient number of new blocks have been appended to the block that contains the particular transaction so as to minimize the probability that the block is not part of the eventual consensus. In Bitcoin, this may take up to an hour. In some situations (for instance, low-value transactions), a payee may be willing to accept a transaction as soon as it is broadcast to the network. These are referred to as zero-confirmation transactions, which are fast but carry a risk of payment reversal.

> **A fair exchange protocol must ensure that a malicious player cannot gain any advantage over an honest player.**

Smart contracts refer to binding contracts between two or more parties that are enforced in a decentralized manner by the blockchain nodes without the need for a centralized enforcer. Smart contracts typically consist of self-contained code that is executed by all blockchain nodes. For example, Ethereum[5] is a decentralized platform that enables the execution of arbitrary applications (or contracts) on its blockchain. Owing to its support for a Turing-complete language, Ethereum offers an easy means for developers to deploy their distributed applications in the form of smart contracts. Ethereum additionally offers its own cryptocurrency Ether, which is also used as the main fuel to execute the contracts and send transactions. Ether payments are commonly used to cover the costs related to contract execution; these costs are measured by the amount of gas they consume.

Fair Exchange

A two-party exchange usually involves two players who exchange items between themselves. Each player holds an item that it wants to contribute to the exchange and an expectation about the other player's item it wants to receive in exchange. Fair exchange is executed between players that do not necessarily trust each other; examples include commercial scenarios such as payment-for-receipt, online purchase, digital contract signing, and certified mail. We say that a player is honest if it follows the protocol; otherwise, it is malicious (behaves arbitrarily). A fair exchange protocol must ensure that a malicious player cannot gain any advantage over an honest player. More specifically, it should satisfy the following requirements:[4]

- *Effectiveness*: If both players behave correctly and are willing to exchange, the protocol will eventually succeed.
- *Fairness*: There are two possible notions of fairness. In *strong fairness*, at the time of protocol termination, either both players get what they want (that is, the exchange succeeds) or neither of them does (that is, the exchange fails). For *weak fairness*, in situations where strong fairness cannot be achieved, an honest player can prove to an (external) arbiter that the other player has received (or can still receive) the item the latter expects.
- *Timeliness*: Regardless of the behavior of the other player, an honest player can be certain that the exchange will complete (either succeed or fail) in a certain time. At completion of the protocol, the state of the exchange is final from that player's perspective (for instance, the fairness achieved by the protocol will not be changed from strong to weak).
- *Noninvasiveness*: The protocol should allow the exchange of arbitrary items without making any demands on their structures. For example, a protocol is invasive if it requires anyone who wants to verify the exchanged signatures to access and perform some check on the blockchain.

The timeliness requirement defines a fixed point in time at which the protocol will be completed. This property aims to avoid the case where one player in the exchange has to wait indefinitely for the other player to take an action that will determine how the exchange will be concluded (succeed or fail). Timeliness is particularly important for resource-constrained IoT devices (for instance, a simplified payment verification [SPV] client), which cannot afford to be online for long stretches of time or poll indefinitely. One way to achieve the timeliness requirement, as stated in "Fairness in Electronic Commerce,"[4] is to agree on a predefined timeout. This is typically a challenging task, because it is difficult to predefine an optimal time point at which the protocol should be completed: a short timeout will result in the exchange failing even when both players are honest (thus harming the effectiveness requirement), whereas a long timeout is unacceptable for resource-constrained devices with limited battery or bandwidth. Ideally, the notion of timeliness should capture the possibility that either player can decide to conclude the exchange at any point during the exchange without having to depend on the actions of the other player. To remedy this, we therefore define a new notion of timeliness, dubbed *strong timeliness*, as follows: an honest player can, any point in time, choose to complete the protocol. At completion, the state of the exchange is final from that player's perspective.

In this article, we consider the "payment-for-receipt," where an entity, Alice, makes a digital payment to another entity, Bob, in order to get a receipt for the payment in the form of a digital signature. Our goal is to explore the solution space for integrating a fair exchange of payment for receipt into existing cryptocurrency payment schemes (hereafter referred to as *fair payments* for the sake of brevity). We assume the communication is weakly synchronous, under which messages are guaranteed to be delivered after a certain time bound.

Fair Payments via Timelocking

Bitcoin and other cryptocurrencies support *timelocking*, which enables a payer to reclaim its payment within a time window *tw* if the payment has not been spent yet. Many blockchain-based applications are built on top of this mechanism, for instance, online lottery,[6] fair multiparty computation,[7–9] and anonymous payments.[10,11] In this section, we discuss why the use of timelocking can negatively impact fairness guarantees and possibly even the overall security of the protocol. We take the anonymous payment protocol in "Blindly Signed Contracts: Anonymous On-Blockchain and Off-Blockchain Bitcoin Transactions"[10] as an example to explain that the fairness issue may lead to a serious attack. Similar attacks exist in all the above-mentioned applications.

Here, a user Alice wants to fairly exchange a cryptocurrency payment for a voucher from an intermediary Bob. The voucher is in fact a blind signature; Alice will unblind the voucher and send it privately to an anonymous payee who can exchange it with Bob in such a way that Bob cannot link Alice and the payee.[10]

This is achieved without relying on any external entity through the use of blockchain-based script and using a fixed, predefined timeout to implement a timely fair exchange. First, Alice generates a transaction that enables her to pay a predefined amount to Bob under the condition that Bob must publish a valid signature on a message within a certain time window. The output of this transaction will become an input in one of the following two blockchain transactions:

- a transaction that is signed by Bob and contains a valid signature on the requested message (that is, exchange is successful and fair); or
- a transaction that is signed by Alice and the time window has expired (that is, exchange fails and the money reverts to Alice).

The condition is fulfilled if Bob posts a transaction that contains a valid signature and the promised payment is transferred from Alice to Bob. If Bob does not publish a signature within the time window, Alice can sign and post a transaction that returns the promised payment amount back to herself. All transactions are broadcast to the blockchain network, thus allowing all blockchain miners to verify whether the payment conditions have been met, and reach consensus on the state of the exchange.

This protocol ensures a fair exchange between Alice and Bob; it prevents Alice from double-spending her payment and enables Bob to spend Alice's payment only if Bob has published his signature. We now analyze this protocol in relation to the requirements listed earlier:

- *Effectiveness*: If the timeout is too short, there may be not enough time for Bob's transaction to be confirmed in the blockchain. Namely, the miners will refuse to confirm that transaction after the timeout has passed. In this case, the effectiveness of the fair exchange cannot be guaranteed since the exchange fails because of the timeout even when both parties behave correctly.
- *Fairness*: The protocol does not ensure strong fairness since it is possible that the timeout is reached after Bob broadcasts his transaction, but before it is confirmed in the blockchain. For example, the adversary may mount a denial-of-service attack to throttle Bob's network connectivity.[12] In this case, Alice might receive the signature without Bob receiving the

payment. However, the protocol satisfies weak fairness, because Bob can prove to an (external) arbiter that his signature on Alice's requested message has indeed been revealed to the public. Note that the lack of strong fairness implies that the anonymous payment scheme of "Blindly Signed Contracts"[10] is insecure, since Alice can use the voucher to claim money from Bob without paying him before!

- *Timeliness*: This protocol satisfies weak timeliness but not strong timeliness, because once Bob's transaction is confirmed, Alice cannot decide to complete the exchange any sooner than the specific timeout.
- *Noninvasiveness*: The protocol is noninvasive because it does not impose any specific structure on Bob's signature.

Optimistic Fair Payments

Next, we introduce our first fair payment-for-receipt solution: optimistic fair payments.

Optimistic Fair Exchange

Optimistic fair exchange (OFE) protocols were first proposed by Asokan and colleagues;[3,4] their protocol relies on the presence of a trusted third party (TTP) but only in an optimistic fashion: TTP is required only when one player attempts to cheat or simply crashes. In the common case where Alice and Bob are honest and behave correctly, TTP need not be involved.

OFE consists of an exchange protocol (protocol `exchange`) and two recovery protocols (protocol `abort` and protocol `resolve`). First, both players agree on what needs to be exchanged and which third party to use in case of an exception. Such an "agreement" is purely a protocol construct: it has no validity outside the context of the protocol. Then, one player (for instance, Alice) sends a verifiable encryption (c_A) of her item (i_A) and her expectation about Bob's item (e_A). The verifiable encryption enables any entity to verify the validity of i_A (without the need for decrypting the message) and can be decrypted only by the TTP. Bob first verifies i_A, constructs an encryption c_B of (i_B, e_B), and decides similarly whether to send it to Alice.

If Bob does not send c_B, Alice can abort the protocol at any point in time by initiating protocol `abort` with TTP that issues an abort token. In this case, the exchange is unsuccessful but fair: neither player receives any additional information about the other's item. If Bob sends c_B and Alice has not decided to abort, she verifies the validity of i_B and decides whether to send i_A to Bob. While waiting for i_A, Bob can initiate protocol `resolve` at any time by sending (c_A, i_B) to TTP. TTP will decrypt c_A to get (i_A, e_A) and return i_A to Bob if i_B meets e_A and has not previously issued an abort token for this particular exchange. If a transaction was previously aborted, TTP will not agree to resolve it. Similarly it will not agree to abort a transaction that had already been resolved. Alice can run resolve in the same way while waiting for i_B from Bob. This is a general fair exchange protocol that can support "items" in the form of signatures in standard signature schemes. It requires TTP to keep state for every aborted or resolved transaction.

Blockchain-Based OFE with a Stateless TTP

We now extend the above OFE protocol by making use of a blockchain to avoid the need for TTP to maintain state. Alice can abort the exchange by publishing an `abort` transaction to the blockchain instead of sending an `abort` message to TTP. Thus TTP only needs to support the resolve protocol. It does not need to keep any state with regard to the protocol execution because all needed state information is recorded in the blockchain. We implemented this variant of OFE using Ethereum's smart contracts as shown in Figure 1. Note that when TTP recovers item i_A in response to a `resolve` request from Bob, it needs to save Bob's item i_B so that any subsequent `abort` from Alice can be answered correctly by the smart contract without violating Alice's fairness. Therefore, TTP will ask the smart contract to save i_B during Bob's invocation of `resolve`. Note that TTPs only use blockchain as an "external storage" to keep its state, so the security level of the new protocol is exactly the same as original. (TTP is required to be a blockchain [Ethereum] user, but it is not necessary to store the whole blockchain—for instance, it can be an SPV client).

We can easily build a fair payment protocol based on this blockchain-based OFE by having i_A be a signature corresponding to a payment message in a cryptocurrency scheme. First, we consider zero-confirmation payments: the two players exchange payment for a receipt but do not wait for the payment to be confirmed in the blockchain. We now analyze this protocol with regard to our defined properties:

- *Effectiveness*: Effectiveness is guaranteed if both players behave correctly, because Alice will get Bob's signature immediately after the OFE and her payment will be eventually confirmed.
- *Fairness*: A malicious Bob cannot gain any advantage from the protocol. He can get Alice's payment only by sending his signature either to Alice or to TTP. In both case, Alice can get his signature. However, a malicious Alice can double-spend the money associated with i_A after the completion of OFE, thus invalidating strong fairness. Bob can, however, prove this misbehavior to an arbiter by showing i_A. So, this scheme satisfies only the weak fairness property.

```
function abort(exchange id id_ex)
   if entry of id_ex exists then
      if sender is the originator and the retrieved entry is a resolved item i_B then
         return i_B to the sender
      end if
   else
      add an entry of id_ex with an abort token
   end if
end function

function resolve(exchange id id_ex, optional resolved item i_B)
   if sender is TTP then
      if entry of id_ex exists and the retrieved entry is an abort token then
         return aborted
      else
         add an entry of id_ex with the optional resolved item i_B
         return ¬ aborted
      end if
   end if
end function
```

Figure 1. Smart contract for blockchain-based optimistic fair exchange to assist `abort` and `resolve` procedures in order to keep trusted third party stateless.

- *Timeliness*: Strong timeliness is inherited from classical OFE: either player can invoke protocol `resolve` at any point if they have received the other player's verifiable encryption (c_A or c_B). Alice can attempt to abort at any time. In all cases, the protocol is guaranteed to terminate timely.
- *Noninvasiveness*: The signature i_B can be any signature in any form.

This variant can be upgraded from zero confirmation to full confirmation by borrowing the approach of Mayes and colleagues[2] to require that Bob and TTP check if i_A is confirmed on the blockchain as follows. After getting i_A from Alice, Bob broadcasts it and waits for it to be confirmed on the blockchain before sending i_B to Alice. When Bob asks TTP to resolve, TTP similarly broadcasts i_A and waits for it to be confirmed on the blockchain before storing i_B. When it resolves for Alice, it returns i_B only after i_A is confirmed on the blockchain. If i_A was double-spent before being confirmed, TTP will treat it as though Alice aborted the protocol. With this modification, a malicious Bob can still gain no advantage from the protocol. In addition, a malicious Alice gain no advantage either, since she can get Bob's signature only her payment has been confirmed on the blockchain. This full confirmation variant achieves strong fairness but at the expense of longer transaction duration.

Fair Payments of Blockchain-Based Signatures

We now describe a variant that dispenses with the need for TTP altogether but at the expense of making the signature invasive. Our proposal is as follows. Alice first constructs the message to be signed and uses it to create a transaction with an output of some amount of digital money that is spendable in one of the following two transactions:

- a transaction that is signed by Bob and contains a valid signature on the requested message; or
- an `abort` transaction that is signed by Alice.

Notice that there are no time constraints in Alice's transaction, and it can trigger either of the above two transactions, depending on which one is confirmed in the blockchain first. Recall that if both are broadcast, only one of them will eventually be confirmed (because they conflict with each other). This protocol is invasive since Bob's signature is valid only if it is stored on the blockchain, hence the term *blockchain-based signature*. Namely, a verifier must check not only that the signature is (cryptographically) valid but also that it is confirmed in the blockchain.

This protocol can be fully deployed as an Ethereum smart contract without the need for TTP. In this case, Alice will first send a deposit to the contract; the contract will forward the deposit either to Bob or back to Alice, depending on whether it receives Bob's signature or Alice's `abort` first. An example of such contract functions is sketched in Figure 2.

Our extension ensures the following properties:

- *Effectiveness*: If both players behave correctly, Bob will receive the payment by publishing a signature, and

```
function initExchange(payment T_pay[v], expected item m and recipient)
   if state is UNINITIALIZED and contract has received the payment with value v then
      record originator, recipient and m
      switch state to INITIALIZED
   end if
end function
function abort
   if state is INITIALIZED and the message is sent by the originator then
      refund v to the originator
      clear up storage and switch state to UNINITIALIZED
   end if
end function
function resolve(signature on m)
   if state is INITIALIZED and the message is sent by the recipient then
      if signature on m is valid then
         send v to recipient and the signature to originator
         clear up storage and switch state to UNINITIALIZED
      end if
   end if
end function
```

Figure 2. Smart contract for fair payment of blockchain-based signature.

Alice will obtain her desired receipt when the signature has been confirmed on the blockchain.

- *Fairness*: A malicious Alice can double-spend the payment, in which case, Bob's signature will not be added to the blockchain. Alice can still get the content of the signature, but it is invalid. A malicious Bob cannot gain any advantage from the protocol because he can get the payment only if his signature has been added to the blockchain.
- *Timeliness*: The protocol completes after either the signature or the `abort` is confirmed. Alice can choose to wait for the signature to be confirmed (exchange succeeds) or issue an `abort` (exchange fails). Similarly Bob can either issue a signature and wait for it to be confirmed, or simply walk away. In either case, the state of the exchange is final.
- *Noninvasiveness*: Clearly, the signature is invasive because it is only valid when it is confirmed in the blockchain.

Experimental Evaluation

We now describe and evaluate our Ethereum-based implementation of fair payment with blockchain-based OFE and with blockchain-based signature.

Implementation Setup

We assigned an Ethereum node to each entity (for example, Alice, Bob, and TTP). These nodes are connected to a private Ethereum network (that is equipped with private mining functionality) with a bandwidth limit of 100 Mbps. We deployed the mining node and TTP on two servers both with 24-core Intel Xeon E5-2640 and 32 GB of RAM. In our testbed, Alice and Bob reside on two machines equipped with 4-core Intel i5-6500 with 8 GB of RAM and 8-core Intel Xeon E3-1230 with 16 GB of RAM, respectively. In our implementation, these entities prepare and send the transactions to the blockchain using the JavaScript library web3.js. This library interfaces the Ethereum nodes through its RPC calls. In the blockchain-based OFE instantiation, we implement OFE computation and communication using GoLang and C. We use the ECDSA signature scheme, which is directly supported by Ethereum contracts. We use the verifiable encryption scheme in "Fairness in Electronic Commerce"[4] implemented with cryptographic library GMP[13] in C. We preset and fix the difficulty of our private Ethereum testnet in the code and the genesis block so that the block generation time is around 5 seconds.

In our experiments, we measured the gas and time consumption for the following procedures: `deploy`, `optimistic completion`, `abort`, and TTP `resolve`. `Deploy` refers to deploying the smart contract into the blockchain. In blockchain-based OFE, the smart contract is deployed by TTP to manage its state. `Optimistic completion` refers to the successful completion of the exchange without invoking `resolve` or `abort`. Finally, the contract is triggered by Alice for abort and by TTP for resolve. We consider only the resolve protocol under the assumption that the exchange has not been aborted.

To measure gas consumption, we observe the difference in the account balance before and after invoking the contract, and we convert this amount to the amount

of gas according to our fixed gas price. We measure the *eclipsed time* starting from the initial contract invocation until the entities receive the notifications from the Ethereum network. In our evaluation, each time measurement is averaged over 10 independent executions of the fair exchange protocol; where appropriate, we also report the corresponding 95 percent confidence intervals.

Evaluation Results

Our evaluation results are discussed as follows.

Gas consumption. Our evaluation results are shown in Table 1. We first observe that for both contracts, contract deployment consumes the most amount of gas because the processes of creating contracts and storing data in the blockchain are expensive in Ethereum.[14] As described earlier, the blockchain-based OFE variant does not need to involve the blockchain at all during optimistic exchanges. Therefore, the gas consumption for an optimistically concluded fair payment is zero. We contrast this with the blockchain-based signature variant, which requires 126,457 gas from Alice to initiate the exchange protocol and 27,935 gas from Bob to complete it. The large overhead incurred on Alice here is mainly caused by storing exchange contract parameters in the blockchain during contract initialization. Notice that `abort` requires considerably more gas in blockchain-based OFE when compared to the blockchain-based signature. This is due to the fact that the contract may spend more gas to transmit the previous resolved item (if any). Similarly, TTP `resolve` potentially needs to store resolved items in the contract—which incurs additional gas consumption.

Time consumption. Table 2 shows the measured eclipsed time. We observe that the contract invocation process is rather time consuming; for instance, the protocol initialization procedure by Alice in blockchain-based signature consumes around 4 seconds. We contrast this with 277 milliseconds required for the completion of the blockchain-based OFE protocol. The latter is almost 14 times faster in spite of the reliance on verifiable encryption, due to the fact that the blockchain needs to generate a block in order to include the transactions. Recall that the average block generation time in our private Ethereum network is tuned to be around 5 seconds.

The time execution of the remaining operations is comparable in both protocols, which is around 4 seconds. This value largely depends on block generation time of the blockchain network. Notice that the width of the confidence interval corresponds to the variation of block generation times exhibited in Ethereum.

Table 1. Gas consumption in Ethereum contracts to perform each action of blockchain-based fair payment protocols.

Actions		Protocols	
		With blockchain-based OFE	With blockchain-based signature
`Deploy`		537,783	645,900
`Optimistic completion`	Alice	0	126,457
	Bob		27,935
`Abort`		67,574	33,746
TTP `resolve`		132,600	–

Table 2. Eclipsed time in milliseconds with 95 percent confidence interval to perform each action of blockchain-based fair payment protocols.

Actions		Protocols	
		With blockchain-based OFE	With blockchain-based signature
`Optimistic completion`	Alice	277.0 ± 9.2	3,831.0 ± 973.2
	Bob		4,195.8 ± 1,077.3
`Abort`		3,432.4 ± 1,000.3	4,301.5 ± 1,305.6
TTP `resolve`		3,773 ± 902.5	–

Summary. Given our findings, we conclude that the fair payment protocol based on blockchain-based OFE is more cost- and time-effective than its counterpart based on blockchain-based signatures when the protocol is executed without exceptions.

In the case where an exception occurs, both protocols incur comparable costs and time overhead. Namely, because transactions can take effect only once they are confirmed in the blockchain (that is, every 12 seconds in the Ethereum public blockchain), the reliance on the blockchain in `abort` and `resolve` protocols incurs considerable time delays.

Comparison and Outlook

In this article, we explored the solution space to realize fair exchange for cryptocurrency transactions. To this end, we proposed two fair payment-for-receipt

Table 3. Comparisons of different blockchain-based fair payment protocols.

Requirements	Types			
	With timelocking	With blockchain-based OFE		With blockchain-based signature
		Zero-confirm	Full-confirm	
Fairness	weak	weak	strong	strong
Timeliness	weak	strong	strong	strong
Effectiveness	?	√	√	√
Noninvasiveness	√	√	√	X
No TTP	√	X	X	√

protocols for cryptocurrency payments that leverage functionality from the blockchain to meet both fairness and strong timeliness. A systematic comparison between our proposals is shown in Table 3.

Our findings suggest that the fair exchange based on timelocking cannot satisfy the strong timeliness property, and as such can only guarantee weak fairness. Furthermore, choosing a short timeout here can harm the effectiveness of this construct. In this respect, the constructs based on blockchain-based OFE and blockchain-based signature provide the strongest tradeoffs between performance and provisions. Our performance evaluation shows that the blockchain-based OFE option is more efficient when the exchange concludes optimistically. As such, it seems to be ideal in those scenarios where only weak fairness is sufficient or when noninvasiveness is required. Otherwise, we recommend using the blockchain-based signature option.

Formal verification techniques[15,16] can be used to complement our security analysis. We leave the investigation of such techniques for future work. Privacy has not been considered as a requirement for fair exchange protocols. However, there are a number of scenarios where privacy considerations play a paramount role. For example, the message to be signed may contain some important information about Alice that cannot be revealed. In all three constructions, the contents of the signature can be seen by the public; there are, however, a number of techniques that can be used to protect the contents of signatures. For instance, one can improve the blockchain-based OFE protocol by having TTP send a verifiable encryption of i_B to the resolve contract—thus preserving the privacy of Alice.

While there are no "bulletproof" solutions that simultaneously achieve all desirable properties discussed above, we observe that a number of tradeoffs exist within the solution space to sacrifice one property in order to achieve the rest. Depending on the application scope, this might already offer a differentiator and stronger value proposition for existing cryptocurrencies. We therefore hope that our findings motivate further research in this largely unexplored area. ■

Acknowledgments

This work was supported in part by NEC Laboratories Europe and a grant from Academy of Finland (309195, BCon).

References

1. S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System, 2009; www.bitcoin.org/bitcoin.pdf.
2. D. Jayasinghe, K. Markantonakis, and K. Mayes, "Optimistic Fair-Exchange with Anonymity for Bitcoin Users," *Proc. IEEE 11th International Conference on e-Business Engineering* (ICEBE 14), 2014, pp. 44–51; http://ieeexplore.ieee.org/document/6982058/?reload=true&arnumber=6982058.
3. N. Asokan, V. Shoup, and M. Waidner, "Optimistic Fair Exchange of Digital Signatures," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 4, 2000, pp. 593–610; http://dx.doi.org/10.1109/49.839935.
4. N. Asokan, "Fairness in Electronic Commerce," PhD thesis, University of Waterloo, 1998; https://uwspace.uwaterloo.ca/bitstream/handle/10012/292/NQ32811.pdf.
5. V. Buterin, *A Next-Generation Smart Contract and Decentralized Application Platform*, 2014; https://github.com/ethereum/wiki/wiki/White-Paper.
6. M. Andrychowicz et al., "Secure Multiparty Computations on Bitcoin," *Proceedings of the 2014 IEEE Symposium*

on Security and Privacy (SP 14), 2014, pp. 443–458; http://dx.doi.org/10.1109/SP.2014.35.
7. I. Bentov and R. Kumaresan, "How to Use Bitcoin to Design Fair Protocols," *Advances in Cryptology* (CRYPTO 14), 2014, pp. 421–439; http://dx.doi.org/10.1007/978-3-662-44381-1_24.
8. A. Kiayias, H.-S. Zhou, and V. Zikas, *Fair and Robust Multi-party Computation Using a Global Transaction Ledger*, Springer Berlin Heidelberg, 2016, pp. 705–734.
9. R. Kumaresan, V. Vaikuntanathan, and P. Nalini Vasudevan, "Improvements to Secure Computation with Penalties," *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (CCS 16), 2016, pp. 406–417; http://doi.acm.org/10.1145/2976749.2978421.
10. E. Heilman, F Baldimtsi, and S Goldberg, "Blindly Signed Contracts: Anonymous On-Blockchain and Off-Blockchain Bitcoin Transactions," *Financial Cryptography and Data Security* (FC 16), 2016, pp. 43–60; http://eprint.iacr.org/2016/056.
11. E. Heilman et al., "Tumblebit: An Untrusted Tumbler for Bitcoin-Compatible Anonymous Payments," IACR Cryptology ePrint Archive, 2016:575, 2016.
12. A. Gervais et al., "Tampering with the Delivery of Blocks and Transactions in Bitcoin," *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (CCS 15), 2015, pp. 692–705; http://doi.acm.org/10.1145/2810103.2813655.
13. "The GNU Multiple Precision Arithmetic Library," GMP, 2016; https://gmplib.org/.
14. G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," 2014; http://bitcoinaffiliatelist.com/wp-content/uploads/ethereum.pdf.
15. A. Mukhamedov, S. Kremer, and E. Ritter, "Analysis of a Multi-party Fair Exchange Protocol and Formal," *Proceedings of the 9th International Conference on Financial Cryptography and Data Security* (FC 05), 2005, pp. 255–269.
16. R. Chadha, S. Kremer, and A. Scedrov, "Formal Analysis of Multi-party Contract Signing," *Proceedings of the 17th IEEE Computer Security Foundations Workshop*, 2004, pp. 266–279.

Jian Liu is a Doctoral Candidate at Aalto University, Finland. He received his Masters of Science in University of Helsinki in 2014. He is instructed in applied cryptography and blockchains. Contact at jian.liu@aalto.fi.

Wenting Li is a Senior Software Developer at NEC Laboratories Europe. She received her Masters of Engineering in Communication System Security from Telecom ParisTech in September 2011. She is interested in security with a focus on distributed system and IoT devices. Contact at wenting.li@neclab.eu.

Ghassan O. Karame is a Manager and Chief researcher of Security Group of NEC Laboratories Europe. He received his Masters of Science from Carnegie Mellon University in December 2006, and his PhD from ETH Zurich, Switzerland, in 2011. Until 2012, he worked as a postdoctoral researcher in ETH Zurich. He is interested in all aspects of security and privacy with a focus on cloud security, SDN/network security and Bitcoin security. He is a member of the IEEE and of the ACM. More information on his research is at http://ghassankarame.com/. Contact at ghassan@karame.org.

N. Asokan is a Professor of Computer Science at Aalto University where he co-leads the secure systems research group and directs Helsinki-Aalto Center for Information Security - HAIC. More information on his research is at http://asokan.org/asokan/. Contact at asokan@acm.org.