Hindawi
Security and Communication Networks
Volume 2020, Article ID 8828984, 16 pages
https://doi.org/10.1155/2020/8828984

Research Article

BNRDT: When Data Transmission Meets Blockchain

Hongjian Jin,[1] **Xingshu Chen**,[1,2] **Xiao Lan**,[2] **Hui Guo**,[3] **Hongxia Zhang**,[1] **and Qi Cao**[1]

[1]*College of Cybersecurity, Sichuan University, Chengdu 610065, China*
[2]*Cybersecurity Research Institute, Sichuan University, Chengdu 610065, China*
[3]*State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China*

Correspondence should be addressed to Xingshu Chen; chenxsh@scu.edu.cn and Xiao Lan; lanxiao@scu.edu.cn

Received 7 August 2020; Revised 12 October 2020; Accepted 26 October 2020; Published 18 November 2020

Academic Editor: Chenquan Gan

Data transmission exists in almost all the Internet-based applications, while few of them consider the property of nonrepudiation as part of data security. If a data transmission scheme is performed without the endorsement of a trusted third party (TTP) or a central server, it is easy to raise disputes while transmitting valuable data, especially digital goods, because a dishonest participant can deny the fact of particular data transmission instance. The above problem can be solved by signing and encrypting. However, digital signature schemes usually assume public key infrastructure (PKI), increasing the burden on certificate management and are not suitable for distributed networks without TTP such as blockchain. To solve the above problems, we propose two new schemes for nonrepudiation data transmission based on blockchain (we call it BNRDT): one for short message transmission and the other for large file transmission. In BNRDT schemes, nonrepudiation evidence of data transmission is generated and stored on the blockchain to satisfy both the properties of nonrepudiation (including nonrepudiation of origin and nonrepudiation of receipt) and data confidentiality. We implement and test the schemes on Hyperledger Fabric. The experimental results show that the proposed schemes can provide appealing performance.

1. Introduction

An overwhelming majority of Internet-based applications are inseparable from data transmission, may be short messages, videos, or even confidential government documents. In most cases (e.g., online chatting and video-on-demand service), data transmission processes rely on a trusted third party (TTP) or a central server, which acts as a data source (or a transmission relay station) and the security provider. With such a trusted platform, data security including confidentiality, integrity, authenticity, and even nonrepudiation when required are easily implemented. However, in some specific scenarios, such as P2P (Peer to Peer) digital goods trading, schemes are in lack of endorsements by trusted platforms; thus, nonrepudiation is no longer easy to achieve. Concretely, we consider that a digital goods seller needs to transmit the commodity over the Internet to an online buyer. Since the data transmission instance affects the parties' own interests, thus both the seller and the buyer want to ensure that the whole process can be undeniable, if they are honest. In other words, the buyer cannot deny having received the data so as to refuse to pay for it, and the seller cannot deny having sent it to the buyer so as to refuse to refund or be responsible for it if any problem arises after purchasing.

Nonrepudiation services [1] have been introduced for a long time to prove the nonrepudiation of user behaviour including the case of data transmission, which can effectively solve the above problem. By now, many nonrepudiation approaches have been proposed, but most of the existing work on nonrepudiation is still based on trusted third parties [2–5]. Such TTP-based implementations usually offer higher efficiency but may suffer from single-point failures. Moreover, a proper TTP is usually not available in distributed environments. Non-TTP-based nonrepudiation approaches are traditionally implemented by increasing interactions between users and gradually releasing secrets [6–8]. However, this kind of technique always leads to lower execution efficiency and cannot provide fairness for every participant equally. Beyond that, with the rapid development of

blockchain technology and smart contract, some blockchain-based nonrepudiation schemes have also been proposed [9–12], and they are trying to solve different nonrepudiation problems such as dispute resolution [9, 12], nonrepudiation of TLS sessions [10], or program delivery in industrial IoT (IIoT) environment [11], and almost all work based on blockchain removes the dependence on TTP, but none of them have been focused on data transmission scenario with supporting data confidentiality.

Summarily, although lot of work [2, 4, 5, 11] try to apply nonrepudiation to data transmission scenarios, but these work either only consider the simplest case that the data is transmitted with plaintext, or encrypt the data with a key, but the key will be published by the TTP, so anybody who obtains the ciphertext gets the data. While in most applications, we need to guarantee the confidentiality of the data when transmitting. So, they are not quite suitable for us to use.

Based on our observation, what function we want to realize in most cases is that the data can be transmitted in a secure manner, in which no TTP is required, fairness is provided equally to any participant, data is transmitted in ciphertext that only desired recipient can get it, and all behaviours throughout the process of data transmission cannot be denied. In order to realize our goal and make our work more adaptable, we give two data transmission schemes based on the characteristics of blockchain. In the first one, a short message can be transmitted securely and undeniably, which does not require TTP, provides fairness equally to any participant, and every behaviour is undeniable. While in the other scheme, which is an extended version of the first one, we can transmit large files undeniably with supporting the same properties.

Similar to the previous work in the research of nonrepudiation property, we consider the following nonrepudiation types [1, 4, 10, 13] in our data transmission schemes:

(i) Nonrepudiation of Origin (NRO): it provides evidence that the transmission data is originated from the specified sender

(ii) Nonrepudiation of Receipt (NRR): it provides evidence that the recipient has received the correct transmission data

Note that we focus on the malicious cases that the participants deny having done something. Therefore, the NRO evidence is generated for the recipient to prove the behaviour of the sender, and the NRR evidence is generated for the sender to prove the behaviour of the recipient. The NRO and NRR evidence should always appear in pairs to guarantee the fairness of generating nonrepudiation evidence.

The remainder of this paper is organized as follows. Section 2 introduces the related work and our contributions to better describe the improvement of our schemes over the existing ones. Section 3 introduces some preliminaries including the security model. Then, we give the detailed construction of the nonrepudiation data transmission schemes for short message transmission and large file transmission in Section 4 and Section 5, respectively. In Section 6, we perform the security analysis of the schemes and evaluate them in Section 7. Finally, we summarize this paper in Section 8.

2. Related Work and Our Contributions

2.1. Related Work

2.1.1. Traditional Nonrepudiation Approaches. Nonrepudiation protocols, being as a solution to the problem of accountability in the distributed communication environment, has gained a lot of attention. As mentioned above, most of existing nonrepudiation solutions rely on TTP to generate and judge the evidence [2–5]. Zhou and Gollman [4] proposed a fair nonrepudiation protocol in 1996, in which a TTP is introduced as a middleman to distribute the key used to encode the message. Once the recipient confirms the receipt of the ciphertext, the TTP will reveal the key (submitted by the sender) to the public to make sure that the recipient can recover the original message. In order to solve the efficiency problem caused by the high participation of TTP in [4], Zhou and Gollman [5] proposed a more efficient scheme in which TTP is involved only when one of the parties could not obtain the nonrepudiation evidence of the other party. In this case, the involved TTP is regarded as an offline TTP [14]. The protocols above are efficient in exchanging messages undeniably with the participation of TTP, but they cannot ensure data privacy if there is no secure channel to transmit the data because the key is later public to everyone. Aiming at extending nonrepudiation to multiparty scenario, Kremer and Markowitch [2] proposed a multiparty nonrepudiation protocol, but it still relies on TTP, while as we all know that a proper TTP is not always available.

A typical way to achieve nonrepudiation without introducing TTP is to increase the interactions and release the secrets gradually [7, 8]. More precisely, Markowitch and Roggeman [7] proposed a nonrepudiation protocol by randomly sending a true or fake decryption key to the recipient after the recipient has confirmed receiving the ciphertext. And the recipient is demanded to return the receipt of the key in every iteration within a time interval of δ, which is much shorter than the decryption time. Once the recipient misbehaves, the sender will terminate the protocol. In such a scheme, the recipient will have a nonnegligible advantage to obtain the true key when receiving a key if he refuses to respond to the sender. Mitsianis [8] realized the same goal by cutting the decryption key into fragments, and the sender sends a fragment to the recipient in each iteration. Similar to Markowitch's approach, the recipient still has an advantage. Furthermore, all these non-TTP approaches get lower efficiency.

In addition to the problem of relying on TTP and low efficiency, the existing traditional nonrepudiation solutions still have an unsolvable problem, that is, the storage of the evidence is accomplished by the participants themselves or the TTP. Considering the constraint of storage space,

business migration, or human error, these records of evidence are generally not permanent. Once the record disappears, there will be hidden trouble in accountability.

2.1.2. Blockchain and Relevant Nonrepudiation Approaches. Blockchain technology emerged as the low-level technology of the Bitcoin cryptocurrency system [15], which is actually a decentralized digital ledger. Each node in the blockchain system maintains an identical backup of the whole ledger (a node only saves part of the ledger in shared blockchains, e.g., [16]), and the consistency of the ledger on these nodes is ensured through a consensus protocol [15, 17]. The mechanism of the blockchain ensures that the data is immutable once been written into the blockchain, since the attacker has to pay dearly to control more than half of the blockchain nodes to temper with the ledger. To implement rather complex functions over the blockchain and even realize decentralized applications, smart contract [18] is supported by many blockchain platforms (e.g., Ethereum [19] and Hyperledger Fabric [20]). In Ethereum, smart contracts are written in Solidity [21], while in Hyperledger Fabric, smart contracts support many traditional programming languages such as golang [22] and nodejs [23], and thus much more richly functional interfaces are supported in Fabric.

Besides its wide application in cryptocurrency systems, there are two main strands of the nonfinancial application of blockchain technology. One is to use blockchain to store data (e.g., Merkle Tree [24] root) for later verification [25, 26], and the other is to exploit Bitcoin scripts or smart contracts to realize specific functions such as replacing TTP [10, 27, 28] or notarizing integrity of data [29]. Schemes or protocols that consider nonrepudiation with the help of blockchain have been proposed recently (e.g., [9–12]). Hubert et al. [10] proposed an extension of TLS protocol called TLS-N to provide each TLS session with the nonrepudiation evidence. TLS-N defines nonrepudiation of a TLS session and the protocol generates nonrepudiation evidence by a generator, which usually refers to a web server with an authorized TLS certificate. The evidence is signed and can be verified by any third party through public key infrastructure (PKI). TLS-N adopts Ethereum blockchain to replace the third parties to verify the nonrepudiation evidence. Schemes proposed in [9, 12] are mainly used for dispute resolution in distributed environments. Both of them exploit smart contracts to process and record the actions of the participants as nonrepudiation evidence, but they do not design enforcement mechanisms to force the participants to confirm their receipts, which is not quite practical to apply.

More recently, Xu et al. [11] proposed a blockchain-based nonrepudiation network computing service scheme for industrial IoT (IIoT) scenario. The scheme solved the nonrepudiation problem of delivering an executable service program from service providers to IIoT clients, in which the blockchain is used as an evidence recorder and a service publication proxy. They realized this by cutting a service program into a small nonexecutable header and the rest part and then delivering them via on-chain and off-chain channels, respectively. The smart contract will force the service provider and the client to interact step by step with some token; any dispute can be resolved through a homomorphic hash-based mechanism [30]. However, in this approach, the delivered programs are not encrypted so that data confidentiality is not supported. Moreover, since the service provider publishes the header part of every service program to the blockchain when delivering them in their scheme, if another IIoT client asks for delivering the same program, the client can obtain the header part directly from the blockchain rather than waiting for the provider to publish. In this case, the client will have an advantage.

Summarily, although there are numerous nonrepudiation supported protocols or schemes, we still cannot find a suitable solution for transmitting some data from a sender to a recipient while supporting nonrepudiation, fairness, data confidentiality, and tolerance of malicious cases at the same time. Therefore, we urgently need such a non-TTP-based nonrepudiation solution for distributed environments.

2.2. Our Contributions

2.2.1. Nonrepudiation Data Transmission Schemes for Both Short Message and Large File. We put forward the construction on secure data transmission problems with nonrepudiation property. To avoid introducing TTP, we exploit blockchain technology to verify the behaviours of participants and promote the execution of the scheme. So, we call our scheme as Blockchain-based Nonrepudiation Data Transmission (BNRDT) scheme. To fit more scenarios and the feature of the blockchain technology, we propose two BNRDT schemes, the original one is for short message transmission (data size less than 1 KB) with fewer interactions, and the other improved one is for large file transmission (data with MB or GB level size). We can choose any scheme according to the environment when transmitting mid-size data. Both the schemes provide nonrepudiation of data transmission process with fair nonrepudiation evidence generation. Meanwhile, the two BNRDT schemes can protect the confidentiality and integrity of the transmitted data and handle all the possible malicious cases.

2.2.2. Security Properties and Theoretical Analysis for Secure Nonrepudiation Data Transmission. We put forward the desired properties of a nonrepudiation secure data transmission scheme to analyze the security of our schemes. In detail, we combine all the necessary security properties we want from both the data transmission and nonrepudiation, including data security, requirement to resistant to malicious cases, nonrepudiation of both the sender and the recipient, and the fairness of nonrepudiation evidence generation. Based on this, we give a detailed security analysis of our schemes.

2.2.3. Prototype Implementation Based on Hyperledger Fabric. We give a prototype implementation of our nonrepudiation data transmission schemes based on

Hyperledger Fabric blockchain. Furthermore, we evaluate the implementation and prove the applicability of the proposed schemes.

3. Preliminaries

In this section, we will introduce some preliminaries including the security model to better describe and analyze the schemes in later sections.

3.1. Notations. The main notations used in this paper are shown in Table 1. Note that $ID_{\mathcal{S}}$ and $ID_{\mathcal{R}}$ represent for the blockchain account identifiers of the sender $\mathcal{S}$ and the recipient $\mathcal{R}$. For example, the identifier of a user can be derived from his certificate in Fabric blockchain. The identifier can uniquely identify the user in the blockchain network.

3.2. Cryptographic Building Blocks

3.2.1. Hash Function. Hash function is used to compress an arbitrary length input string into short and fixed-length string output. An unkeyed hash function can be written as H: $\{0,1\}^* \longrightarrow \{0,1\}^l$. We simply denote the hash calculation of a message M as $h = \text{Hash}(M)$.

We say that a hash function is collision resistant, that is, colliding pairs are unknown and computationally infeasible for a probabilistic polynomial-time (PPT) adversary to find even though they exist [31]. Besides, if a hash function is modelled as a random oracle, this hash function is collision resistant, and any PPT adversary is infeasible to obtain any information about the message M through the hash value h of M.

3.2.2. Commitment Scheme. Take the discrete-log problem-based implementation (e.g., the Pedersen commitment [32]) as an example, a commitment scheme is a tuple of PPT algorithms (ComGen, Com, and Open) executed between two parties: a committer and a recipient. The algorithm ComGen is used for group parameter generation, which will be used for calculating the commitment, and we denote it as $pp \leftarrow \text{ComGen}(1^\lambda)$. For an arbitrary input $x \in \{0,1\}^*$ as the secret to commit, the algorithm Com requires another randomly generated parameter r and the generated parameter pp to produce an commitment com_x, which can be denoted as $\text{com}_x \leftarrow \text{Com}(pp, x, r)$. The algorithm Open takes as input the generated parameter, the commitment com_x, the original secret x, and the randomness r and outputs 1 if $\text{com}_x = \text{Com}(pp, x, r)$ and 0 otherwise, which can be denoted as $\{0,1\} \leftarrow \text{Open}(pp, \text{com}_x, x, r)$.

The commitment scheme have to satisfy two properties including *hiding* and *binding*, where *hiding* means that any PPT adversary cannot obtain any information about x from com$x$ before the corresponding Open is executed. And *binding* means that a specific commitment com_x can only be opened with the original secret x, it is impossible for the committer to open with any other secret $x' \neq x$ [27].

Table 1: Notations used in this paper.

Notation	Meaning
$\mathcal{S}$	Sender of the message or file
$\mathcal{R}$	Recipient of the message or file
$\mathcal{SC}$	The smart contract deployed on the blockchain
$ID_{\mathcal{S}}$, $ID_{\mathcal{R}}$	Blockchain account identifiers of $\mathcal{S}$ and $\mathcal{R}$
pk_{temp}, sk_{temp}	Temporary key pair for asymmetric encryption
M	The short message that needs to be undeniably transmitted
F	The large file that needs to be undeniably transmitted
K	The randomly generated key for symmetric encryption
C or C_x	Ciphertext obtained by symmetric or asymmetric encryption
h or h_x	Hash values
com_x	Commitment value of secret x
$d_{\mathcal{R}}$	The deposit paid by the participant $\mathcal{R}$
L	A unique label that identifies a BNRDT instance
s_L	The state of protocol instance labelled with L

Classically, only algorithm ComGen is executed by the recipient and both Com and Open of a commitment instance are executed by the committer. While, in this paper, we redesign and implement the commitment scheme based on the blockchain and the smart contract. We let the recipient execute the opening operation in order to confirm that he has received the secret. Besides, both the committer and the recipient will interact with the smart contract rather than each other. For implementation details, we refer the reader to the Section 3.2.3.

3.2.3. Encryption Schemes. Both the symmetric and asymmetric encryption schemes are introduced here.

A symmetric encryption scheme is a tuple of PPT algorithms (SymGen, SymEnc, and SymDec) such that SymGen takes as input a security parameter λ and outputs a key k, which can be written as $k \longleftarrow \text{SymGen}(1^\lambda)$. SymEnc takes as input a key k and an arbitrary plaintext M and then outputs the ciphertext C, which can be written as $C \longleftarrow \text{SymEnc}(k, M)$. And SymDec takes as input a key and the ciphertext C and then outputs the original plaintext M or $\perp$. Specifically, when C is the correct ciphertext of M under k, we have $M \longleftarrow \text{SymDec}(k, C)$ or $\perp \longleftarrow \text{SymDec}(k, C)$, otherwise.

We say that the symmetric encryption scheme has indistinguishable encryptions under a chosen-ciphertext attack, i.e., the scheme is IND-CCA (Indistinguishability Chosen-Ciphertext Attack) secure if for any PPT adversary who chooses a pair of messages M_0 and M_1 is given the challenge ciphertext SymEnc (M_b), where b is chosen uniformly at random, and oracle access to $\text{SymEnc}(k, \cdot)$ and $\text{SymDec}(k, \cdot)$ with the limitation that the challenged ciphertext is not allowed; the advantage for the adversary to output a guess bit b' such that the probability of $b' = b$ is not significantly better than random guessing (with the probability of 1/2).

Similar to the above definitions, we define and denote the three PPT algorithms of asymmetric encryption scheme for key generation, encryption, and decryption as $(pk, sk) \leftarrow \text{AsymGen}(1^{\lambda})$, $C \leftarrow \text{AsymEnc}(pk, M)$, and $M \leftarrow \text{AsymDec}(sk, C)$ (or $\perp \leftarrow \text{AsymDec}(sk, C)$), respectively. The ciphertext encrypted under a public key pk can only be decrypted if one knows the corresponding private key sk.

We also define the security of asymmetric encryption schemes. We say that the asymmetric encryption scheme has indistinguishable encryptions under a chosen-ciphertext attack, i.e., the scheme is IND-CCA secure if for any PPT adversary who chooses a pair of messages M_0 and M_1 is given the challenge ciphertext AsymEnc (Mb), where b is chosen uniformly at random, and oracle access to AsymDec($sk, \cdot$) with the limitation that the challenged ciphertext is not allowed; the advantage for the adversary to output a guess bit b' such that the probability of $b' = b$ is not significantly better than random guessing (with the probability of 1/2). When the oracle access is allowed after the adversary gets the challenge ciphertext, the scheme is said to be IND-CCA2 secure with the same output.

3.3. Security Model. Before introducing the construction of our scheme, we present the security properties that we want to achieve and the reasonable trust assumptions here.

3.3.1. Security Properties. To formally describe the security of the scheme, we first give the security objectives we want a nonrepudiation data transmission scheme to achieve.

First, the scheme should provide sufficient data security and transmission fairness, as described in P_1 and P_2 below:

(i) Property P_1 (data confidentiality): when the nonrepudiation data transmission scheme ends normally rather than being interrupted by any malicious behaviour, any adversary cannot recover the original data through the parameters revealed during the data transmission process.

(ii) Property P_2 (fairness of data transmission): when there is a malicious sender or a malicious receiver during data transmission, the scheme guarantees that the malicious participant cannot obtain any advantage over the other participant through performing malicious behaviours.

Furthermore, as a nonrepudiation scheme, the nonrepudiation properties described by P_3 and P_4 below are required:

(i) Property P_3 (nonrepudiation of data transmission): once the scheme ends normally rather than being interrupted by any malicious behaviour, the sender cannot deny that he has sent the data to the receiver (NRO), and the receiver cannot deny that he has received the data from the sender (NRR).

(ii) Property P_4 (fairness of nonrepudiation evidence generation): the nonrepudiation evidence always appears in pairs. For a specific data transmission instance, either all the participants receive the evidence of the other party's behaviour or none of the participants get any valid evidence.

3.3.2. Security Definition. Now, we give the security definition of a nonrepudiation data transmission scheme.

Definition 1. If a nonrepudiation data transmission scheme satisfies properties P_1, P_2, P_3, and P_4 at the same time, we say that the scheme is a secure nonrepudiation data transmission scheme.

3.3.3. Trust Assumptions. We also give some trust assumptions which are necessary. First, we assume that both the sender and the recipient of a data transmission instance are possible to cheat the other party due to their self-interests, but neither of them will perform abnormal behaviours that are detrimental to their own interests, such as revealing their own private key to the adversary. Second, we assume that the blockchain (including the smart contract) is a trusted component. Once the transaction is confirmed by enough nodes, the data on the blockchain is trusted and immutable. For the smart contract, the adversary cannot modify its internally deployed algorithms. We think the last assumption is also reasonable because adversaries have to pay a great price to modify the historical data on the blockchain. In addition, we point out that data on the blockchain will be open to all scheme participants, so there will be no secrets on the blockchain.

4. Construction of BNRDT Scheme

In this section, we will introduce the construction of the original BNRDT scheme for short message transmission. Regularly, the BNRDT scheme involves three parties including a sender $\mathcal{S}$, a recipient $\mathcal{R}$, and a dedicated smart contract $\mathcal{SC}$ designed by us. Assume that $\mathcal{S}$ needs to transmit a short message m to $\mathcal{R}$. At the beginning of the scheme, a recipient $\mathcal{R}$ initiates a data transmission instance by launching a blockchain transaction to $\mathcal{SC}$ with his deposit, then $\mathcal{S}$ sends a transaction to $\mathcal{SC}$ with a commitment of the message encrypted by a temporary public key provided by $\mathcal{R}$, and later the recipient sends his confirmation by opening the commitment to withdraw his deposit. With the help of $\mathcal{SC}$, the whole data transmission process is undeniably recorded in the blockchain ledger.

Our BNRDT scheme requires that the recipient submits the deposit in the first message of an instance, which ensures the smart contract to force the recipient to open the commitment submitted by the sender, thus to get the deposit back. If the recipient misbehaves, he will never get back the deposit. On the contrary, if the sender misbehaves, we guarantee that the deposit will be returned to the recipient in the last phase of the scheme.

Now, we outline how we use the smart contract and the mentioned cryptographic building blocks to undeniably transmit a short message from $\mathcal{S}$ to $\mathcal{R}$. We will introduce the initialization of the scheme firstly and then describe the phases in detail.

4.1. BNRDT Scheme Initialization. While initializing the scheme, our smart contract $\mathcal{SC}$ is deployed to the blockchain, which is independent of any scheme instances and needs to be executed only once. After finishing deployment, the address of the smart contract and the algorithms are revealed to the public. Then, the scheme can be executed multiple times. The deployed smart contract includes 6 algorithms, each algorithm can be triggered with its algorithm identifier (its name) and the necessary parameters in appropriate phases:

(i) *TransRequest* (L, $\mathrm{ID}_{\mathcal{S}}$, $\mathrm{ID}_{\mathcal{R}}$, pk_{temp}, pp): this algorithm handles the data transmission request coming from any invoker and initiates a new BNRDT instance. The parameter L identifies the instance, and $\mathrm{ID}_{\mathcal{S}}$ and $\mathrm{ID}_{\mathcal{R}}$ represent the blockchain account ID of the sender $\mathcal{S}$ and the recipient $\mathcal{R}$, respectively. pk_{temp} is a temporary asymmetric public key that will be used for asymmetric encryption, and pp represents the parameter which will be used by commitment scheme.

(ii) *PubComm* (L, com, C): this algorithm publishes the invoker's commitment com for some secret. And C represents the ciphertext that the committer wants to publish to the public with the commitment, which can be used by the commitment opener to recover the secret. The first parameter L binds this transaction with a specific data transmission instance.

(iii) *OpenComm* (L, com, x, r): this algorithm checks the commitment com with respect to the secret x and the randomness r. Similarly, the first parameter L binds this transaction with a specific data transmission instance. This algorithm has two possible execution results which will lead the scheme to different directions. If x and r succeed in opening the commitment com, the scheme ends normally; otherwise the scheme gets into the dispute resolving phase.

(iv) *Complain* (L, sk_{temp}): this algorithm handles the complain requests to resolve disputes arose when opening commitments. In order to trigger this algorithm, the invoker has to provide the label L and the temporary asymmetric private key sk_{temp}, which is to decrypt the published ciphertext when invoking *PubComm*.

(v) *Terminate* (L): this algorithm is designed for the honest parties to terminate the scheme instance labelled with L in the cases that the other participant refuses to make responses in the previous phase.

(vi) *QueryState* (L): this algorithm handles the query requests coming from any invoker to check the current state of any existing BNRDT instances. It will return the state of the instance labelled with L.

Also note that there is a publicly available blacklist maintained by the smart contract, which records all the blockchain identity IDs of the malicious participants of any BNRDT instance. Any participant with its ID in the blacklist is no longer allowed to use the scheme in the future. This blacklist can also provide a reference for the honesty of the blockchain IDs for other blockchain applications.

4.2. Phases of BNRDT Scheme. The detailed workflow of the BNRDT scheme is shown in Figure 1. Usually, a BNRDT data transmission instance will only include the first three phases: *Data Transmission Request*, *Publish Commitment*, and *Open Commitment*. Only when there is malicious behaviour, the scheme is possible to enter phase Complain. We point out that the algorithm Terminate is also a solution to simple malicious cases to abort the execution, but we do not show it in the figure. In order to distinguish different BNRDT data transmission instances, we label every instance with a unique random number L. And for each instance labelled with L, the smart contract holds a state variable s_L to indicate the current state of the execution. Besides, we point out that when $\mathcal{S}$ checks the parameters, it means $\mathcal{S}$ checks the number and correctness of the parameters, rather than checking the internal formats. Now, we give a detailed description of these phases.

4.2.1. Phase 1: Data Transmission Request

(a) $\mathcal{R}$: in this phase, the recipient $\mathcal{R}$ runs RandomGen (1^{λ}) to obtain a unique label L to identify this data transmission instance, runs ComGen (1^{λ}) to generate a commitment scheme parameter pp, runs AsymGen (1^{λ}) to generate a temporary asymmetric encryption key pair $(pk_{\mathrm{temp}}, sk_{\mathrm{temp}})$, and initiates a blockchain transaction to send the following messages: the algorithm identifier of *TransRequest* which is waiting to be executed by $\mathcal{SC}$, a tuple (L, $\mathrm{ID}_{\mathcal{S}}$, $\mathrm{ID}_{\mathcal{R}}$, pk_{temp}, pp), which is necessary for $\mathcal{SC}$ to run the algorithm, and some blockchain currency $d_{\mathcal{R}}$ chosen by $\mathcal{R}$ as a deposit to open the commitment.

(b) $\mathcal{SC}$: upon receiving the tuple *TransRequest* (L, $\mathrm{ID}_{\mathcal{S}}$, $\mathrm{ID}_{\mathcal{R}}$, pk_{temp}, pp, $d_{\mathcal{R}}$) from any invoker, $\mathcal{SC}$ first ensures that the invoker has paid the deposit and then executes *TransRequest* (L, $\mathrm{ID}_{\mathcal{S}}$, $\mathrm{ID}_{\mathcal{R}}$, pk_{temp}, pp) to process the BNRDT data transmission request. Inside the algorithm, $\mathcal{SC}$ makes sure that the invoker is $\mathrm{ID}_{\mathcal{R}}$ and then checks the label L and the IDs, and if L already labelled any other instance or any of the IDs is in the blacklist, $\mathcal{SC}$ will reject the request. Otherwise, $\mathcal{SC}$ initiates a BNRDT instance, stores these parameters, and sets the state of the instance as $s_L \leftarrow$ requested.

4.2.2. Phase 2: Publish Commitment

(a) $\mathcal{S}$: once the sender $\mathcal{S}$ in the listening state sees the transaction to $\mathcal{SC}$ contains his account identifier $(\mathrm{ID}_{\mathcal{S}})$ and the state s_L is set to requested, he is ready to transmit the message m to the requester $\mathcal{R}$. Firstly, $\mathcal{S}$ obtains the request parameters from the

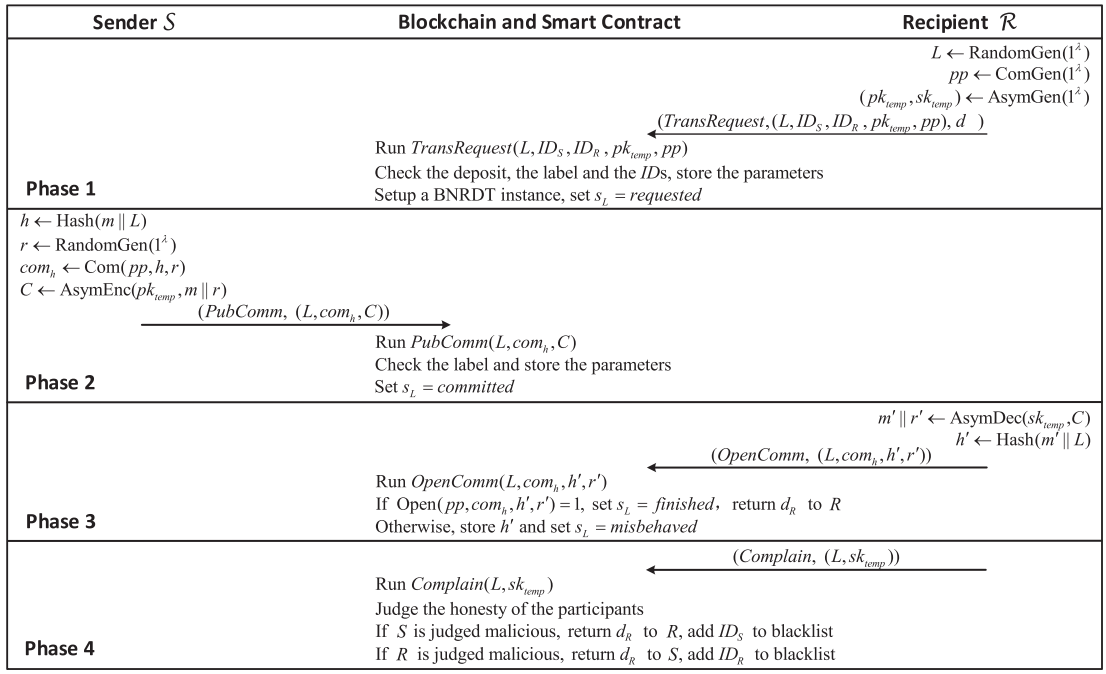

Figure 1: Workflow of the original BNRDT scheme for short message transmission.

blockchain and calculates the hash of the message appended by the label, which is $h \leftarrow \text{Hash}(m \| L)$. Then, $\mathcal{S}$ runs the algorithm $r \leftarrow \text{RandomGen}(1^\lambda)$ to generate a random value and calculate the commitment of the hash value h with the obtained parameter pp, which can be denoted as $\text{com}_h \leftarrow \text{Com}(pp, h, r)$. After that, $\mathcal{S}$ uses $\mathcal{R}$'s temporary public key pk_{temp} to asymmetrically encrypt the original message and the randomness r and generates the ciphertext $C \leftarrow \text{AsymEnc}(pk_{\text{temp}}, m \| r)$. Finally, $\mathcal{S}$ initiates a blockchain transaction to send the following parameters to $\mathcal{SC}$: the algorithm identifier *PubComm* and the tuple (L, com_h, C), which is necessary for $\mathcal{S}$ to run *PubComm*.

(b) $\mathcal{R}$: if $\mathcal{S}$ does not send the blockchain transaction to respond to the data transmission request in phase 1, $\mathcal{R}$ can choose to continue waiting for the response of committing, or to initiate a terminate transaction by sending the tuple (Terminate, L) to $\mathcal{S}$. In this way, $\mathcal{R}$ can terminate the protocol and redeem his deposit $d_{\mathcal{R}}$ back immediately.

(c) $\mathcal{SC}$: upon receiving a committing tuple (*PubComm*, (L, com_h, C)) from an invoker whose account identifier is $\text{ID}_{\mathcal{S}}$, $\mathcal{S}$ executes $PubComm(L, \text{com}_h, C)$ to check the state variable s_L, and $\mathcal{S}$ rejects the transaction if $s_L \neq$ requested. Otherwise, $\mathcal{S}$ stores the parameters and sets $s_L \longleftarrow$ committed.

Upon receiving a tuple (Terminate, L) from an invoker whose account identifier is $\text{ID}_{\mathcal{R}}$, $\mathcal{SC}$ runs Terminate(L) to check the state variable sL and rejects the terminating request if $s_L \neq$ requested. Otherwise, $\mathcal{S}$ sets $s_L \leftarrow$ terminated, returns $d_{\mathcal{R}}$ back to $\mathcal{R}$, and terminates the instance.

4.2.3. Phase 3: Open Commitment

(a) $\mathcal{R}$: the recipient $\mathcal{R}$ in the listening state can now obtain the ciphertext C from the blockchain. Thus, $\mathcal{R}$ runs AsymDec (sk_{temp}, C) to decrypt the ciphertext C, recover $m' \|$, and then $\mathcal{R}$ obtains the original message m'. Next, $\mathcal{R}$ also calculates hash $h' \leftarrow \text{Hash}(m' \| L)$ to get the secret value. If the AsymDec algorithm output $\perp$, $\mathcal{R}$ can set an arbitrary string as h'. Now, $\mathcal{R}$ initiates a blockchain transaction with the tuple (*OpenComm*, $(L, \text{com}_h, h', r')$) to $\mathcal{SC}$. In this way, $\mathcal{R}$ opens the commitment com_h and redeems his deposit $d_{\mathcal{R}}$ back if the obtained secret is correct.

(b) $\mathcal{SC}$: upon receiving a tuple (*OpenComm*, $(L, \text{com}_h, h', r')$) from an invoker whose account identifier is $\text{ID}_{\mathcal{R}}$, $\mathcal{SC}$ runs $OpenComm(L, \text{com}_h, h', r')$. Concretely, $\mathcal{S}$ checks the state variable sL and rejects the request if $s_L \neq$ committed. If the state check passes, $\mathcal{SC}$ executes $\text{Open}(pp, com_h, h', r')$. If the output is 0, $\mathcal{SC}$ sets $s_L \leftarrow$ misbehaved and stores h' and r' which will help $\mathcal{SC}$ judge whether the sender or the recipient has misbehaved. Otherwise, $\mathcal{SC}$ stores h' and r', sets $s_L \leftarrow$ finished, and returns $d_{\mathcal{R}}$ back to $\mathcal{R}$. In this case, the scheme ends normally.

4.2.4. Phase 4: Complain

(a) $\mathcal{R}$: the recipient $\mathcal{R}$ can initiate a blockchain transaction with the tuple (Complain, (L, sk_{temp})) to

$\mathcal{SC}$ in this phase. By uploading his temporary private key sk_{temp}, $\mathcal{R}$ can appeal against the decision $\mathcal{SC}$ made in the previous phase (Phase 3) and finally get his deposit back.

(b) $\mathcal{SC}$: upon receiving the tuple (Complain, (L, sk_{temp})) from an invoker whose account identifier is ID$_{\mathcal{R}}$, $\mathcal{SC}$ executes Complain (L, sk_{temp}) to process the complain request. Concretely, $\mathcal{SC}$ checks the state variable s_L and rejects the request if $s_L \neq$ misbehaved. If the state check passes, $\mathcal{SC}$ continues to check and judge $\mathcal{R}$ to be malicious as long as one of the following statements is positive:

(i) The uploaded sk_{temp} is not the corresponding private key of sk_{temp}.

(ii) $\perp \leftarrow \text{AsymDec}(sk_{\text{temp}}, C)$.

(iii) $m''\|r'' \leftarrow \text{AsymDec}(sk_{\text{temp}}, C)$, $h'' \leftarrow \text{Hash}(m''\|L)$, but $h'' \neq h'$ or $r'' \neq r'$, in which h' and r' are uploaded and stored in Phase 3.

In contrast, if all the above statements are negative, which means $\mathcal{SC}$ succeeded in decrypting C with sktemp and the outputs match with the uploaded parameters by $\mathcal{R}$, $\mathcal{SC}$ judges $\mathcal{SC}$ to be malicious.

If $\mathcal{S}$ is judged to be malicious, $\mathcal{SC}$ returns $d_{\mathcal{R}}$ to $\mathcal{R}$ and adds ID$_{\mathcal{S}}$ to the blacklist. Otherwise, if $\mathcal{R}$ is judged to be malicious, $\mathcal{S}$ returns $d_{\mathcal{R}}$ to $\mathcal{S}$ and adds ID$_{\mathcal{R}}$ to the blacklist. Finally, $\mathcal{SC}$ sets s_L = terminated and terminates the protocol.

4.3. Further Discussion. We now examine some crucial details of the above BNRDT scheme. First of all, we point out the reason why the recipient is required to generate a temporary public/private key pair rather than using its long-term key pair. The generated key pair is used for asymmetric encryption to protect the message from being obtained by any other third party. There is no problem using the recipient's key pair when both the participants are honest. However, if a malicious sender publishes a commitment com$_h$ which is not corresponding commitment to the hash of the ciphertext C, which means the honest recipient will never manage to find the secret (hash value of $m\|L$) to open the com$_h$ through decrypting C. In this case, we need the smart contract to reproduce the calculation process of an honest recipient to prove his innocence. Therefore, the smart contract needs the private key. And the disclosure of a temporary generated private key will not affect the security of the recipient's account or some other information.

We would also like to explain why we take the hash value of the message m appended with the random label L as the secret of the commitment. Note that one of the goals we want to achieve is to make the data transmission process undeniable, including making sure that the recipient has received the right message. However, we also need to consider the presence of malicious senders; thus, the recipient actively confirms receiving the message will be a good choice. Therefore, we ask the recipient to calculate and publish the hash value of the original message with the label; by this way, we make sure that the recipient has received the correct message and the confidentiality of the original data will not be affected. Meanwhile, the smart contract needs to know whether the opened hash is the right one, so we take it as the secret of the commitment, and let the smart contract execute the Open algorithm. The label L is appended to the message when calculating hash as a salt value as well as to bind the message with the instance label, which also ensures that one hash value will not be recorded in the blockchain twice or more times.

Besides, we point out that the algorithm *QueryState* integrated in the smart contract is designed for state checking. Only when the current state s_L of a BNRDT data transmission instance labelled with L is set to finished, it means the data transmission process ended normally rather than being interrupted by any malicious behaviour, and the nonrepudiation evidence (including NRO and NRR) of this data transmission process is generated and stored in the blockchain. And the state of the instance will be one of the most important evidence to resolve disputes and judge the validity of the nonrepudiation evidence. We refer the readers to Section 6 for more detailed analysis and proof of the scheme.

Although the proposed BNRDT scheme can securely transmit the message between two blockchain users with the support of nonrepudiation property (we prove it in Section 6), it is not a perfect solution for data transmission since the message is directly encrypted by an asymmetric key and transmitted over the blockchain, while everyone knows that asymmetric encryption is not suitable for large files, and the blockchain is not suitable for transmission of large files either. Therefore, we present another version of BNRDT in Section 5 to deal with this problem, which supports large file transmission.

5. An Improved Version of BNRDT Scheme Supporting Large File Transmission

As mentioned before, the proposed BNRDT scheme is qualified for short message transmission but not suitable for large files, mainly because the message is directly encrypted by an asymmetric key and the message is transmitted on-chain. In order to support large file transmission, in this section, we give an improved version of the original BNRDT scheme which uses symmetric encryption scheme to encrypt data and transmits data through an off-chain channel.

In order to transmit and protect a large file, we can use an efficient symmetric encryption scheme (e.g., AES) to encrypt the file under a randomly generated symmetric key k, and since the original BNRDT is qualified for secure short messages transmission, we can use it to distribute k undeniably. Note that the symmetrically encrypted ciphertext needs to be transmitted and confirmed before key distribution so that the smart contract can judge the malicious behaviours when sending and confirming ciphertext. Now, we describe the implementation detail of the scheme.

5.1. Construction. The detailed workflow of the improved BNRDT scheme is shown in Figure 2. Similar to the original BNRDT, every nonrepudiation data transmission instance

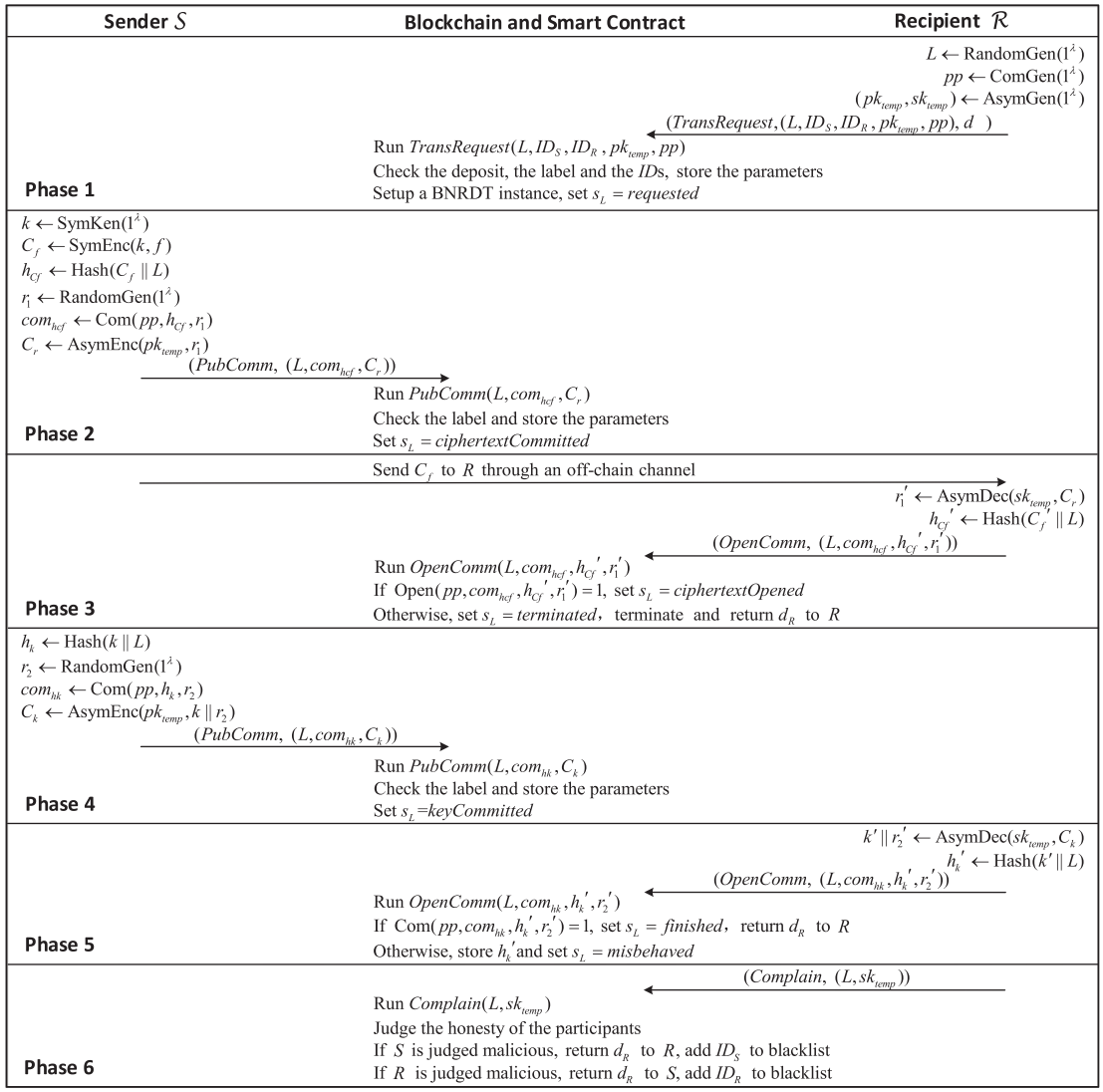

Figure 2: The improved BNRDT scheme for large file transmission.

requires to be labelled with a unique random identifier L. To improve the efficiency of data encryption, we apply symmetric encryption in phase 2. Therefore, our goal is to transmit both the symmetric encryption key and the ciphertext undeniably. As you can see in Figure 2, phases 1, 4, 5, and 6 constitute a complete data transmission instance of the original BRNDT scheme, which transmits the symmetric encryption key k randomly generated in phase 2. What we still need to do is to transmit the ciphertext of the original data through an off-chain channel. To achieve this goal, we introduce the other two phases (Phase 2 and 3).

In phase 2, the symmetric encryption key k is generated and being used to encrypt the original data, which is $C_f \leftarrow \text{SymEnc}(k, f)$. Similar to the *commit-open* process of transmitting k, we also calculate hash on ciphertext C_f and the label. Then, we take this hash value as the secret to commit in the commitment. Therefore, $\mathcal{S}$ is required to select a randomness r_1 and execute $\text{com}_{hcf} \leftarrow \text{Com}(pp, h_{Cf}, r_1)$. After that, $\mathcal{S}$ exploits the obtained temporary public key sk_{temp} to encrypt the randomness r_1. Finally, $\mathcal{S}$ reveals the parameters to $\mathcal{S}$ through a blockchain transaction to trigger the *PubComm* algorithm.

In phase 3, $\mathcal{S}$ sends the ciphertext C_f to $\mathcal{R}$ through an arbitrary off-chain channel rather than transmitting through the blockchain, thus to support large file transmission. Also, the recipient is required to open the commitment com_{hcf} by initiating the *OpenComm* transaction with the obtained and calculated parameters h'_{cf} and the randomness r'_1. Therefore, $\mathcal{R}$ makes everyone know that he has successfully received the right ciphertext. Upon receiving the transaction, $\mathcal{S}$ executes $OpenComm(L, \text{com}_{hcf}, h'_{cf}, r'_1)$ to verify whether the opened secret is the committed one or not, as shown in the figure. Here, in phase 3, the way that $\mathcal{S}$ handles the verification result is different from that in phase 5 (as well as phase 3 in the original BNRDT scheme). That is, if *OpenComm* $(L, \text{com}_{hcf}, h'_{cf}, r'_1) = 1$, everything goes right and the scheme proceeds. Otherwise, $\mathcal{S}$ will directly terminate the scheme and return the deposit $d_{\mathcal{R}}$ to $\mathcal{R}$, since there must be something malicious or unexpected happened. It may be that $\mathcal{S}$ sent the wrong ciphertext, or $C_{\mathcal{R}}$ intentionally

uploaded the wrong commitment secret, or even the ciphertext was changed when being transmitting off the chain. However, $\mathcal{SC}$ cannot judge what indeed happened since no one knows what the $\mathcal{R}$ has received indeed. Therefore, the best choice is to terminate the data transmission instance and no one suffers a loss.

Finally, if h_k' and r_2' successfully opens the commitment about the key, $\mathcal{R}$ runs $f \leftarrow \text{SymDec}\ (k',\ C_f')$ to get the original transmission data. See Section 6, for security analysis and nonrepudiation proof of this scheme.

6. Security Analysis

In this section, we give the detailed security analysis of our BNRDT schemes based on the security properties described in the security model. To prove that our schemes satisfy Definition 1, we give the following theorem.

Theorem 1. *If the hash function is modelled as a random oracle, the asymmetric and symmetric encryption schemes provide IND-CCA2 security, the commitment scheme provides secure hiding and binding properties, and the blockchain as well as the smart contract is trusted and immutable, and the proposed BNRDT schemes are secure data transmission schemes.*

Proof. We begin our proof by considering five types of attacks:

(1) A malicious outsider attempts to obtain the transmission data through the published parameters of a BNRDT instance, while both the scheme participants are honest (Lemma 1)

(2) A malicious participant of any BNRDT scheme attempts to cheat the other side by performing malicious behaviours (Lemma 2)

(3) A malicious sender attempts to deny the fact that he has sent the data to the recipient through any BNRDT scheme (Lemma 3)

(4) A malicious recipient attempts to deny the fact that he has received the data from the sender through any BNRDT scheme (Lemma 4)

(5) A malicious participant of any BNRDT scheme attempts to generate nonrepudiation evidence for himself while also prevent the other one to generate it (Lemma 5)

We start with Lemma 1.

Lemma 1. *The data transmitted through BNRDT schemes is confidential against any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ if the hash function integrated in BNRDT is modelled as a random oracle, and the encryption schemes provide IND-CCA2 security.*

Proof. In our original BNRDT scheme for short message transmission, when scheme ends normally, that is, when the scheme ends after phase 3 and the state variable s_L is set to finished, the revealed public parameters include the blockchain IDs of the participants, the temporary generated public key, the random label L, the hash value h of the original message appended with the label, the commitment scheme parameter pp, the randomness r, the commitment value com_h, and the asymmetrically encrypted ciphertext C of the message appended with the randomness. Parameters that may be useful to the adversary to recover the original data includes the public key pk_{temp}, the ciphertext C, and the hash value h.

Since the asymmetric encryption scheme provides IND-CCA2 security, any PPT adversary $\mathcal{A}$ cannot obtain any information about m through C and pktemp. Besides, since the hash function is modelled as a random oracle, $\mathcal{A}$ is impossible to get any information about the original message from the hash value h, either. Furthermore, since the randomly generated label L is appended to the original message when calculating hash, it is impossible to find two BNRDT instances that reveal the same hash values or commitment values, which further hides the original message.

In our improved BNRDT scheme for large file transmission, we exploit the original scheme to transmit the symmetric encryption key, and we use an off-chain channel to transmit the ciphertext of symmetric encryption. In this case, two more parameters including the symmetric encryption ciphertext and its hash value are revealed to the adversary. However, with the assumption of IND-CCA2 security of symmetric encryption scheme and random oracle of the hash function, the adversary still cannot obtain any information about the original data. With the above analysis of the original BNRDT scheme, the adversary is also impossible to obtain the symmetric encryption key. Therefore, the adversary can neither recover the original data through the symmetric encryption ciphertext nor obtain the key to decrypt the ciphertext.

To conclude the proof of Lemma 1, our schemes guarantee that the transmission data is confidential against any PPT adversary with the assumptions.

Remark 1. We point out that the assumption that the hash function is modelled as a random oracle is reasonable in the blockchain environment since the security of the PoW consensus protocol on many existing blockchain platforms such as Bitcoin and Ethereum also relies on this assumption.

Lemma 2. *The BNRDT schemes guarantee that the malicious participant cannot obtain any advantage over the other party through performing any malicious behaviour if the integrated commitment scheme in BNRDT provides secure hiding and binding property, the encryption schemes provide IND-CCA2 security, and the blockchain as well as the smart contract is trusted and immutable.*

Proof. Recall the design of the two schemes, the behaviours of a malicious participant can be divided into two categories, one is refusing to respond to the smart contract or the other participant, and the other is uploading incorrect parameters to the smart contract. For the first type, we have an algorithm inside the smart contract called Terminate for the honest

participant to terminate the scheme when needed. Precisely, Terminate can be called in Phase 2 by the recipient in both the original and the improved BNRDT schemes when the sender refuses to make the commitment, in Phase 3 of the improved BNRDT by the sender when the recipient refuses to open the commitment, and in Phase 4 of the improved BNRDT scheme by the recipient when the sender refuses to make the commitment about the key. Through algorithm Terminate, the schemes can handle all of these cases easily. A problem comes to an honest sender that once he publishes the ciphertext of the message in phase 3 of the original BNRDT scheme (or ciphertext of the key in phase 5 of the improved scheme), and the recipient will be able to decrypt the asymmetric ciphertext and then obtain the original data, whether the recipient opens the commitment or not. However, since the recipient has not redeemed his deposit $d_{\mathscr{R}}$ yet, if $\mathscr{R}$ does not initiate the opening transaction, he also gets a disadvantage of losing his deposit. This disadvantage can be expanded by increasing the amount of deposit, which offsets the advantage of the recipient. Therefore, the BNRDT schemes are able to handle all the possible cases that a malicious participant does not make responses in the schemes.

Furthermore, since the commitment scheme provides secure *hiding* and *binding* property, the recipient is impossible to obtain the secret hash from the commitment value and impossible to open the commitment with some other parameter $h' \neq h$ and $r' \neq r$, where h is the committed secret and r is the selected randomness when committing. Because the smart contract always performs necessary checks including the identity of the invoker, the current state of the BNRDT instance, and the correctness of the parameters, it is also impossible for a malicious participant to upload any incorrect parameter to the smart contract to cheat the other party. While we point out that if a malicious sender publishes an incorrect commitment value com$h$1 with a correct asymmetric encryption ciphertext C in phase 2 in the original BNRDT scheme or responses an incorrect com_{hk1} with a correct C_k in phase 4 in the improved BNRDT scheme, an honest recipient will asymmetrically decrypt the correct ciphertext to get the correct message or the correct key. However, the calculated hash value of the correct message or correct key is impossible to convince the smart contract to return the deposit back to the recipient in the following *OpenComm* transaction, since it is obvious that $\text{Open}(pp, \text{com}_{h1}, h', r') = 0$ in phase 3 of the original BNRDT scheme and $\text{Open}(pp, \text{com}_{hk1}, h'_k, r'_1) = 0$ in phase 5 of the improved BNRDT scheme. If no further measure is taken, such a malicious sender will succeed in cheating and causing an honest recipient to lose his deposit, even though the recipient did not do anything wrong. The case that a malicious sender sends a correct commitment with an incorrect asymmetric encryption ciphertext will result in a similar problem.

Phase Complain handles the above cases. In this phase, the honest recipient uploads the temporary private key to the smart contract; thus, the smart contract can reproduce the process of decryption and hash calculation and help the recipient to prove his innocence. If any BRNDT instance comes to phase Complain, the scheme ends abnormally and the original date will be available publicly due to the malicious sender, which goes beyond the protection of data confidentiality.

Note that the IND-CCA2 security assumption of the symmetric and asymmetric encryption schemes is also required, thus to guarantee that the recipient cannot directly recover the plaintext of the ciphertext and terminate the scheme without initiating the opening transaction in the following cases:

(1) The recipient receives the symmetric encryption ciphertext of the original data in phase 3 of the improved BNRDT scheme

(2) The sender publishes the asymmetric encryption ciphertext of the key in phase 4 of the improved BNRDT scheme

(3) The sender publishes the asymmetric encryption ciphertext of the original message in phase 2 of the original BNRDT scheme

We point out that the analysis above is based on the immutability of and the trust against the blockchain. To conclude the proof of Lemma 2, our schemes guarantee that the malicious participant will not get any advantage under these assumptions.

Lemma 3. *The BNRDT schemes guarantee that once the scheme ends normally rather than being interrupted by any malicious behaviour, any recipient of the BNRDT instances cannot deny the fact that he has received the data from the specific sender if the hash function integrated in BNRDT is modelled as a random oracle, the commitment scheme provides secure hiding and binding property, and the blockchain as well as the smart contract is trusted and immutable.*

Proof. When an instance of the BNRDT scheme ends normally, to prove the fact that the recipient has received the data from the specific sender, we need to prove that

(1) The data is originated from the sender

(2) The recipient has received the correct data

If the participants transmit a large file through the improved BNRDT scheme, the sender will make two commitments: one is the hash of the symmetric encryption ciphertext and the other is the hash of the key. And the recipient is required to open the two commitments by decrypting and hash calculating. Since the commitment scheme provides secure *binding* property, the sender is impossible to obtain the same commitment value of the input h through some other secret $h' \neq h$. Besides, the algorithm *PubComm* always checks that the transaction invoker is the sender specified in phase *TransRequest*, which binds the commitment with the sender. Therefore, we make sure that if the correct commitment value is revealed to the blockchain, the original secret of the commitment instance is originated from the sender. Moreover, the secret of the commitment scheme is the hash value of the ciphertext or the key; since the hash function is modelled as a random

oracle, the sender is impossible to get the same hash with some other ciphertext or key. Consequently, the scheme ensures that the symmetric encryption ciphertext and the key are originated from the sender.

Similar to the origin of the data, since the hash function is modelled as a random oracle, and the commitment scheme provides secure hiding and binding property, the scheme ensures that the recipient can only open the commitment through calculating hash and uploading the correct parameters. And the algorithm *OpenComm* always binds the opening parameters of the commitment with the recipient. Therefore, once the scheme instance ends normally with state variable s_L = finished, the recipient must have obtained the correct symmetric encryption ciphertext and the key and then the recipient is able to recover the original data. Consequently, the scheme guarantees that the recipient has received the correct data.

Now, we talk about how the behaviour of receiving data can be proved. We still assume that the data is transmitted through the improved BNRDT scheme. In this case, $\mathcal{S}$ claims to have sent the data of file f to $\mathcal{R}$ while $\mathcal{R}$ denies having received f from $\mathcal{S}$. We assume there is a judge, who is the person to whom the sender wants to prove the fact that the recipient has received the data. If $\mathcal{S}$ claims to have sent data f to $\mathcal{R}$ by providing the label L, the transmitted data f, the off-chain transmitted ciphertext C_f, and the symmetric encryption key k as his evidence, $\mathcal{S}$ can easily claim his behaviour with the help of blockchain records. We call this property as Nonrepudiation of Receipt (NRR), while we also need to consider the case that $\mathcal{S}$ is lying. So, as long as one of the following validations fails, we regard $\mathcal{S}$'s claim is invalid:

(i) The judge initiates a transaction of *QueryState* (L) to check that the state variable s_L of the improved BNRDT instance labelled with L is *finished*

(ii) The judge obtains the parameters in *TransRequest* transaction of the improved BNRDT instance labelled with L from the blockchain and checks that $\mathrm{ID}_{\mathcal{S}}$ is the blockchain account identifier of $\mathcal{S}$ and $\mathrm{ID}_{\mathcal{R}}$ is the blockchain account identifier of $\mathcal{R}$

(iii) The judge obtains the parameter h'_{cf} in *OpenComm* transaction in Phase 3, and the parameter $h_{k'}$ in *OpenComm* transaction in Phase 5 of the improved BNRDT instance labelled with L from the blockchain then checks that $h'_{cf} = \mathrm{Hash}\ (C_f|)$ and $h'_k = \mathrm{Hash}\ (k\|C_f\|L)$

(iv) The judge checks that $f = \mathrm{SymDec}\ (k, C_f)$

These necessary checks guarantee that the data $\mathcal{S}$ claimed is indeed the data that $\mathcal{R}$ has successfully received through the BNRDT scheme, and the originator of the data is $\mathcal{S}$. The nonrepudiation evidence is stored on the blockchain. Note that all the analysis above is based on the immutability of and the trust against the blockchain.

Lemma 4. *The BNRDT schemes guarantee that once the scheme ends normally rather than being interrupted by any malicious behaviour, any sender of the BNRDT instances cannot deny the fact that he has sent the data to the specific recipient if the hash function integrated in BNRDT is modelled as a random oracle, the commitment scheme provides secure hiding and binding property, and the blockchain as well as the smart contract is trusted and immutable.*

Proof. To prove the fact that a sender has sent the data to the specific recipient, the proof that the data is originated from the sender and the recipient has received the correct data are also required, please check it in proof of Lemma 3. Now, we talk about how the judge can handle the case that a recipient $\mathcal{R}$ claims to have received some data f from $\mathcal{S}$ while $\mathcal{S}$ denies having sent it to $\mathcal{R}$. If $\mathcal{R}$ can offer the label L, the data f, and the ciphertext C_f, as well as the temporary private key sktemp as his evidence, it can easily claim the behaviour. We call this property as Nonrepudiation of Origin (NRO). Similarly, we also need to consider the case that $\mathcal{R}$ is lying. So, as long as one of the following validations fails, we regard $\mathcal{R}$'s claim is invalid:

(i) The judge initiates a transaction of *QueryState* (L) to check that the state variable s_L of the improved BNRDT instance labelled with L is finished.

(ii) The judge obtains the parameters in *TransRequest* transaction of the improved BNRDT instance labelled with L from the blockchain and checks that $\mathrm{ID}_{\mathcal{S}}$ is the blockchain account identifier of $\mathcal{S}$ and $\mathrm{ID}_{\mathcal{R}}$ is the blockchain account identifier of $\mathcal{R}$.

(iii) The judge obtains the parameter h'_{cf} in *OpenComm* transaction in Phase 3 of the improved BNRDT instance labelled with L from the blockchain and then checks that $h'_{cf} = \mathrm{Hash}(C_f \| L)$.

(iv) The judge obtains the parameter C_k in *PubComm* transaction in Phase 4 of the improved BNRDT instance labelled with L from the blockchain, and executes $k'' \| r = \mathrm{AsymDecrypt}(sk_{\mathrm{temp}}, C_k)$, and then checks that $f = \mathrm{SymDec}(k'', C_f)$.

Similarly, these necessary checks guarantee that the data $\mathcal{R}$ claimed is indeed the data that $\mathcal{S}$ has successfully sent through the BNRDT scheme. Also note that all the analysis above is based on the immutability of and the trust against the blockchain.

Lemma 5. *The BNRDT schemes guarantee that the generation of the nonrepudiation evidence of the finished data transmission instances is fair, which means either all the participants receive the evidence of the other party's behaviour or none of the participants gets any valid evidence, if the blockchain as well as the smart contract is trusted and immutable.*

Proof. In the improved BNRDT scheme, only when the recipient successfully opens the hash value of the symmetric encryption key, the state variable s_L of this BNRDT instance is set to finished, which also means that the scheme ends normally rather than being interrupted by any malicious behaviour. For both the validation of the NRO and the NRR evidence, we require the judge to make sure that the state variable s_L of the BNRDT instance is finished at first. That is

to say, either both the NRO evidence and the NRR evidence of the same BNRDT instance are judged valid, or neither of them is valid. Since the blockchain as well as the smart contract is trusted and immutable, nobody, but the smart contract itself can change the state of s_L when being triggered with the valid parameters. Therefore, the nonrepudiation evidence of NRR and NRO always appear in pairs, which ensures the fairness of generating nonrepudiation evidence.

Note the original BNRDT scheme for short message transmitting is included in the improved one, and the analysis will be completely similar, so we omit it when proving Lemma 3 to 5. Above all, we have accomplished the proof of Theorem 1. To conclude this proof, Lemma 1 has proved that the BNRDT schemes satisfy Property P_1 in the security model, Lemma 2 has proved that the BNRDT schemes satisfy Property P_2, Lemmas 3 and 4 have proved that the BNRDT schemes satisfy Property P_3, and Lemma 5 has proved that the BNRDT schemes satisfy Property P_4. According to Definition 1, the proposed BNRDT schemes are secure data transmission schemes.

7. Evaluation

In this section, we first make a comparison between our BNRDT schemes and some typical nonrepudiation approaches proposed before in many aspects. After that, we will discuss the implementation details and evaluate the performance of our schemes deployed on Hyperledger Fabric blockchain.

7.1. Feature Comparison. In this section, we compare our BNRDT schemes with some typical nonrepudiation approaches including [4, 7, 11] in terms of application, fairness, confidentiality, TTP-independence, malicious participant tolerance, and interactions. We point out that the fairness property here refers to the fairness of generating nonrepudiation evidence of the schemes, while the property P_2 (fairness of data transmission) introduced in the security model is included in malicious participant tolerance. Moreover, the property confidentiality guarantees that the data being transmitted is protected against the outsiders, the property TTP-independence tells us whether the scheme relies on TTP or not, and interactions shows that the number of interactions the participants need to perform, which indicates the scheme's complexity.

The comparison is shown in Table 2, note that the scheme in [7] requires $2n + 3$ interactions between the sender and the recipient, where n represents the number of fake decryption keys sent by the sender. To enhance the security of the scheme, n is always quite large. Also note that the scheme in [11] cannot always provide the property in some specific cases, as we mentioned before, if a program has been delivered via the scheme at some time before and the program is required to be delivered to another client again, the scheme is not able to handle the malicious behaviours of the client. In summary, the comparison indicates that our schemes provide the most balanced and comprehensive performance in nonrepudiation data transmission scenarios.

7.2. Implementation and Performance. To benchmark the performance of our schemes, we implemented and evaluated our BNRDT schemes based on Hyperledger Fabric (version 1.4.1) and a blockchain benchmark framework, Hyperledger Caliper [33] (version 0.2.0). Since Hyperledger Fabric is a permissioned blockchain platform, every user owns a certificate to access the blockchain, the ID of a user can be derived from his certificate. Unlike those permissionless blockchains (e.g., Bitcoin and Ethereum), Fabric blockchain does not have an internal currency; therefore, we need to design it ourselves to meet the requirements of the scheme's deposit. While this is not the focus of our design, so the implementation here uses a key-value pair stored in the blockchain ledger to represent the user and his balance. For the cryptographic implementations, we use the SHA256 algorithm to calculate the hash values, AES for symmetric encryption, ECIES for asymmetric encryption, and the ECC-based Pedersen commitment scheme for commitments. The blockchain smart contract including the above cryptographic algorithms is written in golang programming language. Parameters uploaded when invoking BNRDT algorithms are encoded to hexadecimal characters and finally stored in the blockchain ledger.

The blockchain benchmark framework, Hyperledger Caliper, is introduced here to measure the performance of our protocol. Variables that can be manually controlled when testing include the size of data transmitted using the BNRDT scheme and the send rate of the blockchain transactions. By changing these two parameters, we observe the performance of the BNRDT algorithms on transaction throughput and average transaction latency. In theory, the size of the transmitted data will significantly affect the size of a single Fabric transaction, thereby affecting the time for transaction distribution and block synchronization (which can be called the transaction consensus time), and finally affect the transaction throughput. When the data size is fixed, the maximum throughput of the deployed Fabric network to this particular size of transactions will be fixed. The send rate of transactions identifies the rate at which the Fabric client sends transactions to the blockchain network. When it becomes larger, the blockchain network will eventually be unable to process all transactions in time, resulting in an increase in the average latency of transactions.

We performed the measurements on a five nodes Hyperledger Fabric network with an orderer node and 2 organizations, 2 peer nodes in each organization. Every node is deployed in an Ubuntu 16.04 virtual machine with 1 CPU and 4 GB RAM. Virtual machines to deploy peer nodes of one organization are created in one host physical personal computer with an Intel(R) Core(TM) i5-8400 CPU @2.80 GHz and 16 GB RAM, running Windows 10 (64 bit). Different peer organizations as well as the orderer node are

Table 2: Feature comparison of BNRDT and BNRDT-based data transmission scheme and typical nonrepudiation approaches.

Approaches	Application	Fairness	Confidentiality	TTP-independence	Malicious participant tolerance	Interactions
[4]	Data transmission	✓	×	×	✓	5
[7]	Data transmission	×	✓	✓	×	$2n+3$
[11]	IIoT program delivery	✓ *	×	✓	✓ *	6
Original BNRDT scheme	Short message transmission	✓	✓	✓	✓	3
Improved BNRDT scheme	Large file transmission	✓	✓	✓	✓	5

"✓ * " means the scheme will not provide the property in some specific cases.

deployed on different host machines, such deployment makes the node topology closer to the real application context of Fabric blockchain.

The Caliper benchmark results are shown in Figure 3, including the performance of all the 6 algorithms: *TransRequest*, *PubComm*, *OpenComm*, Complain, Terminate, and *QueryState*. The throughput benchmark results of the algorithms are shown in Figures 3(a)~3(f), and the transaction latency benchmarks are shown in Figures 3(g)~3(l).

As can be seen from Figures 3(a) and 3(g), since the algorithm *TransRequest* has nothing to do with the size of the transmitted data, the time taken by the smart contract to process the transaction is far less than the consensus time and the algorithm *TransRequest* gets quite good performance. There is almost no transaction delay no matter how the data size and send rate change, so the blockchain can always process all the transactions less than a second. However, the performance of the algorithms *PubComm*, *OpenComm*, Complain, and Terminate significantly changes with the two independent variables.

Turning now to the performance benchmark of algorithm *PubComm*, when using the original BNRDT scheme, the third parameter C of the algorithm is the ciphertext obtained by encrypting the original transmission data, so the size of *PubComm* transactions will increase synchronously with the size of the transmission data. As can be seen from Figures 3(b) and 3(h), the transactions can also be processed in time with almost no delay when the send rate and the size of the transmitted data are low. However, the performance decreases significantly with the increase of the data size and send rate. When transmitting 10 KB data, the maximum transaction throughput is only about 28 TPS with huge transaction latency (up to 10 seconds at the send rate of 60 TPS), which is not suitable to use any more. Performance of the algorithms *OpenComm*, *Complain*, and *Terminate* is better than *PubComm* because the transactions initialized by clients do not contain the ciphertext, so the size of the transactions will be fixed. However, these three algorithms also need to read and modify all the stored parameters related to the label in the blockchain ledger, so data synchronizing among the nodes still grows with the transmitting data. That is why the performance of these three algorithms is not as good as the algorithm *TransRequest*. We can see that the benchmark results are consistent with the theoretical analysis.

In terms of algorithm *QueryState*, no matter how the transaction sending rate and the size of transmitted data change, there is almost no transaction delay, which can be explained by the principle of Fabric. Concretely, all peer nodes in the Fabric blockchain network maintain the complete blockchain ledger. Since the *QueryState* algorithm only involves querying the ledger and does not need to modify the blockchain, the peer nodes can directly return the results without submitting and synchronizing the transaction. Therefore, all the query transactions can be processed in time and the *QueryState*t algorithm gets such performance.

To avoid performance degradation caused by the increase in data size, we recommend that users use the improved BNRDT scheme for transmitting data with large-size data. In the improved BNRDT scheme, the length of the data to be asymmetrically encrypted is always short and fixed (less than 150 bytes in hexadecimal encoding). Specifically, the parameter C_r in Phase 2 when invoking the *PubComm* algorithm is originated from the random number r_1 used in the commitment scheme, and the parameter C_k of Phase 4 is originated from the randomly generated symmetric encryption key k and the random number r_2. Therefore, the performance of algorithms *PubComm*, *OpenComm*, Complain, and Terminate will be almost consistent with algorithm *TransRequest*, as shown in Figure 3 when data size is less than 0.15 KB. We choose the improved BNRDT scheme when transmitting data over the size of 5 KB in our environment, thus balancing the burden of the blockchain and the transaction numbers. Users can modulate the threshold value according to the actual blockchain environment.

Overall, these benchmarks indicate the availability of our BNRDT scheme in a practical Hyperledger Fabric network. The original BNRDT scheme we designed can be used to efficiently transmit short messages such as passwords or notifications, and the entire process only needs to initiate three blockchain transactions. Moreover, the scheme provides strong evidence of transmission nonrepudiation. However, when transmitting some big-size data, the direct transmission based on the blockchain will cause unacceptable transaction latency and even result in submission failure of the transaction. In such cases, users can use the improved BNRDT scheme for data transmission by performing two more transactions.

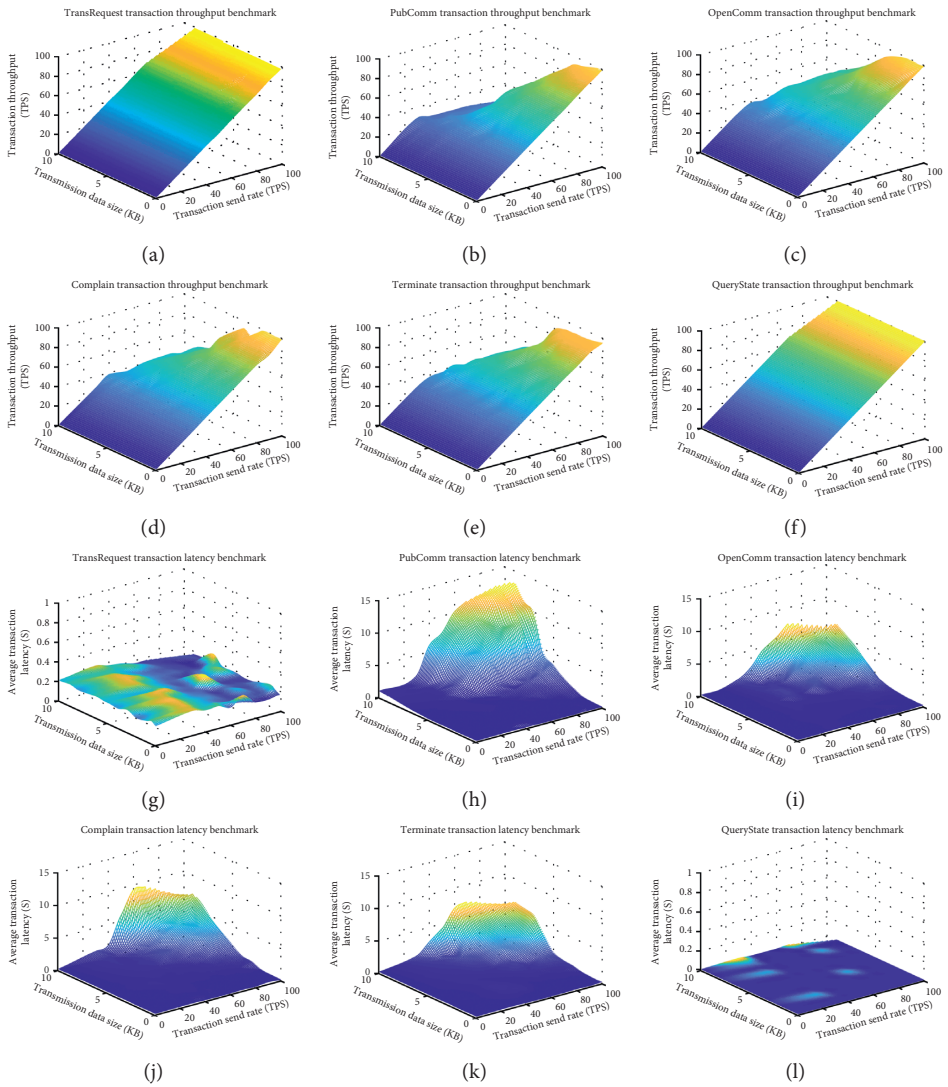

Figure 3: Caliper transaction throughput and latency benchmark of BNRDT algorithms deployed on Hyperledger Fabric. The blockchain network consists of 1 orderer node and 2 peer organizations with 2 peer nodes in each organization.

8. Conclusion

In this paper, motivated by the new requirement of nonrepudiation on secure communication, we start with the propose of a novel data transmission scheme based on blockchain technology and named it as BNRDT, which can get rid of the dependence on TTP and the PKI, and provide data security and nonrepudiation properties at the same time. The original BNRDT scheme requires only 3 interactions to effectively transmit some data in a secure and undeniable manner, but its application is limited by the size of the data. Furthermore, we propose an improved BNRDT scheme based on the original one to support large file transmission at the cost of two more interactions, which is still efficient. Compared with existing work on related topics, our application provides fairness, nonrepudiation, and free of TTP and PKI, as well as the complete security guarantee, i.e., the data is protected against outsiders. For any data transmission instance of the BNRDT schemes, the nonrepudiation evidence is generated and stored directly on the blockchain to realize the properties of Nonrepudiation of origin (NRO) and Nonrepudiation of receipt (NRR) at the same time. Through our experiment on Hyperledger Fabric, we also show the applicability of our schemes. Since data transmission is quite a basic function required in so many scenarios, we firmly believe that the proposed schemes are meaningful.

Data Availability

No data were used to support the findings of the study.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Authors' Contributions

Xiao Lan and Xingshu Chen contributed equally to this work.

Acknowledgments

The authors would like to thank the financial support in part by the Program National Natural Science Foundation of China (NSFC) under Grants U19A2081 and 61802270 and Fundamental Research Funds for the Central Universities under Grants SCU2018D018 and 2019SCU12069.

References

[1] ISO/IEC 13888-1:2009, *Information Technology: Security Techniques, Non-Repudiation—Part 1*, ISO, Geneva, Switzerland, 2009.

[2] S. Kremer and O. Markowitch, "A multi-party non-repudiation protocol," in *Proceedings of the 2000 IFIP International Information Security Conference*, Boston, MA, USA, August 2000.

[3] M. Krotsiani and S. George, "Continuous certification of non-repudiation in cloud storage services," in *Proceedings of the 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, Beijing, China, September 2014.

[4] J. Zhou and D. Gollman, "A fair non-repudiation protocol," in *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 1996.

[5] J. Zhou and D. Gollmann, "An efficient non-repudiation protocol," in *Proceedings of the 10th Computer Security Foundations Workshop*, Rockport, MA, USA, June 1997.

[6] Y. Han, "Investigation of non-repudiation protocols," in *Proceedings of the 1996 Australasian Conference on Information Security and Privacy*, Berlin, Germany, June 1996.

[7] O. Markowitch and Y. Roggeman, "Probabilistic non-repudiation without trusted third party," in *Proceedings of the 2nd Conference on Security in Communication Networks*, vol. 99, Amalfi, Italy, September 1999.

[8] J. Mitsianis, *A New Approach to Enforcing Non-Repudiation of Receipt*, pp. 1–8, 2001.

[9] L. Aniello, R. Baldoni, and F. Lombardi, "A blockchain-based solution for enabling log-based resolution of disputes in multi-party transactions," in *Proceedings of the International Conference in Software Engineering for Defence Applications*, Rome, Italy, May 2016.

[10] R. Hubert, W. Karl, G. Arthur, F. Guillaume, and C. Srdjan, "TLS-N: non-repudiation over TLS enablign ubiquitous content signing," in *Proceedings of the 25th Annual Network and Distributed System Security Symposium, NDSS*, San Diego, CA, USA, February 2018.

[11] Y. Xu, J. Ren, G. Wang, C. Zhang, J. Yang, and Y. Zhang, "A blockchain-based nonrepudiation network computing service scheme for industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3632–3641, 2019.

[12] J. Zou, Y. Wang, and M. A. Orgun, "A dispute arbitration protocol based on a peer-to-peer service contract management scheme," in *Proceedings of the 2016 IEEE International Conference on Web Services (ICWS)*, San Francisco, CA, USA, June 2016.

[13] J. A. Onieva, J. Lopez, and J. Zhou, *Secure Multi-Party Non-Repudiation Protocols and Applications*, Springer Science & Business Media, Berlin, Germany, 1st edition, 2008.

[14] S. Kremer, O. Markowitch, and J. Zhou, "An intensive survey of fair non-repudiation protocols," *Computer Communications*, vol. 25, no. 17, pp. 1606–1621, 2002.

[15] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system," Technical report, Academic Press, Cambridge, MA, USA, 2008.

[16] H. Chen, H.-L. Wu, C.-C. Chang, and L.-S. Chen, "Light repository blockchain system with multisecret sharing for industrial big data," *Security and Communication Networks*, vol. 2019, Article ID 9060756, 7 pages, 2019.

[17] S. De Angelis, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, *PBFT vs. Proof-of-Authority: Applying the Cap Theorem to Permissioned Blockchain*, in *Proceedings of the Italian Conference on Cybersecurity*, Milan, Italy, June 2018.

[18] N. Szabo, "Smart contracts: building blocks for digital markets," *EXTROPY: The Journal of Transhumanist Thought*, vol. 18, no. 2, 1996.

[19] G. Wood, "Ethereum: a secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.

[20] E. Androulaki, A. Barger, V. Bortnikov et al., "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the 13th EuroSys Conference*, Porto, Portugal, 2018.

[21] C. Dannen, "Solidity programming," in *Introducing Ethereum and Solidity*, pp. 69–88, Springer, Berlin, Germany, 2017.

[22] R. Pike, *The GO Programming Language*, Addison-Wesley, Boston, MA, USA, 2009.

[23] S. Tilkov and S. Vinoski, "Node.js: using JavaScript to build high-performance network programs," *IEEE Internet Computing*, vol. 14, no. 6, pp. 80–83, 2010.

[24] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Proceedings of the 1987 Conference on the Theory and Application of Cryptographic Techniques*, Berlin, Germany, August 1987.

[25] M. S. Ferdous, A. Margheri, F. Paci, M. Yang, and V. Sassone, "Decentralised runtime monitoring for access control systems in cloud federations," in *Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, Atlanta, GA, USA, June 2017.

[26] Y. Xu, C. Zhang, Q. Zeng, G. Wang, J. Ren, and Y. Zhang, "Blockchain-enabled accountability mechanism against information leakage in vertical industry services," *IEEE Transactions on Network Science and Engineering*, 2020.

[27] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Secure multiparty computations on bitcoin," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, San Jose, CA, USA, May 2014.

[28] S. Dziembowski, L. Eckey, and S. Faust, "Fairswap: how to fairly exchange digital goods," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, Toronto, Canada, October 2018.

[29] Y. Xu, J. Ren, Y. Zhang, C. Zhang, B. Shen, and Y. Zhang, "Blockchain empowered arbitrable data auditing scheme for network storage as a service," *IEEE Transactions on Services Computing*, vol. 13, no. 2, pp. 289–300, 2019.

[30] M. N. Krohn, M. J. Freedman, and D. Mazieres, "On-the-fly verification of rateless erasure codes for efficient content distribution," in *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, Berkeley, CA, USA, May 2004.

[31] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, CRC Press, Boca Raton, FL, USA, 2nd edition, 2014.

[32] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Proceedings of the 1991 Annual International Cryptology Conference*, Santa Barbara, CA, USA, August 1991.

[33] H. Caliper, "Hyperledger caliper architecture," 2019, https://hyperledger.github.io/caliper/docs/2_Architecture.html.