A Multi-Party Contract Signing Solution Based on Blockchain

Josep-Lluis Ferrer-Gomila † and M. Francisca Hinarejos *†

Department of Mathematics and Computer Science, University of the Balearic Islands (UIB),
07122 Palma de Mallorca, Illes Balears, Spain; jlferrer@uib.es
* Correspondence: xisca.hinarejos@uib.es
† These authors contributed equally to this work.

Abstract: Fair digital signature of contracts and agreements is an essential process in any electronic commerce scenario, and therefore also in data marketplaces, where the relationships and agreements among the different parties (consumers and providers) are more dynamic. In multi-party contract signing, \( N \) parties wish to sign a contract in a such a way that either all signatories obtain evidence of the signing or none obtains conflicting evidence regarding the honest signatories; the exchange must be fair. In this paper, we propose a blockchain-based multi-party contract signing protocol. This solution does not require the existence or potential intervention of a trusted third party (TTP), thus avoiding the difficulty of \( N \) signatories agreeing upon a TTP. In addition, this proposal meets the necessary requirements: fairness, timeliness, non-repudiation of origin, and non-repudiation of receipt. Furthermore, confidentiality can be easily achieved. To minimize costs associated with the use of blockchain, it should be invoked in the case of exception (analogous to optimistic solutions with a TTP) and by only one of the \( N \) signatories. However, when the use of blockchain is required, we show that its cost is within perfectly manageable margins.

Keywords: fair exchange; contract signing; blockchain; fintech; trust-free system; e-commerce; trust; blockchain cost evaluation; digital economy; Ethereum

1. Introduction

The signing of contracts is an essential process in conventional commerce. When a contract is sufficiently important, a signature is obtained face to face to prevent one of the parties from cheating, for example, by obtaining a signed copy from the other party and not providing a signed copy; that is, the exchange must be fair. In the case of very important or special contracts, a trusted third party (TTP) intervenes in the signing of the contract (for example, a notary).

The digital signature of contracts and agreements is also a main process in both electronic commerce and marketplaces, and for this reason, protocols must be established (the steps that the parties must follow). In the electronic world, it is not possible (reasonable) for a signature to be obtained face to face. However, it is possible for a TTP to intervene in the protocols. In fact, most of the schemes that we find in the literature (e.g., [1–4]) are solutions with a TTP (since previous solutions without a TTP are not secure or feasible from a practical point of view [5]). Among the solutions with a TTP, we can differentiate two types: an online TTP, who intervenes in all executions, and an offline TTP, who intervenes only in the event of conflict. In both cases, these solutions are problematic because it can be difficult for the parties to identify a TTP whom all signatories agree is trustworthy.

In the marketplace scenario, where consumers and providers share the same business space, the relationships and agreements among the different parties are more dynamic [6]. Sometimes a contract must be signed by more than two parties, which is the case we discuss here of multi-party contract signing. We also find solutions with a TTP for this case in the literature [7,8], but some problems can still arise. The more signing parties there are, the more difficult it will be to agree on a TTP. In fact, after more than 40 years of proposals
with TTPs, we have not found any that have been put into practice and are available to potential signatories. One aspect that should not be forgotten is that TTPs, regardless of how "trusted" they may be, can act maliciously (harming one or more of the signatories). For this reason, it is very important that if entities other than the signatories are involved in the contract signing process, the behavior of these entities is verifiable. This weakness in classic solutions is one of the strengths of the blockchain: operations carried out through public blockchains are verifiable by the entire community of participants in the system.

In the literature, we find schemes for signing contracts that use blockchain [9]. In some of these solutions, it is possible to achieve the fairness property without using the services of a TTP. Most solutions are for the signing of contracts between two parties and therefore do not solve the problem of multi-party contracting. In fact, we found only one solution for signing contracts for three signatories [10], with the shortcoming that it is not easily extendable to \( N \) signatories. For this reason, in this article, we present a scheme for multi-party contract signing based on blockchain (without TTP) that improves on the solutions presented to date: it is truly multi-party and, in addition, requires the use of the blockchain at a low cost when problems appear during contract signing.

**Contributions.** The main contribution of this paper is that we present the first proposal for multi-party contract signing based on the blockchain (with Ethereum smart contracts). The previous proposals to sign contracts based on the blockchain, which we find in the analyzed bibliography, only allow the signing of contracts between two parties, except for one proposal that is designed for the signing of contracts between three parties. Therefore, it is the first proposal for \( N \) signatories (where \( N \) can be the desired value) that does not require the existence or the intervention of a TTP, as is the case in most of the classic proposals (with the inconveniences already indicated) for multi-party contract signing.

The second contribution is that the proposal is off-chain, that is, transactions are only carried out with the blockchain (the execution of smart contract functions) in the event that one or more of the parties attempts to cheat. If the signatories follow the steps indicated in the protocol, the intervention of the blockchain is not necessary, with the consequent cost savings. None of the solutions analyzed for two or three parties meets this requirement, and we are thus opening a new line to design protocols for blockchain-based contract signing in an off-chain manner.

Finally, we demonstrate that in the event that blockchain intervention is necessary, only one interaction with the blockchain is necessary, regardless of the number of signatories. Thus, the costs that the signatories should incur are very reasonable. Specifically, we conduct a temporal and cost assessment showing that a balance between the delay required to execute the blockchain functions and the price paid for their execution can be achieved.

**Organization.** We have organized this paper into the following sections. The security requirements for multi-party contract signing are defined in Section 2. Section 3 is devoted to analyzing the related work found in the literature. Section 4 provides a complete description of our multi-party contract signing protocol. This section is followed by a description of the code of the smart contract in Section 5 and the analysis of the security of our proposal in Section 6. In Section 7, we analyze the cost of our proposed solution. Finally, in Section 8, we summarize the main results.

### 2. Security Requirements

The security requirements for multi-party contract signing are the same as those for contract signing between two parties: fairness, timeliness, confidentiality, non-repudiation of origin, and non-repudiation of receipt. To these classic requirements, we add the requirement presented in [11]: bc-effectiveness. The other requirements are defined in [12]. Here, we present their definitions adapted to the multi-party case.

**Fairness.** For an honest signatory, the following must be fulfilled:

- If at the end of the protocol execution one or more of the other \( N - 1 \) signatories have evidence that the contract is signed, then the honest signatory must also have evidence that the contract is signed.
• If at the end of the protocol execution the honest signatory has evidence that the contract is signed, then none of the other \(N - 1\) signatories should have evidence that the contract is not signed.

Ultimately, at the end of the process, an honest signatory must have irrefutable evidence of the status of the contract signing with respect to the other \(N - 1\) signatories (be they honest or dishonest).

Timeliness. An honest signatory must know the final state of the contract signature (signed or not) in a finite, and knowable a priori, instant of time.

Failure to comply with this requirement has an impact on the fairness requirement. Suppose that an honest signatory must wait an indefinite time to know the status of the contract signing. During this time, it is possible that another signatory already has evidence that the contract is signed or not. This situation is not fair to the honest signatory.

Non-repudiation of origin and non-repudiation of receipt. Once all parties have signed the contract, it must be impossible for any of them to deny having submitted their signatures (non-repudiation of origin), and they should be equally unable to deny having received the signatures of the other parties (non-repudiation of receipt).

Be-effectiveness. The smart contract functions are executed only in exceptional situations and not in every protocol run. In other words, under normal conditions, only the \(N\) signatories intervene in the execution of the contract signing protocol. The execution of smart contract functions has an associated cost, and it is desirable to design protocols that minimize it.

Confidentiality. The signatories of the contract may want the content of the contract to be unknown to entities other than the \(N\) signatories.

3. Related Work

In this section, we review the articles found regarding contract signing based on the blockchain. For this, a search was carried out in Web of Science, Scopus, and Scholar (of the terms “contract signing”, “fairness”, and “blockchain”). Eight of thirteen references found have been analyzed, since five of them correspond to articles only available in Chinese (of which only the abstract is available in English).

None of the previous solutions analyzed solves the problem of multi-party contract signing, and none of them uses the off-chain mode. Therefore, it will not be possible to compare our proposal with the previous solutions presented in this section. However, the analysis of the previous solutions for blockchain-based contract signing between two parties, and in one case for three parties, is interesting because some lessons can be extracted to consider in the design of blockchain-based solutions (multi-party, two-party, or three-party).

The oldest reference is from [13], with a proposal that bases fairness on the involvement of a trustworthy time-stamping service in the protocol. The authors indicate that this role can be centralized or decentralized. In the second case, they note, tangentially, that a blockchain such as Bitcoin could be used. However, the authors do not specify what the protocol would be like in the event that the time-stamping service was provided by a blockchain. Consequently, they also do not provide any evaluation of the costs involved. In any case, each execution of the contract signing protocol would require the intervention of the blockchain, with its corresponding cost.

In Reference [10] presented a scheme for contract signing between three parties (\(A, B,\) and \(C\)) based on blockchain. This conference article is expanded by the same authors in a journal article ([14]). This latter article contains a security analysis, which the first one lacks. However, the proposal is essentially the same. The authors emphasized that their proposal allows for the privacy of the signed contract and that the number of rounds is constant.

In a first phase, the three signatories exchange directly (without the intervention of the blockchain) verifiable signatures on the contract they wish to sign. These are partial
signatures that do not constitute proof that the contract is signed. In a third phase, they must provide, through the blockchain, additional information that will make it possible to convert partial signatures into “final” signatures that will serve to prove that the contract is signed.

In a second phase, the signatories make deposits in the blockchain, to provide “guarantees” that they will comply with the provisions of the protocol (the third phase). These deposits, by design of the protocol, are asymmetric: $B$ and $C$ deposit twice as much cryptocurrency as $A$. It is true that in the end, the possible penalties are the same for all three signatories, but this asymmetry is not fair: $A$ has an advantage over $B$ and $C$ since he deposits a lower amount.

In a third phase, the signatories must publish on the blockchain the information necessary to convert partial signatures into final signatures before a deadline expires (each signatory has a specific deadline). The signatories who comply with this third phase will be able to recover the deposit they had made. Signatories who do not comply with this third phase will be penalized since they will not be able to recover their deposits, which will be distributed between the other two signatories.

The protocol has a weakness, which the authors themselves highlight. A situation may arise where $C$ obtains the evidence from $A$ and $B$, and he does not provide his evidence to $A$ and $B$. $C$ will be penalized: his deposit will be provided to $A$ and $B$. However, $C$ will have proof that the contract is signed, while $A$ and $B$ will not have such proof.

References [10,14] did not clarify the nature of the deposit (or the amount). Whatever it is, we want to emphasize that this makes this protocol (and others we will discuss later) not viable in all contract signing scenarios. In a purchase-sale contract, it is possible (and reasonable) to think that the buyer has funds (at least those necessary to make the purchase), but forcing the seller to have equivalent funds means that not all sellers can use this type of protocol.

References [10,14] did not provide a cost analysis, which is especially important because the number of interactions with the blockchain (up to 10) suggests that the proposal is not scalable: it is costly for three parties and most likely much more expensive for a possible extension to $N$ parties. In any case, the authors do not indicate how the protocol could be extended to $N$ parties, and from the analysis we have carried out, we conclude that it is not a trivial task.

Reference [15] presented a protocol for signing contracts in PDF format. Although the authors highlighted this fact of the format, it turns out to be quite irrelevant in the design of their protocol (contracts could also be signed in any other format, such as XML). What is highly relevant is that these authors, to overcome the problems posed by the restrictions of the Bitcoin and Ethereum blockchains, proposed to create a new cryptocurrency. This cryptocurrency is created from Bitcoin but expands the script opcodes to be able to perform necessary operations, according to the design of its protocol, on the blockchain. This means that the real viability of the proposal cannot be analyzed, nor can we conduct a cost analysis of the use of the blockchain. The authors especially emphasized that their proposal guarantees privacy regarding the content of the contract and regarding the identity of the signatories.

Tian et al.’s proposal shares with that of Huang et al. the defect that they base fairness on the deposit of cryptocurrencies that the signatories must make. In Tian et al.’s proposal, this amount is also recovered if the signatories follow the steps specified in the protocol. However, regarding the amount of the deposit, they went further than Huang et al., “… the deposit should be a similar value as the contract to be signed”. In addition to the disadvantage already indicated above, this value would not always have to be easy to calculate. Imagine the case in which the buyer and the seller must convert different currencies to cryptocurrencies. Unfortunately, these cryptocurrencies fluctuate continuously (with respect to different foreign exchanges).

It also shares with the Huang et al. proposal the defect that it cannot be considered to meet the fairness requirement. As the authors indicated, a situation may arise where “…
party A gets a signed contract of party B at the cost of losing the deposit”. It is true that B receives compensation, but he does not know whether A will execute the contract. Suppose that the contract is for the purchase of X units of a product. If B does not manufacture these units, and A decides to execute the contract, then B must face a possible lawsuit from A. If B manufactures the units and A decides not to execute the contract, then B will have lost an opportunity cost (he could have accepted other orders that would have brought direct benefits—income—and indirect benefits—the possible diffusion of customer satisfaction).

Execution of the Tian et al. protocol involves a minimum of three and a maximum of five transactions with the blockchain. In this case this is relatively important because, as we have indicated, they use a new blockchain, about which we do not know the possible transaction costs, who manages the cryptocurrency, etc.

In Reference [12], we presented a protocol for contract signing between two parties based on the Bitcoin blockchain (without using scripts or smart contracts). The protocol consists of four steps, and it involves only one transaction with the blockchain. We presented an evaluation of costs, showing that they are reasonable values for a signature between two parties, especially if the volume of signed contracts is low. The proposal is accompanied by a proof of concept that allows us to demonstrate its practical viability. From the perspective of multi-party contract signing, our previous proposal suffers from two defects. On the one hand, each execution implies a transaction with the blockchain and, therefore, associated costs. On the other hand, the proposal targets contracting between two parties, and it is not envisaged that the extension to N parties will be a simple task.

Reference [16] presented a protocol that, although the terms “contract signing” appear in the title, is actually a protocol for the purchase of physical goods in exchange for payment. It is a problem in the same family as contract signing (fair exchange of values), but it has some peculiarities that distinguish it from the objective of contract signing protocols and, therefore, the objective of this article. In any case, it also has the drawback that the achievement of fairness is based on the parties making deposits on the Bitcoin blockchain. Furthermore, the proposal is explained at a very high level, without specifying the details of the protocol, which makes it difficult to carry out an analysis of the protocol. Among the data that are provided, we find different deadlines: one of three days and one of seven days. These are very long periods, which may be reasonable for the purchase-by-payment application but which we do not find in any contract signing proposal.

Reference [17] presented two proposals for contract signing based on blockchain. The first one is simple (three steps), but the content of the contract is published on the blockchain, so its privacy is not possible. Moreover, each of the three steps of the protocol involves a transaction with the blockchain.

The second solution from Mut-Puigserver et al. achieves the privacy of the contract content and is therefore more complex. In this case, an encrypted version of the contract is published on the blockchain. This, which has privacy as a positive aspect, has a drawback: there are very voluminous contracts, and the cost of storing them on the blockchain, encrypted or not, can entail a high cost. The authors also incorporated the deposit mechanism (in both proposals), but note that it is optional (it is only an incentive, not an obligation, for the initiator of the contract to act honestly).

Another drawback of Mut-Puigserver et al.’s “confidential” proposal is that it requires the execution of cryptographic operations on the blockchain, which is costly. The authors indicated that the evaluation of the protocol is pending, particularly the evaluation of the costs of executing the smart contract functions. It is relevant that this proposal requires the execution of five to seven functions on the blockchain. Moreover, on several occasions, the authors affirmed that the design of the protocol has been carried out with the intention that it be extensible for more than two signatories (multi-party). However, it does not seem that it will be a trivial task or have an efficient result, and the authors themselves left as future work to “reformulate the system of confidential contract signing to create a multiparty protocol”.

The last scheme analyzed is that of [18]. This scheme uses a verifiable encryption signature to achieve the privacy of the contract. Their proposal also uses the deposit mechanism but with a great difference from the previous proposals. If both parties are honest, they will get their deposits back (as in all proposals analyzed). If one of the parties is dishonest and the other honest, then the latter will receive the deposit of the former (and will recover its own), but in addition, the blockchain will mark the contract as invalid. Therefore, deposits, which continue to present the aforementioned drawbacks regarding determining the amount and requiring that both parties have sufficient cryptocurrency, in this case are an incentive to act honestly but do not have a negative impact on fairness. As a negative point, the process of invalidating a contract requires the applicant to provide the text of the contract, with the consequent loss of privacy.

Zhang et al.'s proposal also makes intensive use of the blockchain: up to six transactions are carried out with the blockchain in the event that both parties are honest. Fortunately, the authors provide the costs in terms of gas (they work with the Ethereum blockchain) of the six functions established in their protocol.

Although none of the solutions analyzed solves the problem of multi-party contract signing, some interesting conclusions can be drawn, which we have considered when designing our solution. Basing the fairness of the proposal on deposits can limit the scope of the proposal and even lead to proposals that do not meet the fairness requirement. A high number of transactions with the blockchain simplifies the design of the proposal but at the cost of increasing the costs for the signatories of the contract. In relation to costs, the proposals should include a cost analysis, since a system can be secure and practical but not viable from an economic point of view. Worse is the use of non-standard blockchains or cryptocurrencies: this does not allow conclusions to be drawn about the viability of the same, neither from a practical point nor from a cost point of view. Finally, having a solution for contract signing between two or three parties does not mean that we can straightforwardly extend the solution to \( N \) signatories.

4. Proposed Multi-Party Contract Signing Protocol

4.1. Overview of the Proposal

Our protocol proposal includes three sub-protocols (see Figure 1): exchange, finish, and query. Under normal conditions, only the exchange sub-protocol is executed. If any of the signatories observes that the rules established in the exchange sub-protocol are not being followed, they can use the blockchain to execute the other sub-protocols to solve the problem.

Figure 1. An overview of the multi-party contract signing proposal.

The exchange sub-protocol is divided into two rounds (see Figure 2). In the first round (Signatures exchange), the parties exchange signatures on the text of the contract. When the
round ends, all parties must have the contract with the signatures of all other signatories. However, these signatures, by themselves, do not constitute proof that the contract is signed. For this, a second round (ACK exchange) is necessary, in which the parties exchange the acknowledgment that each has the signatures on the contract of all other signatories. When the second round ends, each signatory must have acknowledgments of receipt from all other signatories.

A signatory may receive the signatures of all other signatories but not receive acknowledgment of receipt from all other signatories. If he does not take any action, then he will have no proof that the contract is signed. To avoid this, this signatory can use the blockchain, executing the \textit{finish} function of the smart contract to record that the contract has been signed. The execution of the \textit{finish} function must be requested before an agreed-upon deadline expires. This deadline must be agreed by all signatories and must be selected appropriately according to the temporal requirements of the scenario. In Section 7, we analyze the implications of the deadline value on the cost of the proposal.

If a signatory (A) has signed and sent the contract but does not receive the signatures of all other signatories, then she cannot execute the \textit{finish} function (see the previous paragraph). However, it could be that another signatory (B) had received all signatures and executed the \textit{finish} function. Therefore, signatory A would have no way of knowing the final status of the exchange. To avoid this situation, we have designed a \textit{query} function.

Once the deadline has expired, signatory A can check the final status of the contract and either obtain the necessary evidence if the contract is signed or know that the contract is not signed.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure2.png}
\caption{Exchange sub-protocol: linear transmission followed by the signatories to exchange the signatures and their corresponding acknowledgments.}
\end{figure}
4.2. Specification of the Proposal

In this subsection, we detail the steps followed by the three sub-protocols described above, along with the content of each exchange message. The notations used to describe the protocol are shown in Table 1.

Table 1. Multi-party contract signing protocol parameters.

| Symbol | Description |
|--------|-------------|
| $B_i$  | Signatory $i$ of the contract |
| $\mathcal{B}$ | The set of signatories $B_1, \ldots, B_N$ |
| $SC$   | Smart contract |
| $@X$   | Blockchain address of entity $X$ |
| $M$    | Contract content |
| $\text{Sig}_i$ | Signature on the contract by $B_i$ |
| $\text{ACK}_i$ | Acknowledgement by $B_i$ |
| $t$    | Finish deadline |
| $h()$  | One-way collision-resistant hash function |
| $id_E$ | Identifier of the contract |
| $\text{Sig}_Y(x)$ | Signature on the element $x$ made by party $Y$ |

4.2.1. Exchange Sub-Protocol

As already explained, the exchange sub-protocol requires two rounds to be completed. If it is completed successfully, the use of the blockchain is not required.

Figure 2 shows the linear transmission followed by the signatories to exchange their signatures and their corresponding acknowledgment of receipt. Figure 3 specifies the content of the messages involved in the two rounds of the exchange sub-protocol.

**Round I (signatures exchange).** In the first round, each signatory must sign on a hash of the following information:

- the hash of the contract text,
- the ordered list of the public keys of the signatories,
- the deadline (time in which the process must be finished),
- an identifier $id_E$ of the exchange (a hash value on the data associated to the contract), and
- the blockchain address of the smart contract.

To exchange signatures, a linear transmission will be followed back and forth. The first signatory will send her signature to the second signatory. Then, the second signatory will send his signature and that of the first signatory to the third signatory, and so on, until the last signatory is reached (Round I-1). At this point, the last signatory will have the signatures of all other signatories, and he must send his signature to the penultimate signatory. This signatory must send his signature and that of the last signatory to the previous one, and so on, until the first signatory is reached (Round I-2). All signatories now have the signatures of all other signatories.
Round I-1. $B_i \rightarrow B_{i+1}$: signatures on the contract to the last signatory

$m_{i,1} = \{ C, \text{Sig}_i, \text{Sig}_{i-1}, ..., \text{Sig}_1 \}$

where: $C = \{ h(M), B, t, id_E, @SC \}$

$id_E = h(h(M), B, t, @SC)$

$\text{Sig}_i = \text{Sig}_i(C)$

Round I-2. $B_i \rightarrow B_{i-1}$: signatures on the contract to the first signatory

$m_{i,2} = \{ \text{Sig}_i, \text{Sig}_{i+1}, ..., \text{Sig}_N \}$

where: $\text{Sig}_i = \text{Sig}_i(C)$

Round II-1. $B_i \rightarrow B_{i+1}$: ACKs to the last signatory

$m_3 = \{ \text{ACK}_i, \text{ACK}_{i-1}, ..., \text{ACK}_1 \}$

$\text{ACK}_i = \text{Sig}_i(\text{"ACK"}, h(C))$

Round II-2. $B_i \rightarrow B_{i-1}$: ACKs to the first signatory

$m_4 = \{ \text{ACK}_i, \text{ACK}_{i+1}, ..., \text{ACK}_N \}$

where: $\text{ACK}_i = \text{Sig}_i(\text{"ACK"}, h(C))$

Figure 3. Message specification of the two-round exchange sub-protocol.

**Round II (ACK exchange).** In the second round, the signatories must exchange an acknowledgment of receipt demonstrating that they all received the signatures of all other signatories. A signatory’s acknowledgment of receipt consists of signing a hash of the following information:

- the text ‘ACK’ and
- a hash of all information indicated in the first round.

To exchange acknowledgments, the same procedure is followed as indicated for the first round. Acknowledgments of receipt will circulate from the first signatory to the last signatory (Round II-1), at which time the last signatory has proof that the contract is signed. Acknowledgments will now circulate in reverse (Round II-2). When the process ends, all signatories will have proof that the contract is signed.

4.3. Finish Sub-Protocol

If a signatory has the signatures of all other signatories (the first round has been completed for him) but does not have the acknowledgment of receipt from one or more of them, then he must invoke the **finish** function of the SC. The parameters of the **finish** function (see Figure 4) are the contract identifier, $id_E$, the deadline, $t$, and two arrays with the following content:

1. each signatory’s public key $\{ B_i \in B \}$ and
2. the signatures of all signatories $\{ \text{Sig}_{B_i} : B_i \in B \}$.

The **finish** function must verify whether the deadline $t$ has expired. If $t$ has expired, the **finish** function ends. If $t$ has not expired and all validations are correct (signatures and the contract identifier provided), the SC updates the status of the exchange to **finished** and stores the signatures provided as parameters. This information will be available to the other signatories using the **query** function.
The deadline value must be selected according to the scenario in which the solution is deployed. In a scenario where the signatories sign the contract quickly, or the signature mechanism is highly automated, the deadline will be determined mainly by the cost that the use of the blockchain may require. If time requirements for the contract signature are long, the cost for the possible use of the blockchain will be reduced. If the time requirements for the signature are very short, minutes or less than an hour, the cost increases. In Section 7, we discuss the relationship between the economic cost and the time delay of the blockchain to select an appropriate deadline for each scenario.

4.4. Query Sub-Protocol

If a signatory (Bob) has sent his signature and does not receive the signatures of all other signatories, then he cannot request the execution of the finish function. However, it is possible that another signatory has executed the finish function. Therefore, Bob is in a situation where he does not know the status of the exchange. To solve this situation, Bob must call the query function of the smart contract.

This query function (see Figure 5) has an argument: the exchange identifier for this contract. If the query is called before the deadline has expired, the function response is an indication of the situation. That is, if no signatory has yet executed the finish function, the status can still change before the deadline (the smart contract has no information regarding this contract). However, if a signatory has already successfully executed the finish function, the status will be finished, and the smart contract will return the signatures of all signatories.

If the deadline has expired, the response of the query function indicates the final status of the exchange. In this case, the function can provide one of two possible answers:

1. there is no information regarding this contract (in this case, Bob will know that the contract is not signed), or
2. the contract is finished (because someone executed the finish function), the query will result in the signatures of all signatories being obtained, and Bob will be able to verify that the status of the contract is finished.
5. Smart Contract Specification

We deployed the smart contract (SC) defined in Section 4.2 on Ethereum “because Ethereum was designed as a platform for deploying smart SC that run on the blockchain to facilitate, execute and enforce the terms of an agreement among parties” [19].

Moreover, it is the second largest cryptocurrency by market share according to CoinMarketCap (CoinMarketCap: Cryptocurrency Market Capitalizations. Source: https://coinmarketcap.com/. Data publication date: 1 March 2021). Note that a smart contract defined in Ethereum should not be confused with a contract that outlines the terms of a relationship (usually one enforceable by law).

As explained in Section 4.1, when a signatory has the signatures of all signatories of the contract but does not have the acknowledgment of receipt from one or more of them, he can call the SC functions.

Therefore, the signatories must agree on the SC to be used. This agreement is conducted in the negotiation phase prior to the signing of the contract itself, in the same way that the signatories agree on any other information necessary to carry out the signing of the contract, such as the finish deadline. The address of the contract (@SC) must be included in the first round of the exchange sub-protocol (see Section 4.2.1), so the SC must be deployed before the exchange sub-protocol can be executed.

Once the contract has been deployed (including the code of the smart contract in a transaction), it is self-executed in the EVM (Ethereum Virtual Machine) of each node of the Ethereum network. To execute its functions, a signatory must send a transaction (to the address of the deployed contract, @SC) indicating the function to be executed and the required parameters. Each network node executes the function and, if the execution is successful, the transaction is published on the blockchain, recording its execution.

All analyzed protocols for contract signing based on smart contracts (Section 3), with the exception of [12], require a cost when deploying the smart contract. However, in addition, the execution of these protocols entails an additional cost each time a contract is signed because they require executing functions of the deployed smart contract. In contrast, our proposal involves only the deployment costs, but it does not involve additional subsequent costs for each contract signing (except in cases of conflict, which should be exceptional, since a dishonest signatory will not obtain any benefit by not following the steps indicated in the protocol). Therefore, the SC may be deployed only once (by any of the signatories or other entities) and used as many times as required to sign different multi-party contracts. For this purpose, it is not necessary to provide any information regarding a specific execution of a multi-party contract signing when deploying the SC. This exhibits the importance and advantage of meeting the requirement of bc-effectiveness.

We define a data structure (see Listing 1) to store the data associated with each contract (the hash of the contract, finish deadline, signatures, and status of the contract) and indexed by $\text{id}_E$. All these data are provided by the signatory who calls the $\text{finish}$ function and are verified by the SC before establishing its status (as explained below). A new data structure is created for each multi-party contract signing that requires the SC to be called. Note that no confidential data are stored in the SC data structure; that is, the content of the contract, $M$, cannot be recovered by anyone from the signatures or the stored hash of the contract.

Listing 1: Data structure associated with each exchange identifier ($\text{id}_E$) requiring the use of the blockchain.

```
struct Contract {
    bytes32 hashContract;
    uint deadline;
    bytes[] signatures;
    bool finished;
}

mapping(bytes32 => Contract) exchanges;
```

The SC provides two public functions:
• **finish**: given a contract identifier, any signatory can provide all signatures on the contract, before it expires, establishing the status of the contract as **finished**.

• **query**: given a contract identifier, any signatory may check the status of the contract, obtaining all signatures if the contract was previously finished by one of the signatories.

A signatory provides, as input parameters to the `finish` function (see Listing 2), the contract identifier, the associated deadline, the hash of the contract, the list with the public key of each signatory, an index pointing to the signatory’s public key that calls the `finish` function, and all signatures on the contract.

### Listing 2: Code of the `finish` function.

```solidity
function finish(bytes32 _idE, uint _deadline, uint _index,
    bytes32 _hashContract, address[] memory _signatories,
    bytes[] memory _signatures) public returns (string memory) {
    require(!exchanges[_idE].finished, "Contract already finished");
    require(now <= _deadline, "Contract not finished - Time exceeded");
    require(_signatories[_index] == msg.sender, "Contract not finished - Invalid signatory");
    bytes32 idE = keccak256(abi.encodePacked(_hashContract,
        _signatories, _deadline, address(this)));
    require(idE == _idE, "Contract not finished - Wrong idE");
    bytes32 htosign = keccak256(abi.encodePacked(idE, _hashContract,
        _signatories, _deadline, address(this)));
    for (uint i = 0; i < _signatories.length; i++) {
        address signatory = _signatories[i];
        bytes memory signature = _signatures[i];
        require(isValidData(htosign, signatory, signature),
            "Contract not finished - Invalid signature");
        exchanges[_idE].signatures.push(signature);
    }
    exchanges[_idE].hashContract = _hashContract;
    exchanges[_idE].deadline = _deadline;
    exchanges[_idE].finished = true;
    return("Contract finished");
}
```

The SC checks that the contract has not already been finished (there is no entry for that identifier in the SC data structure), that the signatory calling this function is one of the legitimate signatories of the contract, and that the deadline has not expired. Next, the SC must verify that the data provided by the signatory requesting finalization are linked to a valid contract identifier (idE). As shown in Figure 3, the identifier is generated as the hash of the h(M), B, the expiration time and @SC. To do so, the SC retrieves its Ethereum address (@SC) and the public key of the requesting signatory from its own information to generate the idE (the contract is assigned the address of the signatory who called the `finish` function). Next, the SC checks that the identifier matches the one provided by the signatory as input to the `finish` function. In this way, only one of the legitimate contract signatories can successfully execute the `finish` function. After the identifier is verified, the SC checks the validity of each signature provided by the signatory (see Listing 3).

The use of secure signatures is a key point in any contract signing procedure. Thus, it is crucial to consider secure algorithms using cryptographic keys of suitable length. Moreover, in any electronic contract signing, designers and developers should consider legal regulations, but it is outside the scope of this paper. We use the ECDSA (Elliptic Curve Digital Signature Algorithm) signature scheme [20], the same used by Ethereum for signing transactions. The security level (“a number associated with the amount of work (the number of operations) that is required to break a cryptographic algorithm or system” [21]) recommended today [22,23] is at least 112 bits, equivalent to a 224-bit key for ECDSA. Ethereum uses keys of 256 bits, equivalent to a security level of 128 bits. For these conditions, the use of an SHA-256 hash function is sufficient to maintain the security level of the signature scheme [23]).
Listing 3: Verifying the contract signature for each signatory.

```solidity
function isValidData(bytes32 _htosign, address _signer, bytes memory _signature) private pure returns (bool) {
    return (recoverSigner(_htosign, _signature) == _signer);
}

function recoverSigner(bytes32 resumen, bytes memory sig) private pure returns (address) {
    uint8 v;
    bytes32 r;
    bytes32 s;
    (v, r, s) = splitSignature(sig);
    return ecrecover(resumen, v, r, s);
}
```

The digital signatures on the contract are created off-chain during the exchange sub-protocol but must be validated by the SC (on-chain) when calling the finish function is required. An ECDSA signature together with the hash and the signature on the contract allow the public key used to sign that contract to be recovered (see Listing 3). We use the extended ECDSA signature to allow the public key to be recovered from its signature with no ambiguity. In this way, one can verify whether the signature on the contract was originally produced by each signatory (each public key provided to the finish function).

Once all signatures have been validated, the finish function modifies the status of the exchange for the given exchange identifier, so a new transaction is included in the blockchain, as will be explained in Section 7. Each transaction signature is used to authorize transactions on behalf of the signatory. Therefore, these signed transactions can be used as non-repudiation evidence.

Any signatory can call the query function (see Listing 4) providing the contract identifier. The smart contract checks whether a data structure for this identifier exists (Listing 1) and whether the status of the contract was finished. If the contract was finished, the SC returns the signatures of all signatories. The query function does not modify the status of the exchange and therefore does not require any transaction to be included in the blockchain (see Section 7).

Listing 4: Code of the query function.

```solidity
function query(bytes32 _idE) public view returns (string memory, bytes[] memory) {
    if (!exchanges[_idE].finished) {
        return ("Contract doesn’t exist", exchanges[_idE].signatures);
    }
    return ("Contract finished", exchanges[_idE].signatures);
}
```

6. Security Analysis

In this section, we demonstrate that our proposal meets the requirements explained in Section 2. The bc-effectiveness is the first requirement that we will verify.

If all signatories follow the two rounds of the exchange sub-protocol, at the end of the execution, the signatories will have evidence that the contract is signed, and the query or finish functions of the smart contract will not have been executed. Therefore, our protocol satisfies the bc-effectiveness requirement.

Next, we will prove that for any honest signatory, whatever the behavior of the other \( N - 1 \) signatories (whether they are honest or attempt to cheat), the protocol meets the fairness requirement: at the end of the exchange, he is not in a disadvantageous situation. If the protocol is fair to any honest signatory, regardless of what others signatories do, this
means that the protocol is fair to all honest signatories. Before proceeding with the proof, we present some preliminary questions:

- We will say that a signatory is honest if he follows the steps established in the protocol.
- We do not analyze whether the protocol is fair or unfair for a dishonest signatory since failing to follow the steps established in the protocol is his responsibility.
- In relation to the blockchain, for the signing of a specific contract, two situations may arise:
  - the finish function has not been executed successfully; in this case, no information regarding the contract has been published on the blockchain;
  - the finish function has been successfully executed; this means that:
    * a signatory executed the finish function before the deadline expired (after the deadline, the finish function will not be executed);
    * the signatory provided the correct signatures of all the $N$ signatories of the contract (if one or more of the signatures are not correct, the finish function will not be executed successfully);
    * it will be recorded in the blockchain that this contract has been signed for the $N$ signatories (if a signatory executes the query function, he will obtain evidence that the contract is signed).

Now, we will proceed to analyze the possible situations that the honest signatory (Bob) may face:

**situation-1**: Bob has not received the signatures to be sent by the previous signatory (note that the first signatory cannot be in this situation since he does not have a previous signatory). Since Bob is honest (he follows the steps of the protocol), he does not take any action (he does not send anything to anyone). Therefore, we have the following:

- Since Bob has not sent anything, no other signatory can have all the signatures and all the acknowledgments.
- No signatory can successfully execute the finish function because this requires all $N$ signatures, and Bob has not provided his signature to anyone.

Therefore, in this situation the contract is not signed, and correspondingly Bob has no evidence of the signing of the contract, as none of the other signatories does: it is fair.

**situation-2**: Bob has sent his signature and has not received the signatures from the following signatory in the chain (note that the last signatory cannot be in this situation since he does not have a following signatory). Since Bob is honest (he follows the protocol steps), he will not send his acknowledgment to anyone. Therefore, no other signatory can have all the signatures and all the acknowledgments. However, Bob knows that some signatory could have all the signatures, and therefore he must request the execution of the query function when the deadline expires. Bob can face two situations:

- **situation-2.1**: No one has successfully executed the finish function. Therefore, in this situation the contract is not signed, and correspondingly Bob executes the query function and obtains no evidence of the signing of the contract, as none of the other signatories does: it is fair.
- **situation-2.2**: A signatory has successfully executed the finish function. Therefore, in this situation the contract is signed. Bob executes the query function when the deadline expires and obtains the evidence that the contract is signed. The signatory who successfully ran the finish function had evidence of the contract signing, and now Bob has it too: it is fair.

Therefore, whether situation-2.1 or situation-2.2 occurs, the exchange is fair for Bob.

**situation-3**: Bob has received the signatures of all the other signatories, and he is not receiving the acknowledgments from the previous signatory in the chain (note that, again, the first signatory cannot be in this situation since he does not have a previous signatory).
Bob will not send his acknowledgment, and therefore no one can have evidence of the signing of the contract without the intervention of the blockchain: as Bob has not sent his acknowledgment, no other signatory can have all the signatures and all the acknowledgments. Since Bob is honest (he follows the protocol steps), he will request the execution of the finish function. Now, Bob can face two situations:

- **situation-3.1**: No one has successfully executed the finish function before Bob. Bob has all the signatures, and with this information, he can request the successful execution of the finish function, with which he will obtain the evidence that the contract is signed. There is no evidence in our protocol that could contradict Bob: it is fair.

- **situation-3.2**: Another signatory has successfully executed the finish function. Therefore, in this situation the contract is signed. When Bob requests the execution of the finish function, he will see that it has already been executed (and cannot be executed again), but Bob will still obtain the evidence that the contract is signed. The signatory who successfully ran the finish function had evidence of the contract signing, and now Bob has it too: it is fair.

Therefore, whether **situation-3.1** or **situation-3.2** occurs, the exchange is fair for Bob.

**situation-4**: Bob has received the signatures of all the other signatories, he has received the acknowledgments of the previous signatories, he has sent his acknowledgment of receipt (he follows the steps in the protocol), but he has not received all the acknowledgments of receipt from the following signatory in the chain (note that, again, the last signatory cannot be in this situation since he does not have a following signatory). Once Bob has sent his acknowledgment, one or more of the other signatories can obtain all the signatures and all the acknowledgments. Therefore, Bob, following the protocol, must attempt to execute the finish function before the deadline expires. Analogous to what we explained in **situation-3**, Bob can face two situations:

- **situation-4.1**: No one has successfully executed the finish function before Bob. Bob has all the signatures, and with this information, he can request the successful execution of the finish function, with which he will obtain the evidence that the contract is signed. There is no evidence in our protocol that could contradict Bob: it is fair.

- **situation-4.2**: Another signatory has successfully executed the finish function. Therefore, in this situation the contract is signed. When Bob requests the execution of the finish function, he will see that it has already been executed (and cannot be executed again), but Bob will still obtain the evidence that the contract is signed. The signatory who successfully ran the finish function had evidence of the contract signing, and now Bob has it too: it is fair.

Therefore, whether **situation-4.1** or **situation-4.2** occurs, the exchange is fair for Bob.

**situation-5**: Bob has received the signatures and acknowledgments from all the other signatories. Therefore, Bob has the evidence that the contract is signed. There is no evidence in our protocol that could contradict Bob: it is fair.

In conclusion, any honest signatory is guaranteed that if he follows the steps of the protocol, the fairness requirement is met for him: he will never be at a disadvantage. This statement is valid for each and every one of the signatories who are honest, with which we conclude that the protocol meets the fairness requirement: if any signatory has evidence that the contract is signed, all honest signatories will have evidence that the contract is signed; if the honest signatories have no evidence that the contract is signed, no signatory will have evidence that the contract is signed.

If all parties are honest, the execution of the protocol ends after the second round of the exchange sub-protocol. However, in any of the cases, all parties have the guarantee that at $t$ (the deadline), the exchange will have reached a final state. Before this moment, any signatory not receiving an acknowledgment must have executed the finish function. Signatories not receiving signatures can execute the query function at any time after $t$ to
know the final state of the exchange (and obtain evidence). In short, the protocol satisfies the timeliness requirement because it allows the parties to know that the execution of the protocol ends fairly in a finite time.

In the first round of the exchange sub-protocol, the text of the contract is transmitted without encryption. This is performed to avoid introducing further complexity into the explanation of the protocol, but it is very easy to obtain a confidential exchange. The text of the contract can be encrypted with a symmetric cryptographic key $k$, and this key can be encrypted with the public keys of all signatories, so that only they can access the content of the contract. In the event that a smart contract function must be executed, this function needs only the hash of the contract to conduct the signature verification (if applicable). Therefore, we can affirm that the proposed protocol can easily satisfy the confidentiality requirement.

Once the exchange is positively completed (with or without the execution of the smart contract functions), all parties have evidence of non-repudiation. Each signatory (e.g., Bob) has evidence of the non-repudiation of receipt and non-repudiation of origin of all other signatories: all of them have signed the contract. Bob has obtained the signatures of all other signatories on the contract, and he has obtained either the acknowledgments of all other signatories or the evidence from the smart contract functions. Therefore, the non-repudiation of receipt and non-repudiation of origin requirements are met.

7. Blockchain Cost Assessment

The signatories can finalize the contract signature without the use of the blockchain, avoiding the economic cost that it requires. In this section, we analyze the relationship between the economic cost and the time delay introduced by the use of the blockchain to assess the viability of our solution.

7.1. Gas Cost and Delay at a Fixed Time

The Ethereum nodes validate transactions and execute smart contracts in exchange for rewards (ethers, the Ethereum cryptocurrency unit). The value of these rewards can fluctuate as Ether is traded on public exchanges; thus, its price could fluctuate constantly. Ethereum uses an internal unit known as gas that allows one to know the gas cost to execute any transaction or smart contract regardless of the ether value.

As the execution of the smart contract functions implies a cost, we tested our solution using Ganache [24] because it “allows us to create a private Ethereum blockchain to perform all actions that the real Ethereum network allows but without a real cost” [12]. Besides, Truffle, a development framework, allows us to deploy and execute the smart contract on several blockchains, such as Ropsten, Rinkeby, or the blockchain created with Ganache.

To estimate the cost of our proposal in fiat currency, we have used the API from [25] that provides, at the current instant time, the estimated cost and time to publish a transaction that requires a certain amount of gas (e.g., to execute one of the functions of the contract).

As Figure 6 shows, the gas required to deploy the contract is high compared to the finish function, but once the smart contract is deployed, it can be used to manage different contract signatures (see Section 5). Table 2 lists the recommended values of the gas price (source: https://ethgasstation.info; data accessed and published on 15 November 2020) to publish a transaction in approximately 30 min (least expensive), 3 min (average), and as soon as possible (fast) [25]. As can be seen, if we require rapid contract deployment, it would not exceed $11.73 (considering a gas price of approximately 20 Gwei (Wei refers to the smallest denomination of ether (ETH), the currency used on the Ethereum network (1 ether = $10^{18}$ Wei) [12]) in approximately 27 s. If the deadline of the contract signature is not very restrictive, and we consider a gas price of 1 Gwei (requiring a delay of less than 5 h), the cost can be reduced to $0.59 (see Table 3).
Fewer signatories mean a lower cost for signing. This is mainly due to the memory required and the number of signatures to be verified. For example, the cost for two signatories is between $1.41 and $2.79, and for ten signatories it is between $5.06 and $10.03. This cost can be lowered, as with deployment, by considering lower gas prices (Table 3) and not having high time requirements. However, reducing the gas price means that the execution of the \textit{finish} function is delayed in time. Figure 7 shows the mean time to confirm the publication of the function considering a gas price between 1 Gwei and 22 Gwei. As can be seen, if the \textit{finish} function must be executed in less than 1 h, then it is necessary to pay more than $7.00. However, if the time requirement is not very critical, the publication can be completed in less than 3.5 h in the case of 2 signatories (with a cost of $0.14) and in less than 5 h in the case of 10 signatories (with a cost of $0.50).

We have not included in Figure 6 the cost of the \textit{query} function because, as explained in Section 5, the \textit{query} function (called by a signatory) does not modify the state of the contract, a transaction is not included in a block, and therefore, it does not entail any cost.
Table 3. Estimated cost (in US dollars) considering lower gas prices. Data publication date: 15 November 2020.

|                | 1 Gwei | 3 Gwei | 5 Gwei | 7 Gwei | 10 Gwei |
|----------------|--------|--------|--------|--------|---------|
| **deploy cost** |        |        |        |        |         |
| 2 signatories  | $0.59  | $1.77  | $2.96  | $4.14  | $5.92   |
| 3 signatories  | $0.14  | $0.42  | $0.70  | $0.98  | $1.41   |
| 4 signatories  | $0.18  | $0.56  | $0.93  | $1.30  | $1.86   |
| 5 signatories  | $0.23  | $0.69  | $1.16  | $1.62  | $2.32   |
| 7 signatories  | $0.27  | $0.83  | $1.39  | $1.94  | $2.78   |
| 10 signatories | $0.36  | $1.10  | $1.84  | $2.58  | $3.69   |
| 2-7 signatories| $0.50  | $1.51  | $2.53  | $3.54  | $5.06   |

**Figure 7.** Estimated publication time vs gas price of the `finish` function execution for different numbers of signatories.

7.2. Cost over Time

Above, we analyzed our solution to provide a snapshot of its cost at a fixed time. In this section, we evaluate the viability of the solution by analyzing the gas price over time. To do so, we use the average value of the gas price (paid per day and measured in Wei) provided by [26], and we provide the cost in US dollars using the exchange rate between Ether and US dollars (per day).

Figure 8 depicts the average cost per day from the launch of the Ethereum network if execution of the `finish` function is required. Analyzing the figure, we observe an increase in the gas price required in the last six months (time interval between 1 October 2020 and 1 April 2021), basically due to congestion events. Excluding this high-congestion period, the gas price is quite stable within a reasonable cost. As explained above, we can reduce the cost of the `finish` function at the expense of delaying its publication in the blockchain.
Let us analyze the cost of the last six months in greater detail. Although Figure 8 shows the average value of the gas price, a lower price can be paid, as explained above. From the graphs of the average gas price provided by [26], we can see that the minimum price paid on all days does not exceed 1 Gwei (between 1 Gwei and $10^{-9}$ Gwei, with the mean value being 0.27 Gwei). Considering the daily minimum price paid (see Figure 9), the cost to execute the finish function (when required) over the last six months is less than $0.19 for two signatories and $0.68 for ten signatories. If 1 Gwei is considered (see Figure 10), the cost over the last six months is less than $0.60 for two signatories and $2.20 for ten signatories. Therefore, we can affirm that our proposal is viable in terms of costs when the blockchain is required.

**Figure 8.** Estimated average cost of the finish function (in USD) over time (from February 2018 to April 2021).

**Figure 9.** Estimated cost of the finish function (in USD) over the last six months considering the minimum price paid in the Ethereum network. Time interval between 1 October 2020 and 1 April 2021.
Figure 10. Estimated cost of the \textit{finish} function (in USD) over the last six months considering a price of 1 Gwei paid in the Ethereum network. Time interval between 1 October 2020 and 1 April 2021.

As seen above, at times when demand for the blockchain network is high, miners on Ethereum charge high fees to publish a transaction as soon as possible. If a lower fee is paid, the execution of the smart contract functions could require several hours. Ethereum is addressing the issue, changing the current proof-of-work mining algorithm to a faster proof-of-stake mining algorithm, but it could take some time to complete [27]). Although we have chosen Ethereum to assess the viability of our solution, other blockchains that allow the execution of smart contracts and compliance with the security requirements established for the solution can be used.

8. Conclusions

In this article, we have presented a proposal for multi-party contract signing that achieves the fairness requirement without the need for TTPs based on the use of the blockchain. In addition to demonstrating compliance with this property, we have demonstrated that the following are met: bc-effectiveness, timeliness, non-repudiation of origin, and non-repudiation of receipt. We have also seen that the confidentiality property can be easily enforced (although it is not introduced to reduce the complexity in presenting the protocol). Although a public blockchain is used, this does not mean the possible loss of confidentiality of the contract since published data do not reveal the content of the contract.

Finally, the use of Ethereum entails a cost only if any signatory requires execution of the \textit{finish} function. In these situations, the proposal requires only one interaction with the blockchain regardless of the number of signatories. The design of a proof of concept has allowed us to verify the feasibility of our solution. The implemented Ethereum smart contract has been presented, and results have been provided in terms of gas consumption and economic cost. Because the protocol implies a time restriction, delays in the publication in the blockchain of the \textit{finish} function execution have been analyzed. As a result, we can affirm that the cost can be modulated according to the temporal requirements of the scenario. We have conducted an analysis of the cost required over time, and the results show that the cost of the solution is within perfectly manageable margins.

Author Contributions: Conceptualization, J.-L.F.-G. and M.F.H.; Investigation, J.-L.F.-G. and M.F.H.; Methodology, J.-L.F.-G. and M.F.H.; Supervision, J.-L.F.-G. and M.F.H.; Validation, J.-L.F.-G. and M.F.H.; Writing—original draft, J.-L.F.-G. and M.F.H.; Writing—review and editing, J.-L.F.-G. and M.F.H. All authors have read and agreed to the published version of the manuscript.
Funding: This research was supported by the European Social Fund (ERDF) and the Spanish Government (Ministerio de Ciencia e Innovación (MCI)—Agencia Estatal de Investigación (AEI)) under projects RTI2018-097763-B-I00 and RED2018-102321-T.

Data Availability Statement: The study did not report any data.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ben-Or, M.; Goldreich, O.; Micali, S.; Rivest, R.L. A fair protocol for signing contracts. *IEEE Trans. Inf. Theory* **1990**, *36*, 40–46. [CrossRef]
2. Asokan, N.; Schunert, M.; Waidner, M. Optimistic protocols for fair exchange. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, Zurich, Switzerland, 1–4 April 1997; pp. 7–17.
3. Bao, F.; Wang, G.; Zhou, J.; Zhu, H. Analysis and improvement of Micali’s fair contract signing protocol. In *Proceedings of the Australasian Conference on Information Security and Privacy*, Sydney, Australia, 13–15 July 2004; pp. 176–187.
4. Ferrer-Gomila, J.L.; Payeras-Capella, M.; Huguet-Rotger, L. Optimality in asynchronous contract signing protocols. In *Proceedings of the International Conference on Trust, Privacy and Security in Digital Business*, Zaragoza, Spain, 30 August–1 September 2004; pp. 200–208.
5. Zhou, J.; Deng, R.; Bao, F. Some Remarks on a Fair Exchange Protocol. In *Lecture Notes in Computer Science, Proceedings of the Public Key Cryptography, Victoria, Australia, 18–20 January 2000*; Springer: Berlin/Heidelberg, Germany, 2000; Volume 1751, pp. 46–57.
6. Banerjee, P.; Ruj, S. Blockchain Enabled Data Marketplace—Design and Challenges. *CoRR arXiv* **2018**, arXiv:1811.11462.
7. Mukhamedov, A.; Ryan, M.D. Fair multi-party contract signing using private contract signatures. *Inf. Comput.* **2008**, *206*, 272–290. [CrossRef]
8. Kordy, B.; Radomirovic, S. Constructing optimistic multi-party contract signing protocols. In *Proceedings of the 2012 IEEE 25th Computer Security Foundations Symposium*, Cambridge, MA, USA, 25–27 June 2012; pp. 215–229.
9. Ferrer-Gomila, J.L.; Hinarjeros, M.F. A 2020 perspective on “A fair contract signing protocol with blockchain support”. *Electron. Commer. Res. Appl.* **2020**, *42*, 100981. [CrossRef]
10. Huang, H.; Li, K.C.; Chen, X. A Fair Three-Party Contract Singing Protocol Based on Blockchain. In *International Symposium on Cyberspace Safety and Security*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 72–85.
11. Hinarjeros, M.F.; Ferrer-Gomila, J.L. A Solution for Secure Multi-Party Certified Electronic Mail Using Blockchain. *IEEE Access* **2020**, *8*, 102997–103006. [CrossRef]
12. Ferrer-Gomila, J.L.; Hinarjeros, M.F.; Isern-Deyà, A.P. A fair contract signing protocol with blockchain support. *Electron. Commer. Res. Appl.* **2019**, *36*, 100869. [CrossRef]
13. Wan, Z.; Deng, R.H.; Lee, D. Electronic contract signing without using trusted third party. In *International Conference on Network and System Security*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 386–394.
14. Huang, H.; Li, K.C.; Chen, X. Blockchain-based fair three-party contract signing protocol for fog computing. *Concurr. Comput. Pract. Exp.* **2019**, *31*, e4469. [CrossRef]
15. Tian, H.; He, J.; Fu, L. Contract Coin: Toward Practical Contract Signing on Blockchain. In *Proceedings of the International Conference on Information Security Practice and Experience*, Melbourne, VIC, Australia, 13–15 December 2017; Springer: Berlin/Heidelberg, Germany, 2017; pp. 43–61.
16. Wang, D.; Li, Q.; Li, F.; Zhang, Q.; Xu, B. Privacy-Awareness Fair Contract Signing Protocol Based on Blockchain. In *International Symposium on Cyberspace Safety and Security*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 274–278.
17. Mut-Puigserver, M.; Payeras-Capellà, M.M.; Cabot-Nadal, M.A. Blockchain-Based Contract Signing Protocol for Confidential Contracts. In *Proceedings of the 2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*, Abu Dhabi, United Arab Emirates, 3–7 November 2019; pp. 1–6.
18. Zhang, L.; Zhang, H.; Yu, J.; Xian, H. Blockchain-based two-party fair contract signing scheme. *Inf. Sci.* **2020**, *535*, 142–155. [CrossRef]
19. Wood, G. Ethereum: A Secure Decentralised Generalised Transaction Ledger EIP-150 REVISION (759dccd-2017-08-07). 2017. Available online: https://ethereum.github.io/yellowpaper/paper.pdf (accessed on 15 April 2021).
20. American National Standards Institute. *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*. *ANS X9* **2005**, *65*, 2005. [CrossRef]
21. National Institute of Standards and Technology. *FIPS-186-5: Digital Signature Standard (DSS)*. *FIPS PUB* (accessed on 15 April 2021).
22. Truffle Blockchain Group. Truffle Suite. Available online: https://www.trufflesuite.com/ (accessed on 15 April 2021).
23. Elaine Barker (NIST). *NIST-SP 800-57: Recommendation for Key Management, Special Publication 800-57 Part 1 Rev. 5*; National Institute of Standards and Technology: Gaithersburg, M, USA, 2020. [CrossRef]
24. Station, E.G. Ether Gas Station. Available online: https://ethgasstation.info/ (accessed on 15 April 2021).
26. Etherscan. Block Explorer and Analytics Platform for Ethereum. Available online: https://etherscan.io/ (accessed on 15 April 2021).
27. Ethereum.org. Consensus Mechanisms. Available online: https://ethereum.org/en/developers/docs/consensus-mechanisms/ (accessed on 15 April 2021)