# Fast Lottery-Based Micropayments for Decentralized Currencies

Kexin Hu(✉) and Zhenfeng Zhang

Laboratory of Trusted Computing and Information Assurance, Institute of Software,
Chinese Academy of Sciences, Beijing, China
{hukexin,zfzhang}@tca.iscas.ac.cn

**Abstract.** Transactions using the Bitcoin system, which is built atop a novel blockchain technology where miners run distributed consensus to ensure the security, will cause relatively high transaction costs to incentivize miners to behave honestly. Besides, a transaction should wait a quite long time (about 10 min on average) before being confirmed on the blockchain, which makes micropayments not cost-effective. In CCS'15, Pass and shelat proposed three novel micropayment schemes for any ledger-based transaction system, using the idea of *probabilistic payments* suggested by Wheeler (1996) and Rivest (1997), which are called as the "Lottery-based Micropayments". However, the one among the three schemes, which is fully compatible with the current Bitcoin system and only requires an "invisible" verifiable third party, needs two on-chain transactions during each execution, even if both the user and the merchant are honest. To reduce the transaction costs and increase efficiency, this paper proposes a fast lottery-based micropayment scheme to improve their work. By setting up a time-locked deposit, whose secure utilization is assured by the security of a primitive called accountable assertions under the discrete logarithm assumption, our scheme reduces the number of on-chain transactions to one, and yet maintains the original scheme's advantages.

**Keywords:** Lottery-based micropayments · Decentralized currencies
High efficiency · Accountable assertions

## 1 Introduction

Decentralized cryptocurrency systems based on and led by Bitcoin [1,7,12] have gained rapid popularity in recent years, and are often quoted as "a peek into the future financial and payment infrastructure". Although it is striking using the idea of maintaining a distributed ledger known as the *blockchain* to provide a decentralized and open platform, the blockchain is in its childhood and there's still room for improvement.

Since Satashi Nakamoto proposed Bitcoin [12] in 2008, plenty of work and debates about the Bitcoin system and the corresponding blockchain technology are springing up in various aspects, such as enhancing and analyzing their

W. Susilo and G. Yang (Eds.): ACISP 2018, LNCS 10946, pp. 669–686, 2018.
https://doi.org/10.1007/978-3-319-93638-3_38

security [13,17,20], increasing the efficiency [5,23], etc. We believe that besides the security, a currency system's durability also relies on the its efficiency and convenient level.

However, the inherent scalability insufficiency derived from Bitcoin blockchain protocol is still one of the main reasons that limit the widely adoption of Bitcoin-like currency systems. Bitcoin currently bears less than ten transactions per second, compared to the credit card that deals with 10,000 transactions per second, especially where Visa can achieve 47,000 peak transactions per second [24], Bitcoin's throughput capacity cannot bear real-world demand. As for the transaction latency, Bitcoin costs 10 min to create a new valid block on average. In general, only if a block has been backed up by at least six blocks, the transactions contained can be fully confirmed. This totally costs about one hour which makes the latency too long to suite for many practical payment scenarios, such as supermarkets, vending machines, and take-away stores. Another drawback of Bitcoin is the transaction size. On average, the size of a transaction is 500 bytes. A turnover of 500 transactions per second would require 10 TB of additional disk space per year, which is at the limit of a consumer's storage capacity [9].

Micropayments, i.e. payments of small amounts like cents or fractions of a cent, have relatively high transaction costs. Besides what we have discussed above, the transaction fee for a Bitcoin payment costs at least 0.0001 bitcoin corresponding to between 2.5 and 10 cents in the year of 2013 and 2014 [14], which would be higher considering the rise in value of Bitcoin in recent years [6]. The relative high transaction cost makes micropayments not cost-effective, where the transferred value is just a few cents. Unfortunately, micropayments have many practical applications. Imagine Alice wants to use wireless service that Bob provides. For every minute Alice used, she should pay Bob a relative small amount of money, and this payment happens in every few minutes. Other streaming services such as a user pays for every music he downloaded, every video he watched are all micropayment instances, and some of them can happen *very frequently*. This shows that, how to increase efficiency and reduce transaction costs are central issues when designing micropayment schemes for decentralized cryptocurrencies like Bitcoin.

One idea to overcome this problem is to reduce transaction volume arriving at the blockchain by batching multiple transactions into a larger one. The leading proposals are the micropayment channels [5] and its following work - the Lightning Network [16]. However, the approach of micropayment channels limits in the number of payment recipients to a *single predetermined* one for each channel. Although the Lightning Network can mitigate this restriction, there are several other drawbacks such as the worsening of Bitcoin's privacy weakness, the high bandwidth consumption and the massive storage needed.

Compared to the micropayment channels and the Lightning Network, another interesting proposal, proposed by Pass and shelat at CCS'15 [14], targeting at non-channel based peer-to-peer micropayments has a better performance in the number of payment recipients, meanwhile avoids the drawbacks of the Lightning

Network. Their work is based on the idea of *probabilistic payments* suggested by Wheeler [25] and Rivest [18]. The key idea of probabilistic payments is to batch several small transactions into a large transaction, and this process employs probabilistic "lottery-based" payments that instead of sending a transaction of value $v$, one can also send a lottery ticket whose *expected payout* is $v$. That is, between every $\eta$ transactions (e.g., $\eta = 100$), only one transaction will actually happen on average, and this transaction will pay $\eta$ times the amount of a transaction should pay. For every payment, it's just like issuing a "lottery ticket", which has a winning probability of $1/\eta$, from a user to a merchant. The advantage of this approach is that only the winning lottery ticket yields in a recorded transaction, but every (unopened) lottery ticket is a transfer of value.

Pass and shelat presented three micropayment schemes in their paper, and all of them can support making payments to *arbitrary* recipients. Compared with the first scheme whose implementation needs a modification to the existing Bitcoin script, the second scheme is Bitcoin-compatible by introducing a *verifiable* third party $\mathcal{T}$ to overcomes this barrier. All of $\mathcal{T}$'s operations can be publicly verified by irrefutable evidences (i.e., unforgeable signatures) and it can be legally punished or replaced if anyone catches $\mathcal{T}$'s cheating. In reality, this party $\mathcal{T}$ can be instantiated by a currency Exchange [10] in which clients deposit their money for a better user experience, or any company/organization that regards its reputation as a central concern. For a currency Exchange, its reputation is the key factor for gaining clients' trust and collecting money from clients. Hence, no such company/organization will risk of losing its reputation. Therefore, we believe that the usage of $\mathcal{T}$ is feasible.

The third scheme inherits the advantages of the former two, furthermore, it only requires the intervention of an "invisible" verifiable third party $\mathcal{T}$. Namely, $\mathcal{T}$ is not involved in payment executions when both sides of payments are honest. However, to prevent a user's cheating by refusing to pay for the services/goods he has enjoyed, which can break the financial fairness to the merchant, the third scheme needs two on-chain transactions during each payment execution *even if both sides of a payment are honest.* Considering the scalability limit of Bitcoin blockchain and the frequent uses of micropayments, we believe that every on-chain transaction's cutting down is meaningful.

It's not trivial to design a *secure* lottery-based micropayment scheme *for decentralized currencies* to achieve this goal, meanwhile maintain the prior scheme's advantages. The basic security requirement is to prevent double spending, where users gain additional utilities by issuing the same lottery ticket to several merchants that multiple of them might win but only one will be able to cash in his ticket. Another security property, which is important for people to adopt the scheme in reality, is the financial fairness. It says, neither side of a payment will loss more than it deserves due to a malicious behavior of the other side.

**Contributions.** In this paper we propose a fast lottery-based micropayment scheme for decentralized currencies like Bitcoin. Our scheme uses a set of technologies as well as some primitives to construct a micropayment scheme focusing on enhancing the third scheme in [14], in reducing transaction costs and increasing efficiency. By use of a deposit mechanism which is supported by a primitive called the accountable assertions, our scheme only requires one on-chain transaction during each execution when both sides of payments are honest, compared to the two on-chain transactions needed in the prior scheme, and our scheme can still ensure financial fairness. The secure utilization of the deposit is assured by the security of the accountable assertions which can prevent double-spending attack as well as limit the value needed in the deposit. We define two security properties that should at least be satisfied by a lottery-based micropayment scheme for decentralized currencies, called *double-spending determent* and *financial fairness*, and we prove our scheme can achieve these properties. Furthermore, we give a performance comparison among some related micropayment schemes. More specifically, our micropayment scheme has the following advantages:

- It reduces the number of on-chain transactions during a payment's execution into one without breaking the security of micropayments. Compared to the prior scheme, our proposal increases efficiency, decreases the latency by half and reduces the average micropayment transaction fees, which we believe it's meaningful especially when micropayments are conducted frequently.
- It only requires an "invisible" verifiable third party $\mathcal{T}$ in the process of payments. When both participants honestly follow their instructions, $\mathcal{T}$ is not involved in a payment.
- It can support for making micropayments to *arbitrary* recipients.
- It attains financial fairness, especially to merchants, i.e., merchants can get what he deserves even if users has cheat.
- Our scheme is Bitcoin-compatible. It needs no modification to the existing Bitcoin script when adopting this scheme to make micropayments.

**Paper Organization.** We start with the preliminary on the fundamental of the Bitcoin system and a description of the accountable assertions, also the security assumptions and standard cryptographic building blocks are concerned in this section. In Sect. 3, we define some security requirements and briefly review the third scheme at CCS'15, then we present our proposal with a security analysis. Section 4 shows a performance comparison. Section 5 concludes the paper.

# 2 Preliminary

## 2.1 Background on Bitcoin

Like most cryptocurrencies, Bitcoin is a digital cryptographic currency built atop a decentralized peer-to-peer network, the blockchain. Every transaction published to the Bitcoin network can be verified according to some rules called *release conditions*, which are realized by the Bitcoin script. Transactions are posted to the blockchain within a block after solving a Proof-of-Work (PoW) puzzle. This work is done by nodes from the Bitcoin network called *miners*. The block-contained transactions are publicly readable and verifiable. Once a block is added into the blockchain, especially backed up by a few blocks, like six, as its successors, it is very hard to be modified or deleted. We normally regard blockchain as an append-only decentralized public ledger.

**Transaction.** Before making transactions in the Bitcoin network, a user generates at least one Bitcoin account with a ECDSA key pair $(pk, sk)$ and an (pseudonymous) address. Every user is identified by his addresses[1]. Informally, a Bitcoin transaction is to transfer bitcoins from input addresses to output addresses using valid signatures, w.r.t. input addresses public keys respectively, to satisfy the predefined release condition. In the following, we use a triple $(a, a', v)$ to indicate a transaction transferring $v$ bitcoins from address $a$ to address $a'$, which is simplified as $(a, a')$ to indicate transferring all bitcoins in $a$ to $a'$.

We denote a Bitcoin address as $a = (pk, \sqcap)$, where $pk$ is $a$'s public key, and $\sqcap$ is $a$'s release condition. A release condition can be simply seen as a script that contains a sequence of instructions, which limits the redemption of an account. We regard $\sqcap$ as a predicate function that returns 0/1. If a user wants to withdraw $v$ bitcoins in address $a$ to some other address $a'$, he should present a witness $x$ to satisfy $a$'s release condition, such that $\sqcap(x, (a, a', v)) = 1$. In most cases, $x$ is a signature on the transaction $(a, a', v)$ w.r.t. $a$'s public key.

**Bitcoin Script.** The Bitcoin scripting language ("Bitcoin script" for short) is not Turing-complete. To support the basic functionalities a transaction needed, Bitcoin includes a list of script instructions [4]. Besides some fundamental ones, one of the most popular and practical instructions is `OP_CHECKLOCKTIMEVERIFY` [3], which is used for locking an address until some predetermined point in the future. We explain the idea behind this instruction.

*Lock Time.* The lock time mechanism is to allow a transaction output to be made unspendable until some predetermined time $T$ in the future. For a time-locked account, if the current time $t < T$, then the evaluation fails and a transaction with this account as input is consequently invalid. Only when $t \geq T$, the transactions involved can pass the verification and the funds covered are spendable.

A time-locked account usually serves as a deposit to prevent malicious behaviour. For example, before the locked time $T$, an account $a$ can only be redeemed by a witness generated by some specific parties, such as someone trusted by both sides of a payment. However, after $T$, $a$ is free and the money inside can be transferred using a signature w.r.t. $a$'s public key. This can be very helpful to a scheme to realize financial fairness.

[1] In this paper, we use the terms "address" and "account" interchangeably.

## 2.2 Accountable Assertions

Our construction uses a cryptographic primitive called accountable assertions. This primitive was first proposed by Ruffing et al. [19]. Intuitively, it allows users to assert statements to contexts for no more than a fixed number. Whenever a user asserts two distinct statements $st_1 \neq st_2$ to the same context $ct$, the private key of the user can be extracted *publicly.*

The authors gave out a concrete construction built upon the idea of chameleon authentication trees [11,21,22]. We describe the accountable assertions using the following algorithms:

- $\texttt{Setup}(\lambda) \rightarrow params$. This algorithm chooses a secure elliptic curve and a base point $g$ of prime order $q$ ($|q| \geq 2\lambda$ bites), where $\lambda$ is a security parameter. Let $l$ and $n$ be positive integers defining the depth of a tree and its branching factor. It outputs $(g, q, l, n)$ as *params.*
- $\texttt{KeyGen}(params) \rightarrow (pk, sk, auxsk)$. The key generation algorithm chooses a key $k \leftarrow \{0,1\}^\lambda$ for a pseudo-random function $\mathsf{F}_k$, and a random integer $\alpha \in \mathrm{Z}_q^*$ to generate a key pair $(pk', sk') = (X, \alpha)$ with $X = g^\alpha$. Compute the root node as $x_i^1 = \mathsf{F}_k(p, i, 0), r_i^1 = \mathsf{F}_k(p, i, 1)$, where $p$ is a unique identifier for the position of the root node, and $y_i^1 = g^{x_i^1} X^{r_i^1}$ for $i \in \{1, ..., n\}$ and set $z = \mathsf{H}(y_1^1, ..., y_n^1)$, where $\mathsf{H}$ is a collision-resistant hash function. Finally, it sets $pk := (pk', z), sk := sk', auxsk := k$.
- $\texttt{Assert}(sk, auxsk, ct, st) \rightarrow \tau$. Each node $Y_j = (y_1^j, ..., y_n^j)$ stores $n$ entries, and $a_j \in \{1, ..., n\}$ defines the position in the node. Let $Y_l$ represents the leaf stores the entry with the number $ct$, where $ct \in \{1, ..., n^l\}$, counted across all leaves from left to right, and $a_l$ is the position of this entry within $Y_l$. In the following, let $x_i^j = \mathsf{F}_k(p_j, i, 0), r_i^j = \mathsf{F}_k(p_j, i, 1)$, where $p_j$ is a unique identifier of the position of the node $Y_j$.
  - Compute $Y_l$: Assert statement $st$ to $ct$ by computing ${r'}_{a_l}^l = \alpha^{-1}(x_{a_l}^l - \mathsf{S}(st)) + r_{a_l}^l \pmod q$, where $\mathsf{S}$ is a hash function modeled as a random oracle. Observe that $g^{x_{a_l}^l} X^{r_{a_l}^l} = y_{a_l}^l = g^{\mathsf{S}(st)} X^r$. For $i \in \{1, ..., n\} \backslash \{a_l\}$, $y_i^l = g^{x_i^l} X^{r_i^l}$. Let $z_{l-1} = \mathsf{H}(y_1^l, ..., y_n^l)$ and let further $f_l = (y_1^l, ..., y_{a_l-1}^l, y_{a_l+1}^l, ..., y_n^l)$.
  - Compute the nodes up to the root for $h = l-1, ..., 1$: Assert $z_h$ with respect to $Y_h$ by computing ${r'}_{a_h}^h = \alpha^{-1}(x_{a_h}^h - z_h) + r_{a_h}^h \pmod q$. Observe that $g^{x_{a_h}^h} X^{r_{a_h}^h} = y_{a_h}^h = g^{z_h} X^{{r'}_{a_h}^h}$. For $i \in \{1, ..., n\} \backslash \{a_h\}$, $y_i^h = g^{x_i^h} X^{r_i^h}$. Let $z_{h-1} = \mathsf{H}(y_1^h, ..., y_n^h)$ and let further $f_h = (y_1^h, ..., y_{a_h-1}^h, y_{a_h+1}^h, ..., y_n^h)$.

  Finally, it outputs the assertion $\tau := ({r'}_{a_l}^l, f_l, a_l, ..., {r'}_1^1, f_1, a_1)$.
- $\texttt{Verify}(pk, ct, st, \tau) \rightarrow b$. The verification algorithm parses $pk$ as $(pk', z)$, and $\tau$ as $({r'}_{a_l}^l, f_l, a_l, ..., {r'}_1^1, f_1, a_1)$. It first verifies that $pk'$ is a valid public key, then it checks the validity of a statement $st$ in a context $ct$ by reconstructing a path including nodes $(Y_l, Y_{l-1}, ..., Y_1)$ from a leaf $Y_l$ to the root $Y_1$, and verifies whether $H(y_1^1, ..., y_n^1) = z$. If any of the above verifications fails, this algorithm outputs 0. Otherwise, it outputs 1.

- $\texttt{Extract}(pk, ct, st_1, \tau_1, st_2, \tau_2) \rightarrow sk/\perp$. The extraction algorithm computes, like the verification algorithm, the assertion paths for both $st_1$ and $st_2$ from the bottom up to the root until a position in the tree is found where the two paths form a collision, i.e., a position in the tree where values $(x_1, r_1)$ are used in the assertion path of $st_1$ and values $(x_2, r_2)$ are used in the assertion path of $st_2$ such that $g^{x_1}X^{r_1} = g^{x_2}X^{r_2}$. Then this algorithm outputs $sk = (x_1 - x_2)/(r_2 - r_1) \pmod q$. If no such position is found, it fails.

We illustrate with Fig. 1. Assume the context $ct$ we would like to assert a statement maps to the node entry $y_2^3$. Node entries written in gray background constitute an assertion path for statement $st$ from a leaf to the root. In this example, the assertion is $\tau = ({r'}_2^3, f_3, 2, {r'}_3^2, f_2, 3, {r'}_2^1, f_1, 2)$.

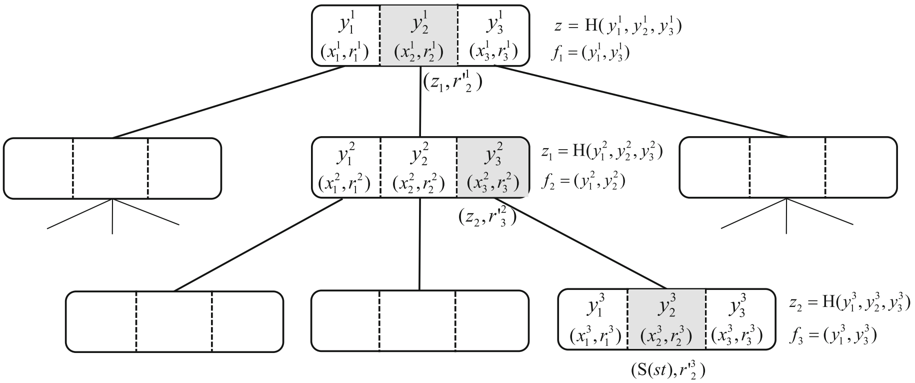

**Fig. 1.** A tree with a specific assertion path, where $l = 3$ and $n = 3$.

Ruffing, Kate and Schröder showed that the accountable assertions satisfy completeness, and security properties of extractability and secrecy under the discrete logarithm assumption. Informally, *extractability* states whenever two different statements have been asserted to the same context, the unique private key can be extracted except for a negligible probability. Opposed to extractability, *secrecy* states if no equivocation happens, i.e., there is a unique statement $st$ for each context $ct$, the private key cannot be extracted except for a negligible probability.

## 2.3 Assumptions and Building Blocks

In this part, we present assumptions and a few more building blocks involved in our scheme.

**Assumptions.** In this paper, we regard the blockchain as a public transaction ledger who runs a consensus protocol among miners to agree on a global state. We make the following assumptions:

- *Correctness and availability.* We assume the blockchain will compute correctly following the predefined instructions, and the blockchain is always available.
- *Public state.* All nodes can see the state of the blockchain at any time, i.e., transactions in the blockchain is public. The blockchain can be seen as a public transaction ledger.
- *Time.* The blockchain embodies a discrete clock. Time increases in rounds. The lock time mechanism, described in Sect. 2.1, relies on this discrete clock to make a decision on the validity of a transaction. Time can be aware by all nodes in the system.
- *Message delivery.* Messages will arrive at the blockchain at the beginning of the next round. This makes the confirmation of any transaction costs a period of time. An adversary has the ability to arbitrarily reorder messages sent to the blockchain within a round, which makes the adversary may attempt a front-running attack (a.k.a. rushing adversary).
- *Verifiable third party.* We assume, there exists a party that can be *partially* trusted in the system. All its behavior is verifiable by irrefutable evidence (i.e., an unforgeable signature) and it can be punished or replaced if being caught of "cheating". In reality, this party can be a currency Exchange, whose reputation is the key factor for collecting money from clients.

**Building Blocks.** In addition to the accountable assertions, our construction uses some standard cryptographic tools as follows.

*Commitment Schemes.* A (non-interactive) commitment scheme $\mathsf{COMM}$ enables a party to generate a commitment to a given message. We call a commitment scheme $\mathsf{COMM}$ secure if it satisfies security properties of hiding and binding. Informally, *hiding* states a commitment does not reveal the committed value, and *binding* states a commitment cannot be opened to two different values, i.e., it is computationally (or statistically) infeasible to find $(r, s, r', s')$, such that, $r \neq r'$ but $\mathsf{COMM}_s(r) = \mathsf{COMM}_{s'}(r')$.

*Signature Schemes.* Our protocol uses signature scheme $(\mathtt{Gen}, \mathtt{Sig}, \mathtt{Vrf})$ that is existentially unforgeable under adaptively chosen-message attacks (EUF-CMA) for all PPT adversaries. Informally, it states that any PPT adversary that is given the public key of the signature scheme and can query to the signing oracle for signatures of polynomial-time messages, the adversary still cannot output a valid pair of signature and the signed message that hasn't be queried before.

## 3 Fast Lottery-Based Micropayments

In this section, we provide our fast lottery-based micropayment protocol based on the prior scheme proposed by Pass and shelat in CCS'15 [14]. First, we define two important security requirements for lottery-based micropayments for decentralized currencies. Second, we briefly review the prior scheme. Finally, we present our protocol and give a security analysis.

### 3.1 Security Definitions

We define the following security definitions that we believe should at least be satisfied by lottery-based payments between users and merchants for decentralized currencies.

**Definition 1 (Double-Spending Determent).** *This property requires that a malicious user cannot produce two valid spending evidences for different payments that share the same serial number without being detected or punished.*

**Definition 2 (Financial Fairness).** *This property requires that in a lottery-based payment between a user and a merchant, the user should pay exactly the same amount of money according to the result of the lottery ticket and his behavior, i.e., whether he cheats, and the merchant can get what he deserves from the user.*

### 3.2 A Brief Review of the Scheme in CCS'15

In the third scheme of [14], the user $U$ and the merchant $M$ jointly generate a lottery ticket used to decide whether the merchant should be paid in this payment by invoking a coin-tossing protocol. Only if the lottery ticket wins, $U$ should pay $\eta$ times the amount of every transaction should pay (suppose the winning probability of the lottery ticket is $1/\eta$). To prevent a malicious $U$'s cheating by withdrawing the money in the account that are supposed to be used to pay $M$, or using the same account to conduct payments with someone else (i.e., double spending), which both can lead to $M$'s loss, this scheme was designed to send $M$ a credential before $M$ tells $U$ the ticket result. With this credential, $M$ can transfer bitcoins from $U$'s account $a$ to a special account $a'$. The specialty of $a'$ relies on its release condition. It limits the recipient account to be either $a^U$, which is fully controlled by $U$, or $a^M$ that belongs to $M$. Furthermore, the release condition of $a'$ demands a 2-of-3 multi-signature from the set $\{\sigma_U, \sigma_M, \sigma_{\mathcal{T}}\}$ which are signed by $U, M, \mathcal{T}$ respectively. Therefore, $U$ cannot withdraw/transfer the money in account $a'$ without the help of $M$ or $\mathcal{T}$. If $U$ and $M$ are honest, they can complete a payment by themselves. However, if $U$ cheats, $M$ will not help $U$ otherwise he will suffer a loss, and $\mathcal{T}$ will not help $U$ if he suspects $U$'s honesty. In this case, $M$ can ask $\mathcal{T}$'s help to get the money he deserves from account $a'$. If $M$ cheats about his winning and maliciously locks $U$'s money into account $a'$, $U$ can still withdraw his money with the help of $\mathcal{T}$ by providing a multi-signature of $(\sigma_U, \sigma_{\mathcal{T}})$ to satisfy the release condition of $a'$. Thus, this scheme can ensure financial fairness both for the user and the merchant as well as preventing double-spending attack.

### 3.3 Our Protocol

We proceed to present the construction of our protocol, shown in Fig. 2. Similar to the prior scheme, our protocol requires the intervention of an "invisible" verifiable third party $\mathcal{T}$, which is involved in an execution only when participants deviate from their prescribed instructions. All transaction-related instructions described in our protocol can be conducted using the existing Bitcoin script, thus, our protocol is Bitcoin-compatible.

We adopt two accounts to prevent potential attacks from $U$. If $U$ and $M$ are honest, an execution only involves the first account to transfer an amount of money when the lottery ticket wins. If, however, a malicious $U$ refuses to pay, $M$ can be compensated using the second account with $\mathcal{T}$'s help before a expiry time $T$. The second account is a locked deposit of $U$ with a expiry time $T$. If $M$ can present a valid evidence showing $U$ has cheated before time $T$, $\mathcal{T}$ will transfer the same amount of value the merchant deserves from the second account, together with a public verifiable evidence showing that $\mathcal{T}$'s computation is correct.

Next, we concern more details. The front-running attack should be prevented where a malicious user wants to avoid losses by withdrawing his deposit before $M$ being compensated. We set a period of time $T'$ as a safety margin to enable a successful transfer on the blockchain, e.g., $T' = 10\,\text{min}$. Thus, when $M$ receives a ticket from $U$, he should check whether the current time satisfies $t < T - T'$. If it fails, $M$ should reject and abort. Also, to prevent $U$'s double-spending attack by conducting multiple payments concurrently which makes a deposit insufficient to pay all victims, we adopt the accountable assertions to limit the number of payments a deposit can be linked. A malicious user will lose all his money in the deposit if he ever initiates even one double-spending payment. Moreover, we set a period of time $\bar{T}$ as a safety margin to protect $U$'s asset when $M$ is malicious and wants to double his income by showing an evidence to $\mathcal{T}$ that $U$ has not published a transaction, and after that telling $U$ that he wins the lottery. $U$ will not do any operation if the current time goes out of scope as defined. Finally, we expect the protocol can be conducted without the intervention of $\mathcal{T}$ when $U$ and $M$ are honest, thus, $U$ should have the ability to withdraw his deposit (i.e., money in the second account) by himself after time $T$.

Money in the first account $a = (pk_a, \sqcap^a)$ gets released if $U$ agrees to a transaction to $M$. Money in the second account $a^{dep} = (pk_d, \sqcap^d)$ can only be released if either (a) both $U$ and $\mathcal{T}$ agree to a transaction before a certain time $T$, or (b) $U$ agrees to a transaction after time $T$. We implement it using the lock time mechanism. Let $\sqcap^a(x, (a, a')) = 1$ if and only if $x = \sigma_1$ is a signature of the transaction $(a, a')$ w.r.t. $pk_a$. This can be implemented with a standard release condition. Define $\sqcap^d(x', (a^{dep}, a'', v)) = 1$ if and only if either (a) before time $T$, $x'$ contains a signature $\sigma_{\mathcal{T}}$ of the transaction $(a^{dep}, a'', v)$ w.r.t. $pk_{\mathcal{T}}$, where $v$ denotes the value being transferred, or (b) after time $T$, $x'$ is a signature $\sigma'_U$ of the transaction $(a^{dep}, a'', v)$ w.r.t. $pk_d$. In the optimistic case when $U$ and $M$ honestly follow their instructions, $\mathcal{T}$ will not involve into an execution. If $M$ finds out that $U$ deviates from his instructions before time $T$, i.e., $U$ refuses to transfer the money to $M$, $M$ and $\mathcal{T}$ can present a valid witness to release the same amount of money from $a^{dep}$, and showing an evidence of $\mathcal{T}$'s honesty.

Let "[·]" be an operation when inputting a random string, it outputs 1 with a probability of $1/\eta$, or 0 otherwise. For example, if the last two digits of the input random string are 00, it outputs 1 and this happens with probability 1/100.

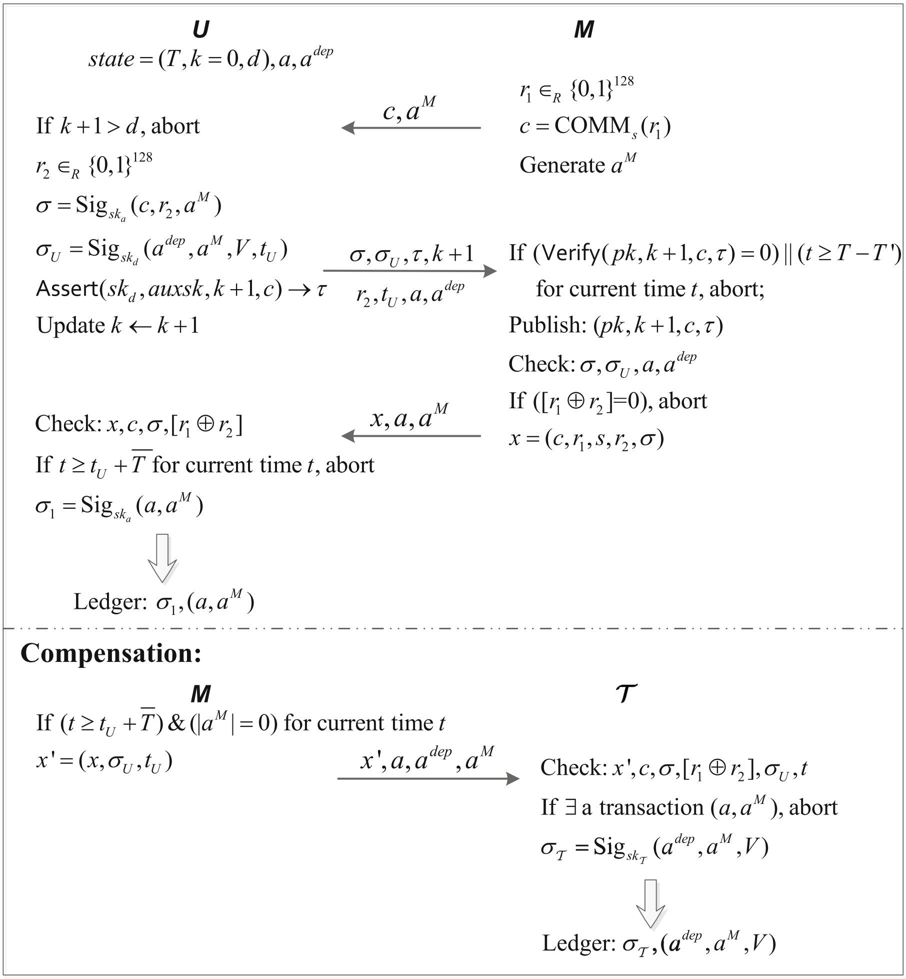
$U$
$M$
$state = (T, k = 0, d), a, a^{dep}$
$r_1 \in_R \{0,1\}^{128}$
$c = \mathrm{COMM}_s(r_1)$
Generate $a^M$
$c, a^M$
If $k + 1 > d$, abort
$r_2 \in_R \{0,1\}^{128}$
$\sigma = \mathrm{Sig}_{sk_a}(c, r_2, a^M)$
$\sigma_U = \mathrm{Sig}_{sk_d}(a^{dep}, a^M, V, t_U)$
$\mathsf{Assert}(sk_d, auxsk, k+1, c) \rightarrow \tau$
Update $k \leftarrow k + 1$
$\sigma, \sigma_U, \tau, k+1$
$r_2, t_U, a, a^{dep}$
If $(\mathsf{Verify}(pk, k+1, c, \tau) = 0) \,||\, (t \geq T - T')$
for current time $t$, abort;
Publish: $(pk, k+1, c, \tau)$
Check: $\sigma, \sigma_U, a, a^{dep}$
If $([r_1 \oplus r_2] = 0)$, abort
$x = (c, r_1, s, r_2, \sigma)$
$x, a, a^M$
Check: $x, c, \sigma, [r_1 \oplus r_2]$
If $t \geq t_U + \overline{T}$ for current time $t$, abort
$\sigma_1 = \mathrm{Sig}_{sk_a}(a, a^M)$
Ledger: $\sigma_1, (a, a^M)$
Compensation:
$M$
$\mathcal{T}$
If $(t \geq t_U + \overline{T})$ & $(|a^M| = 0)$ for current time $t$
$x' = (x, \sigma_U, t_U)$
$x', a, a^{dep}, a^M$
Check: $x', c, \sigma, [r_1 \oplus r_2], \sigma_U, t$
If $\exists$ a transaction $(a, a^M)$, abort
$\sigma_{\mathcal{T}} = \mathrm{Sig}_{sk_{\mathcal{T}}}(a^{dep}, a^M, V)$
Ledger: $\sigma_{\mathcal{T}}, (a^{dep}, a^M, V)$

**Fig. 2.** Fast "Lottery-based" micropayments.

**Set Up:** A user $U$ generates a Bitcoin key pair $(pk_a, sk_a)$ and transfers $V = \eta \cdot v$ bitcoins to an address $a = (pk_a, \Pi^a)$. $U$ generates another Bitcoin key pair $(pk_d, sk_d)$ together with accountable assertions keys $(pk, sk = sk_d, auxsk)$, and transfers $(dV + p)$ bitcoins to an address $a^{dep} = (pk_d, \Pi^d)$ with a expiry time $T$, where $p$ is the penalty if $U$ double spends. $U$ keeps an initial deposit state

$state := (T, k = 0, d)$, where $k$ is a counter and $d$ is the number of payments this deposit can be involved.

**Request:** Whenever $M$ wants to request a payment of $v$ bitcoins from $U$, he picks a random number $r_1 \leftarrow \{0,1\}^{128}$, generates a commitment $c = \mathsf{COMM}_s(r_1)$ where $s$ denotes the string used to open the commitment. $M$ then generates an address $a^M$ for receiving bitcoins during this payment, and sends $(c, a^M)$ to $U$.

**Issuance:** To send a probabilistic payment of amount $v$, $U$ first checks the serial number of this payment (i.e., the counter $k+1$) that not exceeds the predetermined upper bound $d$. After that, he picks a random number $r_2$ and creates two signatures. The first signature $\sigma$ on $(c, r_2, a^M)$ is w.r.t. $pk_a$, and the second signature $\sigma_U$ on $(a^{dep}, a^M, V, t_U)$ is w.r.t. $pk_d$, where $t_U$ is the current time. Then, $U$ creates an assertion $\tau \leftarrow \mathtt{Assert}(sk_d, auxsk, k+1, c)$ where $k+1$ denotes the serial number of the current payment. Next, $U$ increases the $k$ recorded in $state$ by 1 to indicate that one more payment has been made, and sends $(\sigma, \sigma_U, \tau, k+1, r_2, t_U, a, a^{dep})$ to $M$.

**Judgment:** On receiving a message from $U$, $M$ does the following operations:

(1) Verify whether the assertion $\tau$ is valid, i.e. $\mathtt{Verify}(pk, k+1, c, \tau) = 1$, and the current time $t < T - T'$ where $T'$ is a period of time sufficient for a transaction being confirmed on the blockchain;
(2) Publish the transcript $(pk, k+1, c, \tau)$ on a bulletin board;
(3) Check whether $\sigma, \sigma_U, a, a^{dep}$ are valid: verify signatures $\sigma, \sigma_U$ with $pk_a$ and $pk_d$ respectively. Check whether there is enough money in account $a$ and $a^{dep}$, and both of them are spendable;
(4) Check whether $[r_1 \bigoplus r_2] = 1$.

If all of the above conditions hold, $M$ sends $U$ a tuple $(x, a, a^M)$ such that $x = (c, r_1, s, r_2, \sigma)$, $c = \mathsf{COMM}_s(r_1)$, $\sigma$ is the signature received from $U$, and $[r_1 \bigoplus r_2] = 1$. On receiving the message sent by $M$, $U$ checks the validity of these conditions and verifies whether the current time $t < t_U + \bar{T}$. Next, $U$ computes a signature $\sigma_1$ on $(a, a^M)$ w.r.t. $pk_a$ and publishes a transaction from account $a$ to account $a^M$ with value $V$ to the ledger (i.e., the blockchain), using $\sigma_1$ as a witness to satisfy the release condition $\Pi^a$. If $U$ hasn't published this transaction until time $t_U + \bar{T}$, $M$ immediately invokes the Compensation procedure.

**Compensation:** When $\mathcal{T}$ receives a tuple $(x', a, a^{dep}, a^M)$ such that $x' = (x, \sigma_U, t_U)$, $c = \mathsf{COMM}_s(r_1)$, $\sigma$ is a valid signature on $(c, r_2, a^M)$ w.r.t. $pk_a$, $[r_1 \bigoplus r_2] = 1$, $\sigma_U$ is a valid signature on $(a^{dep}, a^M, V, t_U)$ w.r.t. $pk_d$, and the current time $t$ satisfies $t_U + \bar{T} \leq t < T$, $\mathcal{T}$ checks if there is a transaction from address $a$ to $a^M$ in the Bitcoin network or on the ledger. If not, $\mathcal{T}$ signs $(a^{dep}, a^M, V)$ w.r.t. $pk_{\mathcal{T}}$, and publishes a transaction from account $a^{dep}$ to account $a^M$ with value $V$ to the ledger, using $\sigma_{\mathcal{T}}$ as the witness to satisfy the release condition $\Pi^d$ before time $T$. $\mathcal{T}$ also publishes an evidence related to this payment showing its honesty.

**Penalty:** If there exists two assertions $(c, \tau)$ and $(c', \tau')$ that corresponding to the same $(pk, k)$, anyone, including $\mathcal{T}$, can *immediately* extract $sk(= sk_d)$. Before $U$ can withdraw the money in $a^{dep}$, which is only allowed after time $T$, $\mathcal{T}$ can take out all bitcoins in $a^{dep}$ by publishing a signature w.r.t. $pk_d$ as an evidence that $U$ has cheated.

After the expiry time $T$, $U$ is free to withdraw the remaining money in the account $a^{dep}$ with a signature w.r.t. $pk_d$. Even if there are several merchants contacting $\mathcal{T}$ during the period of $T$, as long as $\mathcal{T}$ has settled these disputes *before time* $T$, the honest merchants can be compensated. It is convenient for $\mathcal{T}$ to merge all honest requests into one larger transaction and only release this transaction to the ledger.

### 3.4 Security Analysis

We present three theorems to state that our lottery-based micropayment protocol can achieve the security properties defined in Sect. 3.1, and we defer the proofs to appendix A.

**Theorem 1.** *If* $\mathsf{COMM}$ *is a secure commitment scheme, and the signature scheme* $(\mathtt{Gen}, \mathtt{Sig}, \mathtt{Vrf})$ *is existentially unforgeable under adaptively chosen-message attacks (EUF-CMA), then the probability that an execution of the proposed lottery-based micropayment protocol results in an transaction on the blockchain is exactly* $1/\eta$.

**Theorem 2.** *If accountable assertions* $(\mathtt{Setup}, \mathtt{KeyGen}, \mathtt{Assert}, \mathtt{Verify}, \mathtt{Extract})$ *is extractable, and* $\mathsf{COMM}$ *is a secure commitment scheme, then the proposed lottery-based micropayment protocol is double-spending deterrable.*

**Theorem 3.** *If* $\mathsf{COMM}$ *is a secure commitment scheme, signature scheme* $(\mathtt{Gen}, \mathtt{Sig}, \mathtt{Vrf})$ *is EUF-CMA, and accountable assertions* $(\mathtt{Setup}, \mathtt{KeyGen}, \mathtt{Assert}, \mathtt{Verify}, \mathtt{Extract})$ *satisfy extractability and secrecy, then the proposed lottery-based micropayment protocol can achieve financial fairness.*

## 4 Performance Comparison

In this section, we give a brief comparison of the performances among our protocol, Pass and shelat's scheme (i.e., the third one) presented at CCS'15 [14], the full version [15] of CCS'15, and a lottery-based micropayment protocol on Zerocash [20]. In Table 1, we measure the performances of the four schemes with the transaction number needed, the financial fairness property, and the communication and computation costs both on the user and the merchant sides.

**Table 1.** Performance comparison

| Ref. | Transaction number[a] | Financial fairness | Communication (in rounds)[a] | User[b] | Merchant[b] |
|---|---|---|---|---|---|
| CCS'15 | 1 off-line[c]<br>2 on-chain | Yes | 4 | 6·exp | 5·exp |
| The full version [15] | 2 off-line<br>1 on-chain | No | 3 | 7·exp | 6·exp |
| DAM | 2 off-line<br>1 on-chain | No | 3 | 10·exp+△[d] | 10·exp+⊙[d] |
| Ours | 2 off-line<br>1 on-chain | Yes | 3 | 6·exp | 10·exp |

[a] We only measure the number of transactions (in the 2rd column) and the rounds (in the 4th column) that a *winning* payment needed between *honest* parties.

[b] In comparison of computation cost, we only take into account of the expensive operations that a *winning* payment needed between *honest* parties, where "exp" denotes an exponentiation operation in group $\mathbb{G}$.

[c] We divide the transactions involved in a protocol into two categories: off-line and on-chain, where "off-line" indicates a transaction that can be computed on the fly before an execution of payments.

[d] "△": a Zerocash Pour operation + a NIZK prove operation + 4 (non-)membership prove operation. "⊙": a NIZK verify operation.

Our protocol requires 1 on-chain transaction during each winning micropayment for transferring money from $U$ to $M$ directly. Besides, $U$ creates another 2 off-line transactions, which can be conducted on the fly, to transfer money to $U$'s paying account $a$ and deposit account $a^{dep}$ separately.

In the version of Pass and shelat's third scheme at CCS'15, it requires 1 off-line transaction, which transfers money to $U$'s paying account (i.e. account $a$ in Sect. 3.2), and 2 on-chain transactions. The first on-chain transaction transfers money from account $a$ to a frozen account $a'$, and the second one transfers money from $a'$ to $M$. Compared to ours, the total number of transactions seems to remain unchanged, however, one of our off-line transactions, related to $a^{dep}$, can support for several payments. Considering micropayments can happen frequently, our protocol reduces the average micropayment transaction fees.

The full version [15] improves the third scheme to involve only 1 on-chain transaction using the multi-signature [2]. Besides 1 off-line transaction for transferring money to $U$'s paying account, it needs an extra deposit account that uniquely binds to a lottery ticket (i.e. a paying account) and burns on an evidence of double-spending to deal with the front-running/parallel attack. The size of the deposit requires to be large enough but is unspecified in the paper, and the uniquely binding limits the number of tickets a user can validly create due to the requirement of the size of a binding deposit, thus narrows the number of services a user can enjoy concurrently. Our protocol enables concurrent micropayments with a same deposit by the use of accountable assertions. This is a practical extension for applications of micropayments.

The DAM (Decentralized Anonymous Micropayment) scheme [8], proposed at EUROCRYPT'17, is a lottery-based micropayment scheme based on Zerocash [20]. To resolve the tension between the anonymity and double-spending determent, the DAM scheme involves a deposit account like ours and [15], and introduces plenty of primitives such as NIZK. However, similar to [15], the deposit burns on an evidence of double-spending. This makes a payment not financial fair, i.e., $M$ cannot be compensated if $U$ cheats. The computation cost of the DAM scheme is obviously higher than the other three schemes, and in Table 1 we only take out some very expensive operations in the DAM scheme as a whole in "$\triangle$" and "$\odot$".

## 5 Conclusion

Bitcoin, as well as many crypto-currencies based on the novel blockchain technology, has inherent scalability limits such as low capacity in transaction throughput, long transaction latency, large transaction size and relatively high transaction costs, which makes micropayments not cost-effective. This paper proposed a fast lottery-based micropayment scheme for decentralized currencies, especially for Bitcoin. It adopted the idea of probabilistic "lottery-based" micropayments, but further reduced transaction costs and increased efficiency of a prior scheme at CCS'15 by establishing a time-locked deposit account with the accountable assertions. As long as both sides of payments are honest, our scheme can be conducted without any third party's involvement and require at most one "on-chain" transaction during each execution. However, if a user or a merchant is malicious, our scheme can protect the counterparty's fund security with the help of a verifiable third party.

**Acknowledgements.** We thank the anonymous reviewers for their comments which helped to improve the paper. This work was supported by the National Key R&D Program of China (Nos. 2017YFB0802500, 2017YFB0802000), and the National Natural Science Foundation of China (No. U1536205).

## A Proofs for Theorems

**Theorem 1.** *If COMM is a secure commitment scheme, and signature scheme* $(\texttt{Gen}, \texttt{Sig}, \texttt{Vrf})$ *is existentially unforgeable under adaptively chosen-message attacks (EUF-CMA), then the probability that an execution of the proposed lottery-based micropayment protocol results in an transaction on the blockchain is exactly* $1/\eta$.

*Proof.* Suppose the user can bias the result and spend less than he ought to be, which means that after he receiving a commitment $c$ from the merchant, the user can select a $r_2$ satisfying $[r_1 \bigoplus r_2] = 0$. This equals to that the user can know $r_2$, the committed value of the commitment $c$, before the merchant opens $c$. This will break the hiding property of the commitment scheme.

Suppose the merchant can bias the result and earn more money, which means that he can either present a new $r_1'(\neq r_1)$ satisfying $([r_1' \bigoplus r_2] = 1) \wedge (c = \mathsf{COMM}_{s'}(r_1'))$ which breaks the binding property of the commitment scheme where $c$ is the commitment of $r_1$, or he can succeed by presenting a new pair $(r_2', \sigma')$ satisfying $([r_1 \bigoplus r_2'] = 1) \wedge (\mathtt{Vrf}_{pk_a}(\sigma', (c, r_2', a^M)) = 1)$, and this will break the existentially unforgeable of the signature scheme. □

**Theorem 2.** *If accountable assertions* $(\mathtt{Setup}, \mathtt{KeyGen}, \mathtt{Assert}, \mathtt{Verify}, \mathtt{Extr-act})$ *is extractable, and* $\mathsf{COMM}$ *is a secure commitment scheme, then the proposed lottery-based micropayment protocol is double-spending deterrable.*

*Proof.* Suppose an adversary $\mathcal{A}$ can break the double-spending determent of our lottery-based micropayment protocol, then he can produce at least two assertions $\tau$ and $\tau'$ with $\tau \leftarrow \mathtt{Assert}(sk_d, auxsk, k, c)$, $\tau' \leftarrow \mathtt{Assert}(sk_d, auxsk, k, c')$ and finish the corresponding payments without being caught, where $c$ and $c'$ belong to two different payments generated by the corresponding merchants, and $k$ denotes a serial number. When $c \neq c'$, this means that $\mathcal{A}$ can break the extractability of the accountable assertions. When $c = c'$, where $c = \mathsf{COMM}_s(r_1)$ and $c = \mathsf{COMM}_{s'}(r_1')$, according to the binding property of the commitment scheme, $(r_1, s) = (r_1', s')$. However, this happens with only a negligible probability when two merchants randomly choose the same pair $(r_1, s)$ which is used to ensure the merchants asset security. □

**Theorem 3.** *If* $\mathsf{COMM}$ *is a secure commitment scheme, signature scheme* $(\mathtt{Gen}, \mathtt{Sig}, \mathtt{Vrf})$ *is EUF-CMA, and accountable assertions* $(\mathtt{Setup}, \mathtt{KeyGen}, \mathtt{Assert}, \mathtt{Verify}, \mathtt{Extract})$ *satisfy extractability and secrecy, then the proposed lottery-based micropayment protocol can achieve financial fairness.*

*Proof.* Suppose a malicious merchant can break the financial fairness of the user by receiving more money than the user should pay for the payment. This means that the merchant can either (1) bias the result of the lottery ticket, or (2) transfer the money from $a$ to his account $a^M$ even if he loses the lottery ticket by forging a signature $\sigma = \mathtt{Sig}_{sk_a}(a, a^M)$, or (3) collect all published assertions related to the user and extract the private key of $a^{dep}$ then generate a valid signature, or (4) transfer the money from $a^{dep}$ to his account $a^M$ by forging a signature $\sigma = \mathtt{Sig}_{sk_{\mathcal{T}}}(a^{dep}, a^M, V)$ before time $T$, or (5) publish a signature $\sigma = \mathtt{Sig}_{sk_d}(a^{dep}, a^M, V)$ to withdraw the money in $a^{dep}$ after time $T$.

The condition (1) is infeasible due to our proof for Theorem 1. For the condition (2), (4) and (5), any one of them can break the existentially unforgeable of the signature scheme. The condition (3) is conflicting with the secrecy of the accountable assertions. Besides, a transaction published by the user in order to transfer money from $a$ and a transaction published by $\mathcal{T}$ in order to transfer money from $a^{dep}$ will not coexistent, due to the assumptions that the blockchain is available and public, and the discrete clock blockchain embodies makes the time in the system is synchronous.

Suppose a malicious user can break the financial fairness of the merchant by refusing to pay the merchant even the lottery ticket has won. The user may refuse to publish a transaction from $a$ to $a^M$, and $M$ cannot obtain the money he deserves from the deposit account $a^{dep}$ by himself. Remember that there exist a verifiable third party $\mathcal{T}$ whose operations should follow the instructions of the scheme. Thus, a merchant can ask $\mathcal{T}$'s help to obtain the money from the deposit account $a^{dep}$ when facing a malicious user. The user cannot withdraw the money in the locked account $a^{dep}$ before time $T$, otherwise it violates the assumption of the correctness of the blockchain. Although the deposit is unlocked after the time $T$ and the user can freely withdraw the money, the protocol limits that every request received by $M$ should be before the time $T - T'$ where it leaves a period of time $T'$ for $\mathcal{T}$ to handle a dispute.

It remains one more case that should be considered, i.e., the money in the deposit account $a^{dep}$ may be insufficient to compensate $M$. In this case, the user has conducted multiple (more than $d$) payments by issuing $n$ assertions $\{\tau_i\}_{i=1}^n$, where $n$ ($n > d$) is the number of payments the user conducts hoping that the deposit cannot afford all merchants compensation requests. As a result, there must be at least two assertions satisfying $(\tau_i \leftarrow \texttt{Assert}(sk_d, auxsk, k_i, c_i)) \wedge (\tau_j \leftarrow \texttt{Assert}(sk_d, auxsk, k_j, c_j)) \wedge (\texttt{Verify}(pk, k_i, c_i, \tau_i) = 1) \wedge (\texttt{Verify}(pk, k_j, c_j, \tau_j) = 1) \wedge (k_i = k_j)$. According the proof of Theorem 2, this can either break the extractability of the accountable assertions, or the binding property of the commitment scheme. □

## References

1. Barber, S., Boyen, X., Shi, E., Uzun, E.: Bitter to better—how to make bitcoin a better currency. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 399–414. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32946-3_29
2. Bitcoin Wiki: BIP 0010. https://en.bitcoin.it/wiki/Multisignature
3. Bitcoin Wiki: BIP 0065. https://en.bitcoin.it/wiki/BIP_0065
4. Bitcoin Wiki: Script. https://en.bitcoin.it/wiki/Script
5. Bitcoinj Project: Working with micropayment channels. https://bitcoinj.github.io/working-with-micropayments
6. BLOCKCHAIN: Blockchain Charts. https://blockchain.info/charts
7. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: SoK: research perspectives and challenges for Bitcoin and cryptocurrencies. In: 2015 IEEE Symposium on Security and Privacy, pp. 104–121. IEEE (2015)
8. Chiesa, A., Green, M., Liu, J., Miao, P., Miers, I., Mishra, P.: Decentralized anonymous micropayments. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017 Part II. LNCS, vol. 10211, pp. 609–642. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_21
9. Decker, C., Wattenhofer, R.: A fast and scalable payment network with bitcoin duplex micropayment channels. In: Pelc, A., Schwarzmann, A.A. (eds.) SSS 2015. LNCS, vol. 9212, pp. 3–18. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21741-3_1
10. INVESTOPEDIA: Bitcoin Exchange. https://www.investopedia.com/terms/b/bitcoin-exchange.asp
11. Krupp, J., Schröder, D., Simkin, M., Fiore, D., Ateniese, G., Nuernberger, S.: Nearly optimal verifiable data streaming. In: Cheng, C.-M., Chung, K.-M., Persiano, G., Yang, B.-Y. (eds.) PKC 2016 Part I. LNCS, vol. 9614, pp. 417–445. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49384-7_16

12. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). https://bitcoin.org/bitcoin.pdf
13. Noether, S.: Ring signature confidential transactions for Monero. Cryptology ePrint Archive, Report 2015/1098 (2015). http://eprint.iacr.org/
14. Pass, R., Shelat, A.: Micropayments for decentralized currencies. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 207–218. ACM (2015)
15. Pass, R., Shelat, A.: Micropayments for decentralized currencies (2016). http://eprint.iacr.org/2016/332
16. Poon, J., Dryja, T.: The Bitcoin lightning network: scalable off-chain instant payments. Technical report, Technical Report Draft v. 0.5.9.2 (2016) https://lightning.network/lightning-network-paper.pdf
17. Reid, F., Harrigan, M.: An analysis of anonymity in the Bitcoin system. In: 2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing, pp. 1318–1326. IEEE (2011)
18. Rivest, R.L.: Electronic lottery tickets as micropayments. In: Hirschfeld, R. (ed.) FC 1997. LNCS, vol. 1318, pp. 307–314. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63594-7_87
19. Ruffing, T., Kate, A., Schröder, D.: Liar, liar, coins on fire! Penalizing equivocation by loss of Bitcoins. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 219–230. ACM (2015)
20. Sasson, E.B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: decentralized anonymous payments from Bitcoin. In: 2014 IEEE Symposium on Security and Privacy, pp. 459–474. IEEE (2014)
21. Schöder, D., Simkin, M.: VeriStream – a framework for verifiable data streaming. In: Böhme, R., Okamoto, T. (eds.) FC 2015. LNCS, vol. 8975, pp. 548–566. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47854-7_34
22. Schröder, D., Schröder, H.: Verifiable data streaming. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 953–964. ACM (2012)
23. Sompolinsky, Y., Zohar, A.: Secure high-rate transaction processing in Bitcoin. In: Böhme, R., Okamoto, T. (eds.) FC 2015. LNCS, vol. 8975, pp. 507–527. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47854-7_32
24. Trillo, M.: Stress Test Prepares VisaNet for the Most Wonderful Time of the Year (2013). http://www.visa.com/blogarchives/us/2013/10/10/stress-test-prepares-visanet-for-the-most-wonderful-time-of-the-year/index.html
25. Wheeler, D.: Transactions using bets. In: Lomas, M. (ed.) Security Protocols 1996. LNCS, vol. 1189, pp. 89–92. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-62494-5_7