computers

MDPI

Article

A Hard-Timeliness Blockchain-Based Contract Signing Protocol

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: In this article, we present the first proposal for contract signing based on blockchain that meets the requirements of fairness, hard-timeliness, and bc-optimism. The proposal, thanks to the use of blockchain, does not require the use of trusted third parties (TTPs), thus avoiding a point of failure and the problem of signatories having to agree on a TTP that is trusted by both. The presented protocol is fair because it is designed such that no honest signatory can be placed at a disadvantage. It meets the hard-timeliness requirement because both signatories can end the execution of the protocol at any time they wish. Finally, the proposal is bc-optimistic because blockchain functions are only executed in case of exception (and not in each execution of the protocol), with consequent savings when working with public blockchains. No previous proposal simultaneously met these three requirements. In addition to the above, this article clarifies the concept of timeliness, which previously has been defined in a confusing way (starting with the authors who used the term for the first time). We conducted a security review that allowed us to verify that our proposal meets the desired requirements. Furthermore, we provide the specifications of a smart contract designed for the Ethereum blockchain family and verified the economic feasibility of the proposal, ensuring it can be aligned with the financial requirements of different scenarios.

Keywords: blockchain technology; contract signing; fair exchange; timeliness; EVM-based blockchain; trust-free system; e-commerce; blockchain cost

Citation: Ferrer-Gomila, J.-L.; Hinarejos, M.F. A Hard-Timeliness Blockchain-Based Contract Signing Protocol. *Computers* **2023**, *12*, 246. https://doi.org/10.3390/computers12120246

Academic Editors: Nino Adamashvili, Caterina Tricase, Otar Zumburidze, Radu State and Roberto Tonelli

Received: 17 October 2023
Revised: 16 November 2023
Accepted: 17 November 2023
Published: 24 November 2023

1. Introduction

Contract signing is an essential process in commerce in general and in electronic commerce in particular. Therefore, it is necessary to design protocols for contract signing that are secure. The fundamental security requirements are fairness, timeliness, and non-repudiation. Traditionally, the fairness requirement (no signatory can be at a disadvantage during the contract signing process) has been achieved with the assistance of trusted third parties (TTPs). However, these TTPs can become a point of failure for the protocol, and it can also be difficult for the parties to agree on a TTP that is trusted by both.

Recently, we have witnessed the incorporation of blockchain into multiple processes to provide transparency and efficiency in various business transaction scenarios, such as energy trading [1], e-commerce [2], healthcare [3], and spectrum sharing [4], and contract signing has become aligned with the adoption of blockchain technology. Service level agreements (SLAs) are a type of contract that establish service expectations. These SLAs could be automated through smart contracts in blockchains, which allow for the definition of terms and conditions in agreements, triggering actions automatically when the specified conditions are met. Blockchain technology provides a transparent and immutable record of all transactions and events, making it an interesting option for efficiently verifying and monitoring contract signing procedures [5,6].

We found multiple proposals for contract signing based on blockchain in the literature. However, the use of public blockchains entails a cost for those involved in the signing of a contract. Therefore, our objective was to design a solution for contract signing that meets

the bc-optimistic requirement; that is, that blockchain functions are only executed in case of exception and not in each protocol run.

Regarding the timeliness requirement, the bibliography is confusing in its definition. On the one hand, it is defined as guaranteeing that the execution of the protocol ends within a finite time, and on the other hand, it is defined as guaranteeing that the signatories can decide the moment at which the execution of the protocol ends; the same term is used for both definitions. Our objective was, in addition to clarifying the definition of the timeliness requirement, to design a protocol that meets the second definition, which is more restrictive, and which we have called *hard-timeliness*.

We have not found any proposal for contract signing based on blockchain that meets the three requirements: fairness, hard-timeliness, and bc-optimism. This was the fundamental objective of this work, and thus we present a protocol that satisfies these three requirements, in addition to the non-repudiation and confidentiality requirements.

We also want to demonstrate the practical viability of the proposal, and to do so, we present a smart contract code for the Ethereum blockchain family. This allows us to provide the cost of executing the functions and to prove that the proposal, in addition to being secure, is viable from a practical perspective.

Contributions. We provide the first protocol for contract signing based on blockchain that, in addition to meeting the mandatory security requirements (fairness, timeliness, and non-repudiation), meets the optional requirements of confidentiality and bc-optimism. Regarding the timeliness requirement, a review of definitions used to date is provided. Our solution satisfies the most restrictive definition (which we call *hard-timeliness*). Moreover, an analysis of all proposals for contract signing based on the blockchain was performed. Finally, we conducted an economic cost analysis of our solution, to verify its feasibility.

Organization. This paper is organized as follows. Section 2 provides a review of the timeliness definitions used to date. The related work in the literature is analyzed in Section 3. Section 4 introduces the blockchain technology features. Section 5.1 outlines our proposal for *hard-timeliness* in contract signing, followed by a full specification of the protocol in Section 5.2. The smart contract execution logic is provided in Section 6. A security review of our proposal is conducted in Section 7, and a cost analysis is performed in Section 8. Finally, the conclusions are presented in Section 10.

2. Timeliness

The first article in which the term *timeliness* appeared was [7], where it is defined as follows: "at the beginning of the exchange, P [a signer] can be certain that the protocol will be completed at a finite point in time; at completion, the state of the exchange as of that point is either final or any changes to the state will not degrade the level of fairness achieved by P thus far". In the same article, the authors define *timely conclusion*: "at any time during a protocol run, either player can unilaterally choose to force an end to the protocol without losing fairness". The two definitions pursue the same goal, but they are not identical. Although the authors use two terms, from the rest of the explanations in the article, nothing suggests that their objective was to define two requirements with different characteristics. These definitions led us to the initial considerations that we develop in the following paragraphs.

The main difference between the two previous definitions is the temporal aspect: "at a finite point in time" vs. "at any time". A solution based on the establishment of deadlines may meet the first definition (at a finite point in time) and not meet the second definition (at any time). Clearly, if a proposal meets the "at any time" requirement, then it also meets the "at a finite point in time" requirement. Therefore, we propose to use the term *hard-timeliness* for the definition of "at any time" (we do not propose *strong-timeliness* because other authors have used the term for other definitions) and the term $soft$-*timeliness* for the definition of "at a finite point in time".

Some authors (e.g., [8–15]) restricted compliance using the requirement of honest signers. This approach seems absolutely reasonable, since we should not worry about what

happens to dishonest signatories. In any case, if a proposal meets the requirement for all signatories, then the proposal meets the requirement for honest signatories.

We can also observe a difference in the action of the signer. In the *hard-timeliness* definition, the signer has an active role: he or she forces completion. In the $soft$-*timeliness* definition the signer has a passive role: "... the protocol ends". This fact is closely linked to the temporal aspect ("at any time" vs. "at a finite point in time").

Note that one of the two definitions specifies "unilaterally" and the other does not. It should be understood that, unless otherwise stated, the decision to terminate/abandon the execution of the protocol can be made unilaterally. Again, the difference between the two definitions is related to the temporal aspect. In the "at any time" case, the signer must perform an action, and most likely, this is why Asokan et al. took care to make explicit that the signer should not depend on the actions of others to be able to end the protocol run.

Both definitions end with a reference to no loss of fairness. We believe that this is not harmful, but it is not strictly necessary. Recall that the fairness requirement is the fundamental demand that any contract signing proposal must meet. If a proposal, for whatever reason, causes an honest signatory to be placed in an unfair situation, then it is an invalid proposal [16] (in this case, it is inconsequential whether the timeliness requirement is met).

In a previous article [17], the same authors provided two other definitions (reproduced in a later article [18]), although without using the term *timeliness*. The first reads as follows: a player can always force a timely and fair termination without the cooperation of the other player. While the second definition is as follows: one player cannot force the other to wait for any length of time—a fair and timely termination can always be forced by contacting the third party. The second definition clearly states the problem to be solved: a signatory does not have to wait an indefinite time to finish the protocol. Both definitions appear to better accord with the definition we have called *hard-timeliness* (a signer can force completion). We want to highlight an element that we consider negative in the second definition. This definition includes how the requirement must be satisfied: "contacting the third party" (similarly stated by other authors [19–22]). We believe that the mechanisms should be left to the choice of the authors of the proposals.

Since then, multiple authors have used one definition or another, using the term *timeliness* or, alternatively, a term similar to *timely protocol*. Some authors have even made some significant changes to the definition using the same term *timeliness*. Some authors [19,21–31] used definitions that fit the definition we have called hard-timeliness. While other authors [8,9,13,15,32–45] used definitions that fit the definition we have called soft-timeliness. Some introduced nuance: predetermined time [34,37], agreed time [35,36], bounded time [39], or that the protocol defines a deadline [45]. Finally, some authors [46–48] presented both types of definition.

Some authors [10,11,27,49–57] redefined the concept of *timeliness* (or *timely protocol*), leaving the temporal aspect unspecified: each party has some recourse to avoid/prevent/stop unbounded/endless waiting. Alternatively, we find [58]: there exists some mechanism to ensure termination.

The authors in [12] argued that users of contract signing protocols are not typically experts and that they should be relieved of certain responsibilities when executing a protocol. For this reason, these authors defined a new requirement that they called *strong-timeliness*: at any moment in an ongoing protocol run, an honest party P can be certain that the protocol will be automatically completed at a certain point in time; if any action is required from P, it should be clearly stated, along with the circumstances in which it should be taken.

We believe that it would be desirable to standardize the nomenclature (to avoid erroneous interpretations). In any case, all authors should clearly state the definition they use in their proposal.

3. Related Work

After performing a search via the Web of Science, Scopus, and Scholar for the terms "blockchain" and "contract signing", 14 articles containing proposals for contract signing based on blockchain were obtained. These articles make proposals for different scenarios:

- two-party [59–64]
- three-party [65–67]
- multi-party [68–71]
- multi-two-party [72]

Although our goal was to provide a two-party protocol, all articles were considered in our analysis, since "multiparty" cases can be reduced to two-party cases (interestingly, proposals for the three-party scenario are the most difficult to convert, because they are specifically designed for contracts among three signatories).

The first observation is that 7 [59,62–64,67,70,71] of the 14 articles did not even mention a *timeliness* requirement. We find this surprising because, unlike other requirements that are optional (such as the confidentiality requirement), the *timeliness* requirement should be met by any proposal of contract signing. More serious is the fact that three proposals [63,67,70] did not meet the *timeliness* requirement.

Most authors [60,65,66,68,69,72] presented a definition corresponding to $soft$-*timeliness*. Notably, Ref. [68] clarified that the (maximum) end time should be known a priori.

After analyzing the proposals (whether the authors considered the timeliness requirement), we observed that the majority [59–62,65,66,68,69,71,72] met the $soft$-*timeliness* requirement through the establishment of deadlines in the protocol specification. The only exception was found in [72], where two proposals were presented, and one met the *hard-timeliness* requirement. However, this proposal did not meet the bc-optimistic requirement (another requirement we want our proposal to satisfy).

In addition to the temporal aspect, we observed two trends in the analyzed articles. Some authors [59,61,62,64–66,69,73] based fairness, totally or partially, on penalties for dishonest signers (deposits must be made in an initial phase of the protocol). We consider that this is not an adequate mechanism, since it is not easy to determine a reasonable economic amount that discourages fraud without discouraging contracting. The case of proposals [65,66,69] in which the deposits are asymmetrical (not all signatories must deposit the same amount) appears more serious to us, since a degree of unfairness is introduced a priori.

The other trend we observed is that most proposals [59–67,69–72] execute blockchain functions in all protocol runs, and most of them do so intensively. We must not forget that, in public blockchains, the execution of blockchain functions involves a cost (which must be assumed by the signatories). Therefore, we believe that it is beneficial to develop proposals that meet the *bc-optimism* requirement; that is, that blockchain functions are executed only in the case of an exception. We found only two proposals [68,72] that met this requirement, but as we have already indicated, they did not meet the *hard-timeliness* requirement.

In short, we did not find any proposals that satisfied all the requirements that we wanted to meet: fairness, hard-timeliness, non-repudiation, confidentiality and bc-optimism.

4. Technological Background

In this section, we explain the different types of blockchain technologies and determine which is most suitable for our proposed solution. Furthermore, we explore how smart contracts work and store information within the blockchain environment. This is an important point in determining the most suitable data storage method to be used for the evidence in the contract signing protocol.

4.1. Blockchain Overview

Blockchain technology has become a transformative force, providing innovative solutions for reducing dependence on TTPs in a variety of scenarios, such as energy [1],

e-commerce [2], healthcare [3], and dynamic spectrum sharing in 6G technology [4]. At its core, blockchain can be described as a distributed ledger system that operates transparently within the nodes of a computer network, commonly known as miners. In this context, the data recorded in the general ledger remain immutable.

Blockchain can be broadly classified as public or private and permissionless or permissioned [74]. In public blockchains, anyone can participate in the network, and no single entity controls it, creating a permissionless and trustless environment. This feature provides enhanced security through a large number of participating nodes. In contrast, private blockchains are governed by a central authority responsible for managing access, making them authorized and trustworthy. Unlike their public counterparts, private blockchains limit active participation to a limited number of authorized nodes, thereby raising potential security concerns, due to this centralized control.

Although initially associated with cryptocurrencies, blockchain has evolved to support many applications, largely thanks to the development of smart contracts. Smart contracts can be described as self-executing code that triggers actions when specific events occur [75]. Running such code will incur costs, which depend on the computational complexity of the tasks involved [76]. Moreover, associated data are stored on the blockchain, which serves as a decentralized and distributed ledger that records all data associated with the smart contract. Notably, while data are stored on the blockchain, private or sensitive information should be handled with care and must not be stored on-chain [77].

Ethereum, a pioneering public blockchain, has revolutionized the execution of smart contracts with the introduction of the Ethereum Virtual Machine (EVM) [78]. Initially adapted for Ethereum, EVM has transcended its origins and is now adopted by other blockchains such as Polygon, Binance Smart Chain (BSC), and Arbitrum. This interoperability provides a number of benefits to developers, including the seamless migration of smart contracts from one EVM-compatible blockchain to another, expanding blockchain-based applications.

4.2. Smart Contract Data Storage

Smart contracts can hold and manage assets or data and are typically composed of functions that can be executed [79]. When a function within a smart contract is triggered, it performs a specific action based on predefined logic, such as transferring cryptocurrency, updating data, or verifying conditions. Solidity [80], a programming language for smart contracts, provides three distinct types of memory that enable developers to manage the storage of variables in the EVM [78,81]: *memory*, *call data*, and *storage*.

The *memory* is used for variables and parameters within the scope of a function. These variables only exist during function execution and are deleted at the end of function execution. The memory is similar to that familiar to programmers with a background in traditional coding.

Call data is similar to memory and is essential when setting dynamically sized parameters in an external function signature. Unlike memory, *call data* variables are read-only and refer to an area of memory that cannot be modified.

Storage represents the long-term memory of a contract, preserving variables even after a function or transaction has ended. State variables, those declared in the contract but outside of any functionality, are stored in the storage memory area. This concept is unique to blockchain, as smart contract data are secured with cryptographic properties, ensuring tamper-proof data persistence directly on the blockchain.

5. Hard-Timeliness Contract Signing Protocol

5.1. Our Proposal in a Nutshell

In this section, we give the security requirements that our proposal must meet, and we present a summarized vision of our proposal. Table 1 defines the notation used in the explanation of the protocol.

Our contract signature solution involves two signatories (A and B) and exceptionally (in case of conflict) the use of a smart contract deployed on an EVM-compatible blockchain (Figure 1). In this scenario, our solution must meet the following requirements [7,23,72,82]:

- *Fairness*: No honest signer should be disadvantaged;
- *Hard-timeliness*: At any time, a signatory can terminate the execution of the protocol;
- *Non-repudiation*: The signatories should not be able to deny their actions once the execution of the protocol is finished;
- *Effectiveness*: No TTP should be involved in the protocol;
- *Confidentiality*: The content of the contract should only be known by the signatories;
- *Bc-optimistic*: Blockchain functions should only be executed in case of exception (and not in each protocol run).

Our proposal consists of two subprotocols: *exchange* and *resolution*. Under normal conditions, only the four-step *exchange* subprotocol should be executed. First, A must send a signed copy of the contract to B. B must then send his signature to A. A must confirm that she has received B's signature, and finally B must confirm that he has received A's confirmation. The evidence that the contract has been signed are the signature-confirmation pairs. Note that no blockchain function has been executed.

If A does not receive confirmation from B, then she must execute the *resolution* function of the blockchain, requesting to finish the contract signing, and providing evidence of the first three steps of the *exchange* subprotocol. If everything is correct and B has not canceled, the smart contract will record the evidence provided by A and mark the status of the contract signing as finished.

If after B sends his signature, he notices that he has not received confirmation from A, then he must execute the *resolution* function of the blockchain, requesting to cancel the contract signing. If everything is correct and A has finished the contract signing, the smart contract will inform B that the contract signing is finalized, and he can obtain evidence of the fact. If everything is correct and A has not finished the contract signing, the smart contract will record the evidence provided by B and mark the status of the contract signing as canceled.

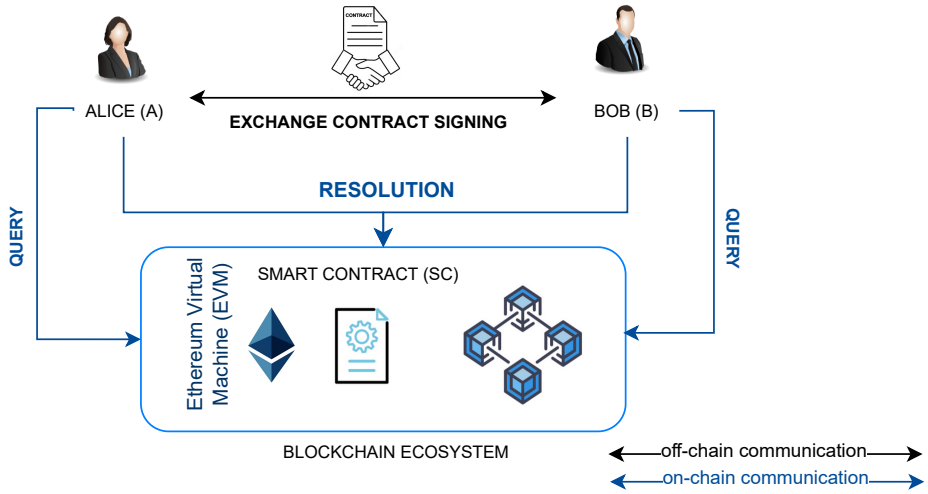

Figure 1. Hard-timeliness contract signing scenario.

Table 1. Hard-timeliness contract signing protocol notation.

A and B	Signatories
SC	Smart contract
$@X$	Blockchain address of entity X
M	Content of the contract
$H()$	One-way collision-resistant hash function
id_{AB}	Identifier of the contract signing
$Sig_X(y)$	Signature on element y made by entity X
f_X	signature on the contract signing agreement made by entity X
ACK_X	signature on the contract signing confirmation made by entity X
$status$	the status of the contract signing reflected on the SC: $finished$ or $canceled$

5.2. Design Specifications of Our Proposal

In this section, we provide a detailed description of the two subprotocols mentioned in Section 5.1. In the "Security Review" section, we show that, with these two subprotocols, compliance with the established requirements is guaranteed.

5.2.1. Exchange Subprotocol

The *exchange* subprotocol is always executed between the two signatories (A and B) when a contract signing occurs, and this involves the following four steps (as illustrated in Figure 2).

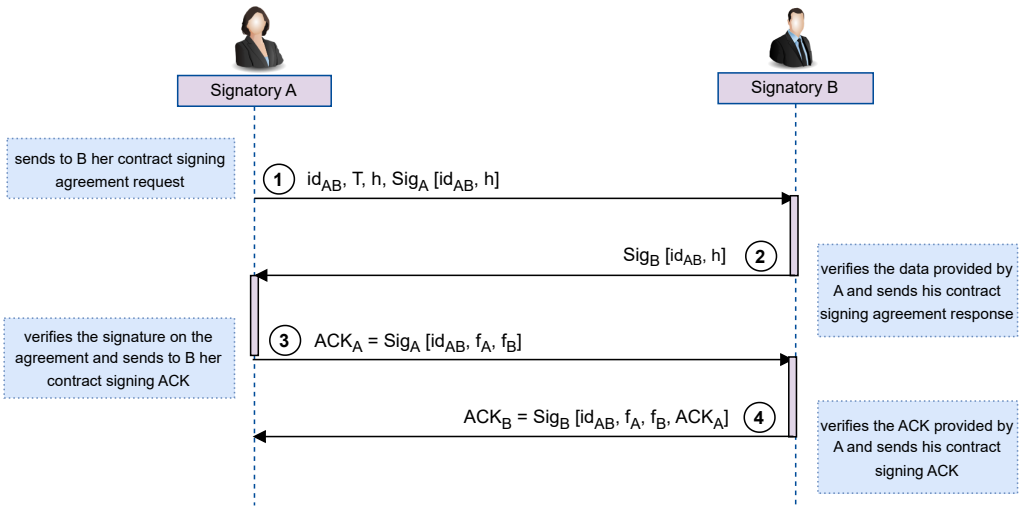

Figure 2. Hard-timeliness protocol: *exchange* subprotocol.

Step 1. A sends her contract signing agreement to B conveying the following information:

- a unique identifier of the transaction, id_{AB};
- a timestamp indicating the current date and time, T;
- the hash of the contract, $h = H(M)$;
- the signature on the data of the contract signing agreement, $f_A = Sig_A[id_{AB}, h]$.

The unique identifier is calculated as follows:

$$id_{AB} = H(@A, @B, @SC, T, h)$$

In this calculation, we introduce the T value because, if a protocol run is aborted and then a new execution is started to sign the same contract, there will be no confusion between the two executions (the identifiers will be different).

If B does not want to sign the contract, he can ignore the received message; otherwise, he must proceed with step 2.

Step 2. B must recalculate the identifier and verify A's signature. If the above data are correct, he must send his response to the agreement to A, conveying the following information:

- his signature on the data of the contract signing agreement (id_{AB} and h):

$$f_B = Sig_B[id_{AB}, h]$$

Then, if A and B are honest, they must complete the execution of the *exchange* subprotocol.

Step 3. A must send her contract signing confirmation to B with the following information:

- the signature on the data of the contract signing confirmation (id_{AB}, f_A and f_B):

$$ACK_A = Sig_A[id_{AB}, f_A, f_B]$$

Step 4. Finally, B must send his contract signing confirmation to A with the following information:

- the signature on the data of the contract signing confirmation (id_{AB}, f_A, f_B and ACK_A):

$$ACK_B = Sig_B[id_{AB}, f_A, f_B, ACK_A]$$

If the two signatories have followed the four steps of the exchange subprotocol, both signatories have evidence that the contract has been signed. A has f_B and ACK_B, and B has f_A and ACK_A; no blockchain function has been executed.

5.2.2. Resolution Subprotocol

In the course of the *exchange* subprotocol execution, certain circumstances may arise, whether deliberately or unexpectedly, that could result in non-completion of the contract signing. To maintain the security requirements, a smart contract is designed to handle such circumstances. A and B may request contract signing resolution, whether finalization or cancellation, by providing the necessary evidence (see Figure 3).

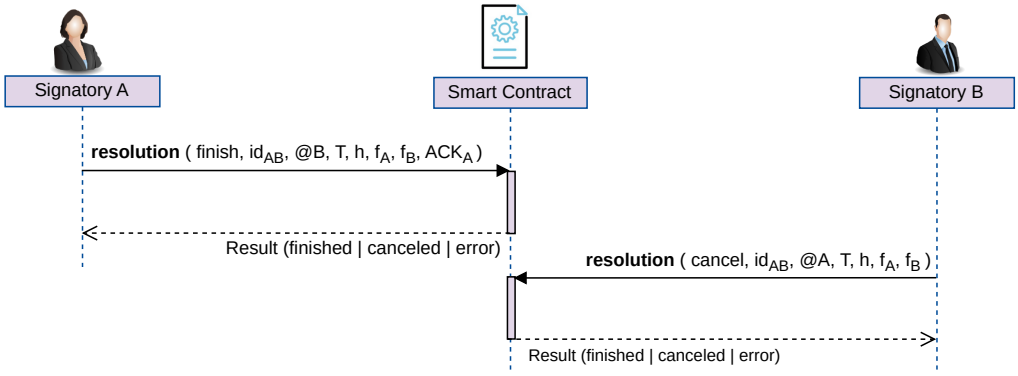

Figure 3. Hard-timeliness protocol: *resolution* subprotocol.

Requesting finalization. If A observes that she has not received confirmation from B (step 4), she must call the blockchain's *resolution* function to request finalization of the contract signing. Recall that B could have previously canceled the contract signing (see Section 5.1). Below, we detail the steps.

Step 1. A requests finalization of the contract signing by calling the *resolution* function with the following parameters:

$$resolution(finish, id_{AB}, @B, T, h, f_A, f_B, ACK_A)$$

Step 2. The *resolution* function performs the following checks and operations:

1. It verifies if the contract signing identified by id_{AB} was previously finished or canceled; in this case, the smart contract reports the status of the contract signing to A and ends the execution of the function. A can recover evidence associated with id_{AB} by calling the *query* function (see the "requesting information" point explained below);

2. It recalculates id_{AB} using the parameters provided by A ($@B, T, h$) and the information recovered by the smart contract ($@A$, SC);
3. It checks whether the evidence provided is correct (f_A, f_B, ACK_A); that is, the signatures are valid; if they are invalid, the smart contract informs A and ends execution of the function;
4. It stores the ACK_A parameter to resolve a possible future request from B and updates the status of the contract signature to `finished`. In this way, the resolution transaction is recorded in the blockchain.

Requesting cancellation. If B observes that he has not received confirmation from A, he must execute the blockchain's *resolution* function to request cancellation of the contract signing. Recall that A could have previously finished the contract signing (see Section 5.1). Below, we detail the steps.

Step 1. B requests execution of the *resolution* function with the following parameters:

$$resolution(cancel, id_{AB}, @A, T, h, f_A, f_B)$$

Step 2. The *resolution* function performs the following checks and operations:

1. It verifies if the contract signing identified by id_{AB} was previously finished or canceled; in this case, the smart contract reports the status of the contract signing to B and ends the execution of the function. B can recover evidence associated with id_{AB} by calling the *query* function (see the "requesting information" point explained below);
2. It recalculates id_{AB} using the parameters provided by B ($@A, T, h$) and the information recovered by the smart contract ($@B$, SC);
3. It checks whether the evidence provided is correct (f_A, f_B); that is, if the signatures are valid; if they are invalid, the smart contract informs B and ends the execution of the function;
4. It stores f_A and f_B to resolve a possible future request from A and updates the status of the contract signature to `canceled`. In this way, the resolution transaction is recorded in the blockchain.

Requesting information. Given an identifier id_{AB}, any signatory can, at any time, track the status of the contract signature and gather the evidence provided by the party who requested the finalization or cancellation of the contract signing.

The *query* function (see Figure 4) takes an argument, the exchange identifier for a specific contract signature, id_{AB}. When the *query* function is called, it provides information about the current status of the contract signing associated with id_{AB}. If none of the signatories previously requested finalization or cancellation, there are no data associated with this id_{AB}, and SC returns an error. However, if a signatory successfully requested finalization or cancellation, the status is finished or canceled, respectively. In these cases, the smart contract provides the evidence associated with this id_{AB}.

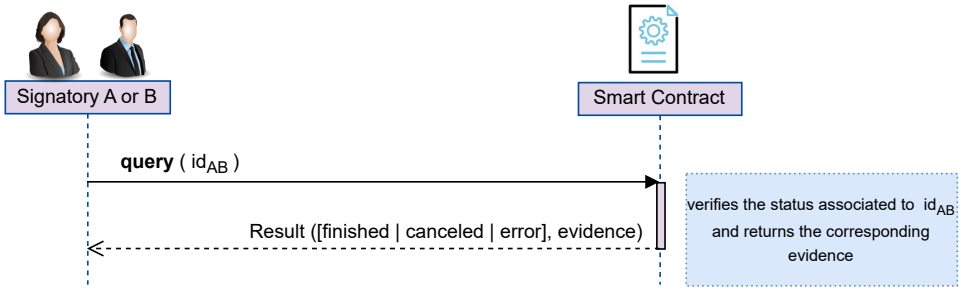

Figure 4. Hard-timeliness protocol: *query* subprotocol.

6. Smart Contract Specification

The solution presented in Section 5.2 is built upon blockchain technology, where the use of a smart contract is required when problems arise during the *exchange* subprotocol. In this section, we outline the specific smart contract code necessary for implementing each of the functions required to handle these situations.

Solidity [80] is a high-level programming language specifically designed to facilitate the creation of self-executing and self-enforcing contracts within the decentralized blockchain ecosystem. Therefore, we use Solidity to define the rules and logic of the smart contract, because it is the preferred programming language for developing smart contracts in blockchain-based distributed applications.

6.1. Data Structure Definition

In the smart contract, we define a data structure indexed by id_{AB} (see Section 5.2.1). This structure is designed to store data associated with each contract signing and includes four fields, as shown in Listing 1. These fields include the contract signing agreement values (f_A and f_B) provided by the signatory calling the *resolution* function, A's contract signing confirmation (ACK_A) when required, and the current status of the contract signing process (*status*) (see Table 1 and Sections 5.2.1 and 5.2.2).

These data hold significant importance within our solution, since they must be accessible to both signatories whenever necessary. This serves as tangible evidence of the contract signing status. Therefore, we use *storage* memory to ensure tamper-proof data persistence (see Section 4.2).

Listing 1. Hard-timeliness in contract signing: data structure

```
1
2  enum State { undefined, finished, canceled }
3  mapping(bytes32 => dataHTCS) htContracts;
4
5  struct htContracts{
6  bytes fA;
7  bytes fB;
8  bytes ackA;
9  State status;
10 }
```

6.2. Signature Validation

In our proposal, one of the critical validations is digital signature verification. Due to the critical nature of this process, we are required to implement it with the strictest security measures. OpenZeppelin [83] is a company specializing in cryptocurrency cybersecurity technology and services. They offer secure and audited smart contracts and libraries that have established themselves as industry standards. They also offer open-source code templates that have had widespread adoption and rigorous testing, reducing the risks associated with deploying cryptographic solutions.

We employ the openZeppelin function `ECDSA.recover` to obtain the address of the signer of specific data to authenticate the evidence (such as f_A, f_B, and ACK_A) submitted by each signatory. For this purpose, we define the function *validSign* (see Listing 2) to evaluate the authenticity of a signature, taking into account the data, the signature on the provided data, and the signer. This function returns `true` when the signature is valid and `false` otherwise.

Listing 2. Hard-timeliness in contract signing: `validSign` function

```
address recover(bytes32 hash, bytes signature)

function validSign(bytes32  hash, bytes memory signature, address signer)
    private pure returns(bool){
return (ECDSA.recover(toEthSignedMessageHash(hash), signature) == signer);
}
```

6.3. Main Functions Definition

Functions in a smart contract perform specific tasks or sets of actions when called. They are typically defined by specific inputs (parameters) and may return outputs or modify the internal state of the contract [81].

Our proposal requires two main functions to allow signatories to interact and manipulate the state and functionality of the smart contract. Following the description of the proposal (see Section 5.2), *SC* provides two public functions:

- *resolution* (Listing 3): Given a transaction identifier, A can finish the contract signature by providing signatures on the contract agreement (her own signature and B's signature) and her contract signing acknowledgment (ACK_A); B can cancel the contract signature by providing signatures on the contract agreement: his own signature and A's signature;
- *query* (Listing 4): Given a transaction identifier, A and B can check the status of the transaction and obtain the required data. If the contract was finished by A, B obtains evidence of A's signature. If the contract was canceled by B, A obtains evidence of this fact.

Listing 3. Hard-timeliness in contract signing: *resolution* function

```
function resolution(State _type, bytes32 _idAB, address _addr, uint _T,
    bytes32 _hashM, bytes memory _fA, bytes memory _fB, bytes memory _ackA)
validStatus(_idAB) validRequest(_type) public {
if( _type == State.finished) {
checkProofs(_idAB, msg.sender, _addr, addrSC, _T, _hashM, _fA, _fB);
bytes32 hashACK = keccak256(abi.encodePacked(_idAB, _fA,_fB));
require(validSign(hashACK,_ackA,msg.sender), ''Invalid ACK'');
htContracts[_idAB].ackA = _ackA;
} else {
checkProofs(_idAB, _addr, msg.sender, addrSC, _T, _hashM, _fA, _fB);
}
htContracts[_idAB].status = _type;
htContracts[_idAB].fA = _fA;
htContracts[_idAB].fB = _fB;

emit Result(msg.sender, _idAB, uint(_type));
}
```

The execution of the *resolution* function is controlled by modifiers. Modifiers [80] are code that can be run before and/or after a function call and can be used to restrict access, validate inputs, etc. We have defined the following two modifiers:

- *validRequest*: given a request identifier type, this modifier checks if the type of request is valid: finish or cancel.
- *validStatus*: given a transaction identifier, this modifier checks whether the status of the contract allows the execution of the *resolution* function; that is, the contract signing is neither finalized nor canceled.

Listing 4. Hard-timeliness in contract signing: *query* function

```
function query(bytes32 _idAB) view public returns (State, bytes memory,
    bytes memory, bytes memory){
State status = htContracts[_idAB].status;
if(status == State.finished){return (status, htContracts[_idAB].fA,
    htContracts[_idAB].fB, htContracts[_idAB].ackA);}
else if(status == State.cancelled){return (status, htContracts[_idAB].fA,
    htContracts[_idAB].fB, ‘‘’’);}
else{revert(status);}
}
```

After successful completion of the above validations, the *resolution* function calls the private function *checkProofs* (as shown in Listing 5). This private function is responsible for conducting the following tasks:

- calling the private function *validID*, to check whether the given transaction identifier is valid. To achieve this, the *SC* generates id_{AB} using both the blockchain address of the signer of the transaction and its own address. It then compares this derived value with the identifier supplied as a parameter by the signatory to determine whether they match;
- calling the *validSign* private function (defined in Section 6.2) to ensure the validity of the evidence provided by the signatory: f_A and f_B.

If the signatory requested finalization, in addition to the previous signature validations, the *resolution* function must also validate the ACK_A signature by calling the *validSign* private function.

After all validations have been successfully completed, the *resolution* function stores the evidence provided by the signatory who called the *resolution* function and updates the status of the contract signature to finalized or canceled, based on who called the function and what evidence was provided.

Listing 5. Hard-timeliness in contract signing: *checkProofs* function

```

function checkProofs(bytes32 _idAB, address _addrA, address _addrB, address
     _addrSC, uint _T, bytes32 _hashM, bytes memory _fA, bytes memory _fB)
    internal pure {

bytes32 hashData = keccak256(abi.encodePacked(_idAB,_hashM));

require(validID(_idAB,_addrA,_addrB,_addrSC,_T,_hashM), ‘‘Invalid
    identifier’’);
require(validSign(hashData,_fA,_addrA),‘‘A Invalid evidence’’);
require(validSign(hashData,_fB,_addrB),‘‘B Invalid evidence’’);
}
```

When the *resolution* function is completed, the smart contract emits an event to communicate the execution's outcome. Consequently, both A and B gain real-time insight into the contract signing status. To further facilitate this process, we implement a *query* function (see Listing 4) that enables both A and B to check the contract's status at any time. When the status is `finalized`, the smart contract returns the evidence f_A, f_B, and ACK_A provided by A. If the contract is canceled, it returns the evidence f_A and f_B provided by B. When the provided identifier has no associated information, the smart contract employs the revert mechanism (In Solidity, the revert statement [84] is used to stop the execution of a smart contract. This mechanism is a crucial part of writing secure and robust smart contracts, as it helps prevent unexpected or erroneous behavior that could otherwise lead to unwanted state changes on the blockchain) to address this scenario and notifies the calling signer accordingly. Importantly, the *query* function operates without altering the contract signing status, eliminating the need for any transactions to be added to the blockchain and, consequently, the associated economic cost (see Section 8).

7. Security Review

Next, we show that our protocol satisfies the desired requirements. The fairness and hard-timeliness requirements are presented together to avoid duplicating explanations.

Effectiveness. There is no TTP involved in any of the subprotocols. Therefore, the protocol meets the effectiveness requirement.

Non-repudiation. The evidence available to a signatory is signed by the other signatory, who cannot deny his/her involvement. Therefore, the protocol meets the non-repudiation requirement.

Confidentiality. The content of the contract is known to only A and B, and the smart contract functions require only the hash of the contract. Therefore, the protocol meets the confidentiality requirement.

Bc-optimistic. Under normal conditions, only the *exchange* subprotocol is executed, and no blockchain functions are executed. Therefore, the protocol satisfies the bc-optimistic requirement.

Fairness and hard-timeliness. We analyzed the states in which signatories can be found and the actions that can be taken in each state. We only considered cases in which at least one of the signatories is honest (if both are dishonest, what happens is inconsequential).

State 1: Nothing has been sent. Neither signatory has evidence proving the contract is signed, and both can "stop" the execution of the protocol (in fact, the execution has not started).

State 2: A has sent f_A. No signatory has evidence proving the contract is signed. Both can stop the execution of the protocol without loss of fairness. If B requests cancellation of the smart contract, this would be irrelevant.

State 3: B has sent f_B. No signatory has evidence proving the contract is signed or canceled; however, they can obtain evidence with the help of the blockchain. The following situations may occur:

1. A stops execution of the exchange; If B is honest (and smart) he will cancel the exchange (when he wants). None of the signatories has evidence proving the contract is signed;
2. A finishes using the smart contract (when she wants), and the smart contract records the evidence proving this fact. If B attempts to cancel the exchange, the smart contract will provide him with ACK_A. Both have evidence proving the contract is signed;
3. B cancels, and A stops. The result is analogous to situation 1 explained above;
4. B cancels, and A attempts to finish. Once the exchange is canceled, the smart contract only provides evidence of cancellation. No one will have evidence that the contract is signed;
5. B cancels, and A follows with execution. This case is analogous to the case that will be discussed below ("State 4—situation 2").

State 4: A has sent ACK_A. B already has evidence of the signing of the contract and A does not. The following situations may occur:

1. At the moment A wishes, she can request finalization of the smart contract, and if B has not canceled, the smart contract will update the status of the contract signature and record the associated evidence; therefore, she will have evidence of the signing of the contract;
2. If B is dishonest, although he already has f_A-ACK_A, he can issue the order to cancel the exchange. If this execution is prior to A's $finish$ request, the smart contract will cancel the transaction. If A now requests the execution of $finish$, the smart contract will send evidence of cancellation. If B attempts to prove that the contract is signed (providing f_A-ACK_A), A will be able to show the cancellation evidence recorded by the smart contract, which will prove that B was dishonest (when he already had

f_A-ACK_A, he executed the cancellation). This situation is what we had left pending in "State 3—situation 5)".

State 5: B has sent ACK_B. Both have evidence of the signing of the contract (the protocol has finished). The following situations may occur:

1. A could request finalization of the contract signing, but this is an unnecessary and senseless situation, since the smart contract would only record information that both contracting parties already possess. All parties can prove the signing of the contract without the involvement of the smart contract;
2. B can request cancellation of the contract signing. However, if he attempts to use the data recorded by the smart contract (the contract signing is canceled), A could show f_B and ACK_B, which would prove that B was dishonest.

The previous explanations enable us to confirm that the protocol satisfies the fairness requirement (no signatory is at a disadvantage) and the timeliness requirement (both can finish the execution of the protocol at the moment they want).

8. Cost Assessment

Paying for executing smart contracts is necessary to cover the costs of using computing resources on blockchain networks such as Ethereum [85,86]. This prevents spam (by making it costly for attackers to flood the network with unnecessary transactions), ensures fair resource allocation, supports network sustainability, and incentivizes network participants to authenticate and secure transactions. This economic model helps maintain the efficiency and security of the blockchain ecosystem.

To evaluate the costs associated with our solution, we deployed our smart contract on the Hardhat Network, a dedicated local Ethereum network node suitable for development purposes [87]. This approach enabled us to deploy, test, and debug smart contract code in a local environment, avoiding the costs associated with a real public blockchain.

8.1. Gas Cost

The complexity of the operations involved in executing smart contract functions on the EVM imposes specific economic costs, quantified in gas units, as specified in [78]. For example, the amount of gas required to create a contract is fixed to 32,000 gas units, even before any contract functions have been performed. Several tools exist for estimating gas consumption for individual smart contract functions. We chose the Ethereum gas reporting plugin [88], due to its adaptability, allowing it to seamlessly integrate with multiple development frameworks, including Hardhat.

To assess the cost of our smart contract, we developed a script in the JavaScript language. This script allowed us to deploy the contract using Hardhat and execute the resolution function. After completing the test, the Ethereum gas reporting plugin provided the cost of each operation measured in gas units. In Table 2, we present the cost measured (as explained before), in gas units, associated with the main function of our solution, *resolution*, as well as the deployment cost of the smart contract itself. Additionally, we assessed the cost of the resolution function by considering whether the contract signature had been requested to be finalized or canceled. As shown in the table, deploying the smart contract on the blockchain was the most expensive operation (1,658,625 gas units). Its cost primarily depends on the fixed costs associated with contract creation and the size of the contract's bytecode [78]. However, a smart contract can be deployed in advance and used multiple times to oversee the resolution of different contract signatures.

The execution of the resolution function depends on the type of request made. In the case of requesting finalization of the contract signature, the *SC* must perform the validations specified in Section 5.2.2, primarily involving cryptographic hashing and verifying and storing the signatures of three pieces of signed evidence (see Sections 6.1 and 6.3). The execution of these operations and the storage of evidence entailed a cost of 343,282 gas units (see Table 2). In the case of requesting cancellation of the contract signature, the *SC* must

perform validations similar to those for finalization; but in this case, it only needs to verify and store two pieces of signed evidence (see Sections 6.1 and 6.3). The execution of these operations and the storage of evidence entailed a cost of 247,847 gas units (see Table 2). This made the cost of canceling the contract signature approximately 38% lower than the cost of finalizing it. The *query* function does not alter the blockchain's state; therefore, it does not incur any gas consumption.

Table 2. Cost in gas units of the deployment of the *SC* and the execution of the *resolution* function to finish and cancel the contract signing.

	Deploy	*Resolution*	
		Finish	**Cancel**
Gas units	1,658,625	343,282	247,847

8.2. Cost in Fiat

Measurement in gas units serves as a valuable indicator of the complexity of the operations executed and provides a consistent metric to compare different solutions, because this metric remains unaffected by variations in the price of the cryptocurrency associated with the blockchain on which the *SC* is executed [78]. However, the economic costs required to perform the tasks involved in our solution can fluctuate on a daily basis due to changes in the market value of cryptocurrencies [60,89]. Therefore, the final cost of implementing a function was calculated by multiplying the gas units required and the current gas price at the time of execution.

Gas price refers to the amount of Wei (Wei refers to the smallest denomination of Ether (ETH), the currency used on the Ethereum network (1 ETH = 10^{18} Wei).) that a user is willing to spend per unit of gas. To account for daily gas price fluctuations, we analyzed historical data on gas prices over time. For this analysis, we relied on the dataset provided by [90], which records the average daily gas price in Wei for the Ethereum blockchain. Leveraging the Ether–USD exchange rate (per day [91]), we determined the total cost of deploying the smart contract and running the resolution function when finish or cancel is requested by signatories. Therefore, the final cost in fiat of deploying and executing the resolution function was obtained by multiplying the three parameters mentioned above: the gas units (obtained in Section 8.1), the average gas price per day, and the daily Ether–USD exchange rate.

Figure 5 illustrates the average daily costs (in USD) that could be incurred by signatories during 2023 (1 January to 30 September) when using the Ethereum network to perform smart contract functions. Close inspection of the figure indicates that some significant cost peaks coincide with network congestion events (https://markets.businessinsider.com/currencies/eth-usd, accessed on 1 October 2023), leading to escalating gas demand and, therefore, higher gas prices. Beyond these peaks, the remaining data show a relatively stable trend (as shown in Figure 5 and Table 3). However, the overall price can be considered significant considering both the quantity and frequency of contract signatures requiring a resolution. Note that this function is exclusively called under specific circumstances (see Section 5.2.2), such as when A has not received acknowledgment (ACK_B) from B and wishes to complete the contract signature or when B wants to cancel the contract signature.

In recent years, new EVM-based blockchains have emerged to address some of the challenges faced by Ethereum, particularly its scalability limitations and the resulting cost implications [92]. Two of these blockchains are Binance Smart Chain (BSC) [93] and Polygon [94]. BSC is a layer-one blockchain ("A layer-1 network is another name for a base blockchain. BNB Smart Chain (BNB), Ethereum (ETH), and Bitcoin (BTC) are all layer-1 protocols" Source: https://academy.binance.com, accessed on 1 October 2023) that supports smart contracts and is designed to operate independently as a standalone blockchain, while remaining compatible with the Ethereum ecosystem. Polygon is a layer-two blockchain ("Layer-2 solutions build on layer 1 and rely on it to finalise its transactions" Source:

https://academy.binance.com, accessed on 1 October 2023) scaling solution operating on top of Ethereum, enhancing its scalability and functionality. Next, we examine the cost of implementing our solution on BSC and Polygon, considering them as representative examples of EVM-based blockchains.

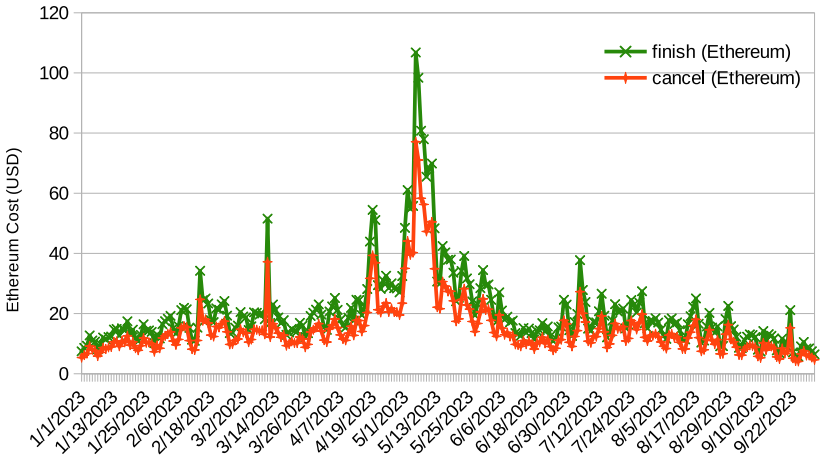

Figure 5. *Resolution* function: estimated average spend (in USD) for 2023, spanning from 1 January to 30 September, taking into account average gas prices observed on the Ethereum network.

Figures 6 and 7 depict the average costs (in USD and obtained following the same procedure as for Ethereum) for executing the *resolution* function, considering both finalization and cancellation requests and during the same time period as analyzed for Ethereum. Notably, these prices exhibited a substantial reduction when compared to Ethereum, with all falling below USD 1.0. This held true even when considering the maximum price rather than the average, as detailed in Table 3; the highest price on Ethereum reached USD 106.80, compared to USD 1 on BSC and USD 0.24 on Polygon.

When comparing BSC and Polygon, the latter stood out with the most cost-effective rates. For the resolution execution (finish and cancel), the cost of Polygon was under USD 0.08, while the cost of BSC was approximately USD 0.6. Even when considering the maximum price, the cost of BSC was approximately USD 1.0, while the cost of Polygon remained below USD 0.25.

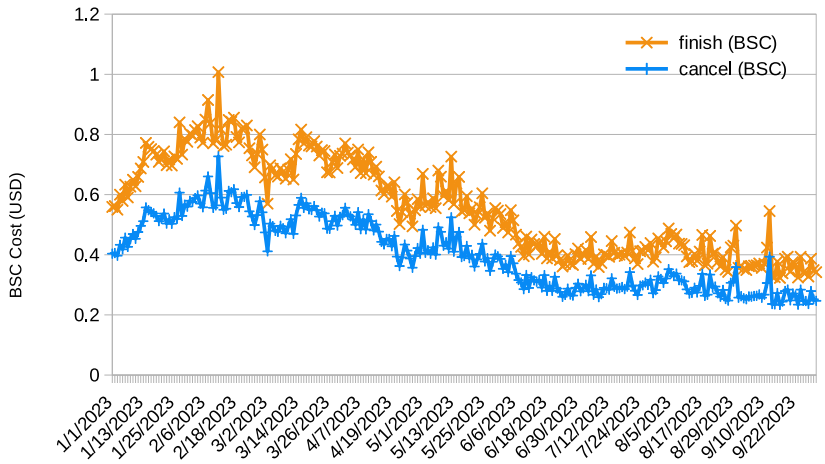

Figure 6. *Resolution* function: estimated average spend (in USD) for 2023, spanning from 1 January to 30 September, taking into account average gas prices observed on the BSC network.

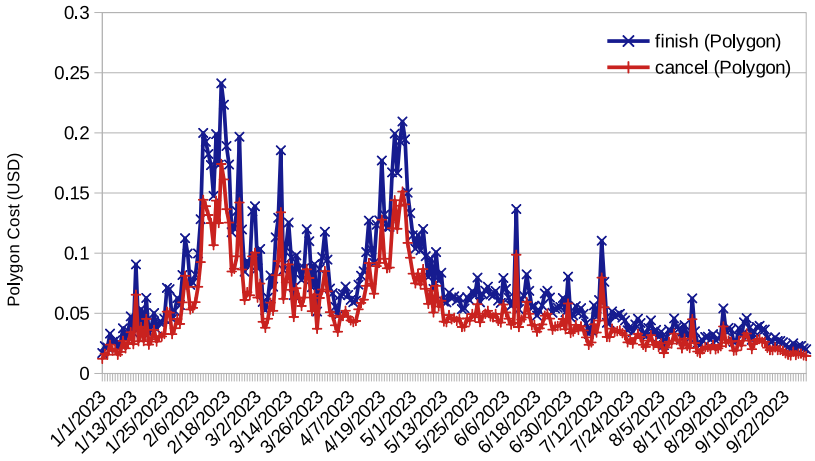

Figure 7. *Resolution* function: estimated average spend (in USD) for 2023, spanning from 1 January to 30 September, taking into account average gas prices observed on the Polygon network.

Table 3. Maximum, minimum, and average (and standard deviation) of the price (in USD) required by each function executed in 2023.

	Ethereum			**BSC**			**Polygon**		
	Deploy	***Finish***	***Cancel***	**Deploy**	***Finish***	***Cancel***	**Deploy**	***Finish***	***Cancel***
Avg.	99.78	20.65	14.91	2.696	0.558	0.403	0.34	0.07	0.05
Max.	516.03	106.80	77.11	4.866	1.007	0.727	1.17	0.24	0.17
Min.	27.25	5.64	4.07	1.561	0.323	0.233	0.08	0.02	0.01
(std.)	65.85	13.63	9.84	0.777	0.161	0.116	0.22	0.04	0.03

9. Discussion

Our proposal meets the *hard-timeliness* requirement and therefore it is better than proposals [63,67,70] that do not meet any *timeliness* requirement. Regarding the proposals [59–62,65,66,68,69,71,72] that meet the *soft-timeliness* requirement (establishing one or more deadlines), we also consider that our proposal is an enhancement, specially when public blockchains are used, since in these blockchains the moment of execution of the functions cannot be guaranteed.

From the point of view of compliance with the *hard-timeliness* requirement, our proposal is comparable with one of the proposals in [72]. But this proposal requires the execution of functions of the blockchain, regardless of the behavior of the signatories. Therefore, our proposal represents an improvement from a cost point of view, as regards the use of the blockchain. Recall that in our proposal blockchain functions are only executed in case of exception and not in each protocol run.

Finally, our proposal achieves the fairness requirement without the signatories having to make any a priori financial deposit. We think that this achievement represents an improvement over [59,61,62,64–66,69,73], since this "restriction" could be a serious obstacle in contract signing scenarios where one or both signatories may be reluctant to have money blocked (without knowing if the contract will finally be signed).

10. Conclusions

The first conclusion of this work is that the concept of timeliness has not been clearly defined since its first use. It would have been desirable to use different names for different requirements (as we have done in this work). In any case, authors of contract signing protocols must clearly specify what requirements they want their proposal to satisfy. This is especially important when attempting to compare different proposals.

In this work, we have presented the first proposal for contract signing based on blockchain that meets the following requirements: fairness, hard-timeliness, non-repudiation,

confidentiality, and bc-optimism. This approach ensures that a contract's content remains confidential and is never exposed on the blockchain. Moreover, any signatory can finalize the contract signing, without any disadvantages or the imposition of strict time limits. Since blockchain transactions involve costs, signatories only resort to it in exceptional circumstances. As indicated by our cost assessment, the selection of an appropriate blockchain solution helps keep costs minimal. Furthermore, this analysis ensures that the proposed solution aligns with the financial requirements across various scenarios, making this a valuable guide for its application in diverse financial contexts.

As part of our future work, we plan to include an optional abuse-freeness requirement, which has not been addressed in this paper. Additionally, we aim to investigate the applicability of our protocol in different blockchain environments, particularly focusing on its feasibility within private and consortium blockchains.

Author Contributions: All authors have contributed to this work equally. All authors have read and agreed to the published version of the manuscript.

Funding: This work was conducted as part of projects BLOBSEC (PID2021-122394OB-I00, funded by MCIN/AEI/10.13039/501100011033, Spain, and by ERDF "A way of making Europe"), and GERMINAL (TED2021-131624B-I00, funded by MCIN/AEI/10.13039/501100011033, Spain, and by European Union "NextGenerationEU"/PRTR).

Data Availability Statement: All data generated or analysed during this study are included in this published article.

Conflicts of Interest: The authors declare no conflict of interest in this article.

References

1. Wang, Y.; Su, Z.; Zhang, N.; Chen, J.; Sun, X.; Ye, Z.; Zhou, Z. SPDS: A Secure and Auditable Private Data Sharing Scheme for Smart Grid Based on Blockchain. *IEEE Trans. Ind. Inform.* **2021**, *17*, 7688–7699. [CrossRef]
2. Hinarejos, M.F.; Ferrer-Gomila, J.L.; Barceló, A.J. A Secure Solution for a Blockchain-Based Consortium Promotional Scheme. *IEEE Access* **2022**, *10*, 119676–119691. [CrossRef]
3. Arbabi, M.S.; Lal, C.; Veeraragavan, N.R.; Marijan, D.; Nygård, J.F.; Vitenberg, R. A Survey on Blockchain for Healthcare: Challenges, Benefits, and Future Directions. *IEEE Commun. Surv. Tutor.* **2023**, *25*, 386–424. [CrossRef]
4. Li, Z.; Wang, W.; Wu, Q.; Wang, X. Multi-Operator Dynamic Spectrum Sharing for Wireless Communications: A Consortium Blockchain Enabled Framework. *IEEE Trans. Cogn. Commun. Netw.* **2023**, *9*, 3–15. [CrossRef]
5. Azzahra, Z.F.; Nugraha, I.G.B.B. Service-Level Agreement Management with Blockchain-Based Smart Contract to Improve the Quality of IT Service Management. In *Proceedings of the 2023 12th International Conference on Software and Computer Applications*; Association for Computing Machinery: New York, NY, USA, 2023; ICSCA '23; pp. 260–266. [CrossRef]
6. Tan, W.; Zhu, H.; Tan, J.; Zhao, Y.; Xu, L.D.; Guo, K. A novel service level agreement model using blockchain and smart contract for cloud manufacturing in industry 4.0. *Enterp. Inf. Syst.* **2022**, *16*, 1939426. [CrossRef]
7. Asokan, N.; Shoup, V.; Waidner, M. Asynchronous protocols for optimistic fair exchange. In Proceedings of the 1998 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 6 May 1998; pp. 86–99.
8. Kremer, S.; Raskin, J. A Game-Based Verification of Non-repudiation and Fair Exchange Protocols. In Proceedings of the International Conference on Concurrency Theory—CONCUR, Aalborg, Denmark, 20–25 August 2001; pp. 551–565.
9. Drielsma, P.H.; Mödersheim, S. The ASW protocol revisited: A unified view. *Electron. Notes Theor. Comput. Sci.* **2005**, *125*, 145–161. [CrossRef]
10. Chadha, R.; Kremer, S.; Scedrov, A. Formal analysis of multiparty contract signing. *J. Autom. Reason.* **2006**, *36*, 39–83. [CrossRef]
11. Mauw, S.; Radomirovic, S.; Dashti, M.T. Minimal message complexity of asynchronous multi-party contract signing. In Proceedings of the 2009 22nd IEEE Computer Security Foundations Symposium, Port Jefferson, NY, USA, 8–10 July 2009; pp. 13–25.
12. Piva, F.R.; Monteiro, J.R.; Dahab, R. Regarding timeliness in the context of fair exchange. In Proceedings of the International Conference on Network and Service Security—N2S, Paris, France, 24–26 June 2009; pp. 1–6.
13. Chen, M.; Wu, K.; Xu, J.; He, P. A new method for formalizing optimistic fair exchange protocols. In Proceedings of the Information and Communications Security: 12th International Conference, ICICS 2010, Barcelona, Spain, 15–17 December 2010; Proceedings 12; Springer: Berlin/Heidelberg, Germany, 2010; pp. 251–265.
14. Küpçü, A.; Lysyanskaya, A. Optimistic fair exchange with multiple arbiters. In Proceedings of the Computer Security–ESORICS 2010: 15th European Symposium on Research in Computer Security, Athens, Greece, 20–22 September 2010; Proceedings 15; Springer: Berlin/Heidelberg, Germany, 2010; pp. 488–507.

15. Abraham, A.; Ewards, V.; Mathew, H.M. A survey on optimistic fair digital signature exchange protocols. *Int. J. Comput. Sci. Eng.* **2011**, *3*, 821–825.
16. Asokan, N.; Schunter, 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.
17. Asokan, N.; Shoup, V.; Waidner, M. Optimistic fair exchange of digital signatures. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 1998; pp. 591–606.
18. Asokan, N.; Shoup, V.; Waidner, M. Optimistic fair exchange of digital signatures. *IEEE J. Sel. Areas Commun.* **2000**, *18*, 593–610. [CrossRef]
19. Norman, G.; Shmatikov, V. Analysis of probabilistic contract signing. In *Formal Aspects of Security*; Springer: Berlin/Heidelberg, Germany, 2002; pp. 81–96.
20. Ferrer-Gomila, J.L.; Martínez-Nadal, A.L.; Payeras-Capellà, M.; Huguet-Rotger, L. A juridical validation of a contract signing protocol. In Proceedings of the E-Commerce and Web Technologies: Third International Conference, EC-Web 2002, Aix-en-Provence, France, 2–6 September 2002; Proceedings 3; Springer: Berlin/Heidelberg, Germany, 2002; pp. 343–352.
21. Lee, B.; Kim, K. Fair exchange of digital signatures using conditional signature. In Proceedings of the Symposium on Cryptography and Information Security, Shirahama, Japan, 29 January–1 February 2002; pp. 179–184.
22. Norman, G.; Shmatikov, V. Analysis of probabilistic contract signing. *J. Comput. Secur.* **2006**, *14*, 561–589. [CrossRef]
23. Zhou, J.; Deng, R.; Bao, F. Some Remarks on a Fair Exchange Protocol. In *Proceedings of the Public Key Cryptography*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2000; Volume 1751, pp. 46–57.
24. 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*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 200–208.
25. Onieva, J.A.; Zhou, J.; Lopez, J. Attacking an asynchronous multi-party contract signing protocol. In Proceedings of the Progress in Cryptology-INDOCRYPT 2005: 6th International Conference on Cryptology in India, Bangalore, India, 10–12 December 2005; Proceedings 6; Springer: Berlin/Heidelberg, Germany, 2005; pp. 311–321.
26. Zhou, J.; Onieva, J.A.; Lopez, J. A synchronous multi-party contract signing protocol improving lower bound of steps. In Proceedings of the Security and Privacy in Dynamic Environments: Proceedings of the IFIP TC-11 21st International Information Security Conference (SEC 2006), Karlstad, Sweden, 22–24 May 2006; Springer: Boston, MA, USA, 2006; Volume 201, pp. 221–232.
27. Zhang, Y.; Zhang, C.; Pang, J.; Mauw, S. Game-based verification of multi-party contract signing protocols. In Proceedings of the Formal Aspects in Security and Trust: 6th International Workshop, FAST 2009, Eindhoven, The Netherlands, 5–6 November 2009; Revised Selected Papers 6; Springer: Berlin/Heidelberg, Germany, 2010; pp. 186–200.
28. Sun, Y.; Gu, L.; Qing, S.; Zheng, S.; Sun, B.; Yang, Y.; Sun, Y. Timeliness optimistic fair exchange protocol based on key-exposure-free chameleon hashing scheme. In Proceedings of the 2010 The 12th International Conference on Advanced Communication Technology (ICACT), Gangwon, Republic of Korea, 7–10 February 2010; Volume 2; pp. 1560–1564.
29. Sun, Y.; Gu, L.; Qing, S.; Zheng, S.; Yang, Y.; Sun, Y. New optimistic fair exchange protocol based on short signature. In Proceedings of the 2010 Second International Conference on Communication Software and Networks, Singapore, 26–28 February 2010; pp. 99–104.
30. Gu, L.; Sun, Y. New Optimistic Fair Exchange Protocol Based on VE-RSA Signature. In Proceedings of the 2010 Second International Workshop on Education Technology and Computer Science, Wuhan, China, 6–7 March 2010; Volume 1; pp. 292–295.
31. Xiao, H.; Wang, L.; Wei, Y. A new fair electronic contract signing protocol. In *Proceedings of the Advances in Intelligent Networking and Collaborative Systems: The 11th International Conference on Intelligent Networking and Collaborative Systems (INCoS-2019)*; Springer: Cham, Switzerland, 2020; pp. 289–295.
32. Khill, I.; Kim, J.; Han, I.; Ryou, J. Multi-party fair exchange protocol using ring architecture model. *Comput. Secur.* **2001**, *20*, 422–439. [CrossRef]
33. Wang, H.; Guo, H.; Yin, J.; He, Q.; Lin, M.; Zhang, J. Abuse-free item exchange. In Proceedings of the Computational Science and Its Applications–ICCSA 2005: International Conference, Singapore, 9–12 May 2005; Proceedings, Part IV 5; Springer: Berlin/Heidelberg, Germany, 2005; pp. 1028–1035.
34. Wang, G. An abuse-free fair contract signing protocol based on the RSA signature. In Proceedings of the 14th International Conference on World Wide Web, Chiba, Japan, 10–14 May 2005; pp. 412–421.
35. Islam, S.; Zaid, M.A. Probabilistic Analysis and Verification of the ASW Protocol using PRISM. *Int. J. Netw. Secur.* **2008**, *7*, 388–396.
36. Islam, S.; Zaid, M.A. Probabilistic analysis of the ASW protocol using PRISM. In Proceedings of the IEEE SoutheastCon 2008, Huntsville, AL, USA, 3–6 April 2008; pp. 159–164.
37. Wang, G. An abuse-free fair contract-signing protocol based on the RSA signature. *IEEE Trans. Inf. Forensics Secur.* **2009**, *5*, 158–168. [CrossRef]
38. Torabi Dashti, M. Optimistic fair exchange using trusted devices. In *Proceedings of the Symposium on Self-Stabilizing Systems*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 711–725.
39. Heidarvand, S.; Villar, J.L. A fair and abuse-free contract signing protocol from boneh-boyen signature. In Proceedings of the Public Key Infrastructures, Services and Applications: 7th European Workshop, EuroPKI 2010, Athens, Greece, 23–24 September 2010; Revised Selected Papers 7; Springer: Berlin/Heidelberg, Germany, 2011; pp. 125–140.
40. Küpçü, A.; Lysyanskaya, A. Usable optimistic fair exchange. *Comput. Netw.* **2012**, *56*, 50–63. [CrossRef]

41. Draper-Gil, G.; Ferrer-Gomila, J.L.; Hinarejos, M.F.; Zhou, J. An asynchronous optimistic protocol for atomic multi-two-party contract signing. *Comput. J.* **2013**, *56*, 1258–1267. [CrossRef]
42. Draper-Gil, G.; Zhou, J.; Ferrer-Gomila, J.L.; Hinarejos, M.F. An optimistic fair exchange protocol with active intermediaries. *Int. J. Inf. Secur.* **2013**, *12*, 299–318. [CrossRef]
43. Chatterjee, K.; Raman, V. Assume-guarantee synthesis for digital contract signing. *Form. Asp. Comput.* **2014**, *26*, 825–859. [CrossRef]
44. Draper-Gil, G.; Ferrer-Gomila, J.L.; Hinarejos, M.F.; Zhou, J. On the efficiency of multi-party contract signing protocols. In Proceedings of the Information Security: 18th International Conference, ISC 2015, Trondheim, Norway, 9–11 September 2015; Proceedings 18; Springer: Berlin/Heidelberg, Germany, 2015; pp. 227–243.
45. Xu, G.; Zhang, Y.; Jiao, L.; Panaousis, E.; Liang, K.; Wang, H.; Li, X. DT-CP: A double-TTPs-based contract-signing protocol with lower computational cost. *IEEE Access* **2019**, *7*, 174740–174749. [CrossRef]
46. Kremer, S.; Raskin, J.F. Game analysis of abuse-free contract signing. In Proceedings of the 15th IEEE Computer Security Foundations Workshop, CSFW-15, Cape Breton, NS, Canada, 24–26 June 2002; pp. 206–220.
47. Imamoto, K.; Zhou, J.; Sakurai, K. An evenhanded certified email system for contract signing. In Proceedings of the Information and Communications Security: 7th International Conference, ICICS 2005, Beijing, China, 10–13 December 2005; Proceedings 7; Springer: Berlin/Heidelberg, Germany, 2005; pp. 1–13.
48. Imamoto, K.; Zhou, J.; Sakurai, K. Achieving evenhandedness in certified email system for contract signing. *Int. J. Inf. Secur.* **2008**, *7*, 383–394. [CrossRef]
49. Chadha, R.; Mitchell, J.C.; Scedrov, A.; Shmatikov, V. Contract signing, optimism, and advantage. In *Proceedings of the International Conference on Concurrency Theory*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 366–382.
50. Chadha, R.; Mitchell, J.C.; Scedrov, A.; Shmatikov, V. Contract signing, optimism, and advantage. *J. Log. Algebr. Program.* **2005**, *64*, 189–218. [CrossRef]
51. Mukhamedov, A.; Ryan, M.D. Resolve-impossibility for a contract-signing protocol. In Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW'06), Venice, Italy, 5–7 July 2006; pp. 176–182.
52. Mukhamedov, A.; Ryan, M. Improved multi-party contract signing. In Proceedings of the Financial Cryptography and Data Security: 11th International Conference, FC 2007, and 1st International Workshop on Usable Security, USEC 2007, Scarborough, Trinidad and Tobago, 12–16 February 2007; Revised Selected Papers 11; Springer: Berlin/Heidelberg, Germany, 2007; pp. 179–191.
53. Wang, X. Modeling and Analysis of Multi-party Fair Exchange Protocols. In Proceedings of the 2007 International Conference on Wireless Communications, Networking and Mobile Computing, Shanghai, China, 21–25 September 2007; pp. 2246–2250.
54. Mukhamedov, A.; Ryan, M.D. Fair multi-party contract signing using private contract signatures. *Inf. Comput.* **2008**, *206*, 272–290. [CrossRef]
55. 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.
56. Zhang, Y.; Zhang, C.; Pang, J.; Mauw, S. Game-based verification of contract signing protocols with minimal messages. *Innov. Syst. Softw. Eng.* **2012**, *8*, 111–124. [CrossRef]
57. Mauw, S.; Radomirović, S. Generalizing multi-party contract signing. In Proceedings of the Principles of Security and Trust: 4th International Conference, POST 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, 11–18 April 2015; Proceedings 4; Springer: Berlin/Heidelberg, Germany, 2015; pp. 156–175.
58. Orzan, S.; de Vink, E. Multiparty contract signing over a reliable network. *Electron. Notes Theor. Comput. Sci.* **2006**, *157*, 27–41. [CrossRef]
59. Tian, H.; He, J.; Fu, L. Contract coin: Toward practical contract signing on blockchain. In Proceedings of the Information Security Practice and Experience: 13th International Conference, ISPEC 2017, Melbourne, VIC, Australia, 13–15 December 2017; Proceedings 13; Springer: Berlin/Heidelberg, Germany, 2017; pp. 43–61.
60. Ferrer-Gomila, J.L.; Hinarejos, M.F.; Isern-Deya, A.P. A fair contract signing protocol with blockchain support. *Electron. Commer. Res. Appl.* **2019**, *36*, 100869. [CrossRef]
61. Mut-Puigserver, M.; Payeras-Capellà, M.M.; Cabot-Nadal, M.À. 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.
62. Zhang, L.; Zhang, H.; Yu, J.; Xian, H. Blockchain-based two-party fair contract signing scheme. *Inf. Sci.* **2020**, *535*, 142–155. [CrossRef]
63. Yang, K.; Wu, Y.; Chen, Y. A Blockchain-based Scalable Electronic Contract Signing System. In Proceedings of the 2022 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics), Espoo, Finland, 22–25 August 2022; pp. 343–348.
64. Wang, G.; Yu, Y.; Song, Z.; Fu, T. Fair Contract Signing Model Based on Blockchain and VES Algorithm. In Proceedings of the 2023 5th International Conference on Communications, Information System and Computer Engineering (CISCE), Guangzhou, China, 14–16 April 2023; pp. 476–480.

65. Huang, H.; Li, K.C.; Chen, X. A fair three-party contract singing protocol based on blockchain. In Proceedings of the Cyberspace Safety and Security: 9th International Symposium, CSS 2017, Xi'an China, 23–25 October 2017; Proceedings; Springer: Berlin/Heidelberg, Germany, 2017; pp. 72–85.
66. 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]
67. Zhang, Q.; Gao, J.; Qin, Q.; Wang, C.; Yin, K. FutureText: A blockchain-based contract signing prototype with security and convenience. In Proceedings of the 3rd ACM International Symposium on Blockchain and Secure Critical Infrastructure, Hong Kong, China, 7–11 June 2021; pp. 77–83.
68. Ferrer-Gomila, J.L.; Hinarejos, M.F. A multi-party contract signing solution based on blockchain. *Electronics* **2021**, *10*, 1457. [CrossRef]
69. Payeras-Capellà, M.M.; Mut-Puigserver, M.; Cabot-Nadal, M.À.; Huguet-Rotger, L. Blockchain-based confidential multiparty contract signing protocol without TTP using elliptic curve cryptography. *Comput. J.* **2022**, *65*, 2755–2768. [CrossRef]
70. Zhang, T.; Wang, Y.; Ding, Y.; Wu, Q.; Liang, H.; Wang, H. Multi-party electronic contract signing protocol based on blockchain. *IEICE Trans. Inf. Syst.* **2022**, *105*, 264–271. [CrossRef]
71. Zhang, T.; Wang, Y.; Ding, Y.; Jiang, X.; Liang, H.; Wang, H. Privacy-preserving blockchain-based contract signing with multi-party supervision. *Trans. Emerging Tel. Tech.* **2022**, ett.4710. [CrossRef]
72. Hinarejos, M.F.; Ferrer-Gomila, J.L.; Isern-Deyà, A.P. Enforcing Fairness with Blockchain Support: Proposals for Multi-two-party Contract Signing. *IEEE Access* **2023**, *11*, 67893–67911. [CrossRef]
73. Wang, D.; Li, Q.; Li, F.; Zhang, Q.; Xu, B. Privacy-awareness fair contract signing protocol based on blockchain. In Proceedings of the Cyberspace Safety and Security: 11th International Symposium, CSS 2019, Guangzhou, China, 1–3 December 2019; Proceedings, Part I 11; Springer: Cham, Switzerland, 2019; pp. 274–278.
74. Johar, S.; Ahmad, N.; Asher, W.; Cruickshank, H.; Durrani, A. Research and applied perspective to blockchain technology: A comprehensive survey. *Appl. Sci.* **2021**, *11*, 6252. [CrossRef]
75. Saini, K.; Roy, A.; Chelliah, P.R.; Patel, T. Blockchain 2.O: A Smart Contract. In Proceedings of the 2021 International Conference on Computational Performance Evaluation (ComPE), Shillong, India, 1–3 December 2021; pp. 524–528. [CrossRef]
76. Li, X.; Jiang, P.; Chen, T.; Luo, X.; Wen, Q. A survey on the security of blockchain systems. *Future Gener. Comput. Syst.* **2020**, *107*, 841–853. [CrossRef]
77. Li, C.; Palanisamy, B.; Xu, R. Scalable and Privacy-Preserving Design of On/Off-Chain Smart Contracts. In Proceedings of the 2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW), Macao, China, 8–12 April 2019; pp. 7–12. [CrossRef]
78. Wood, G. Ethereum: A Secure Decentralised Generalised Transaction Ledger; EIP-150 REVISION (759dccd) ; 2017. Available online: https://ethereum.github.io/yellowpaper/paper.pdf (accessed on 16 October 2023).
79. Ethereum.org. The Community-Run Technology Powering the Cryptocurrency Ether (ETH) and Thousands of Decentralized Applications. Available online: https://ethereum.org/ (accessed on 1 August 2023).
80. Ethereum.org. Smart Contracts Languages. Available online: https://ethereum.org/en/developers/docs/smart-contracts/languages/ (accessed on 1 October 2023).
81. Marchesi, L.; Marchesi, M.; Destefanis, G.; Barabino, G.; Tigano, D. Design Patterns for Gas Optimization in Ethereum. In Proceedings of the 2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE), London, ON, Canada, 18 February 2020; pp. 9–15. [CrossRef]
82. Kremer, S.; Markowitch, O.; Zhou, J. An intensive survey of fair non-repudiation protocols. *Comput. Commun.* **2002**, *25*, 1606–1621. [CrossRef]
83. OpenZeppelin. The Standard for Secure Blockchain Applications. Available online: https://www.openzeppelin.com/ (accessed on 1 August 2023).
84. Ethereum. Solidity Language: Expressions and Control Structures. Available online: https://docs.soliditylang.org/en/v0.4.24/control-structures.html (accessed on 1 October 2023).
85. Baird, K.; Jeong, S.; Kim, Y.; Burgstaller, B.; Scholz, B. The Economics of Smart Contracts. *arXiv* **2019**, arXiv:1910.11143
86. Koutmos, D. Network Activity and Ethereum Gas Prices. *J. Risk Financ. Manag.* **2023**, *16*, 431. [CrossRef]
87. Nomic Foundation. Ethereum Development Environment for Professionals. Available online: https://hardhat.org/ (accessed on 1 October 2023).
88. Ether Gas Reporter. A Mocha Reporter for Ethereum Test Suites. Available online: https://github.com/cgewecke/eth-gas-reporter (accessed on 1 August 2023).
89. Ammer, M.A.; Aldhyani, T.H.H. Deep Learning Algorithm to Predict Cryptocurrency Fluctuation Prices: Increasing Investment Awareness. *Electronics* **2022**, *11*, 2349. [CrossRef]
90. Etherscan. Block Explorer and Analytics Platform for Ethereum. Available online: https://etherscan.io/ (accessed on 1 August 2023).
91. CoinMarketCap. Price-Tracking Website for Cryptoassets. Available online: https://coinmarketcap.com/ (accessed on 1 October 2023).
92. Gangwal, A.; Gangavalli, H.R.; Thirupathi, A. A survey of Layer-two blockchain protocols. *J. Netw. Comput. Appl.* **2023**, *209*, 103539. [CrossRef]

93. Bnbchain.org. BNB Smart Chain White Paper. Available online: https://github.com/bnb-chain/whitepaper (accessed on 1 February 2023).
94. Polygon Technology. Ethereum's Internet of Blockchains. Available online: https://polygon.technology/ (accessed on 1 August 2023).