MicroBTC: Efficient, Flexible and Fair Micropayment for Bitcoin Using Hash Chains

Zhiguo Wan, Robert H. Deng, Kui Ren, and David Lee

No Institute Given

Abstract. While Bitcoin gains increasing popularity in different payment scenarios, the transaction fees make it difficult to be applied to micropayment. Given the wide applicability of micropayment, it is crucial for all cryptocurrencies including Bitcoin to provide effective support therein. In light of this, a set of low-cost micropayment schemes for Bitcoin have been proposed recently, and significantly reduce micropayment costs. The existing schemes, however, suffer from drawbacks such as high computation cost, inflexible payment value and possible unfair exchange. In this paper, we propose a new micropayment scheme for Bitcoin by integrating the hash chain technique into Bitcoin transactions. The proposed scheme MicroBTC not only enjoys great efficiency in computation and flexibility in micropayments, but also achieves fairness for both the payer and payee. We further propose a refund mechanism with a lock time to avoid indefinite transaction waiting. We analyze both complexity and security of MicroBTC and implement MicroBTC using the Bitcoin script language. Extensive experiments are conducted to validate its performance over Bitcoin. It is showed that a micropayment session can be processed within about 73ms for a laptop, and over 90% micropayment sessions end with a ratio of transaction fee to transaction value lower than 0.08%.

1 Introduction

Though the idea of digital currency dates back to 1980's and quite a few proposals have been suggested in the literature, only Bitcoin has gained surprising success and the widest adoption all over the world. Not only does Bitcoin have tens of millions of users, but also more and more merchants begin to support Bitcoin as a payment method, even though major countries have declared that Bitcoin is not officially accepted. Bitcoin is deemed as the most trustworthy cryptocurrency, and its market capitalization has reached several billion US dollars. Bitcoin transactions are also very active: there are tens of thousands of transactions every day and the transaction volume is about 1 million bitcoins.

Bitcoin enjoys great convenience and low cost for fund transactions. Built on top of Internet, the Bitcoin system can conveniently transfer funds to anyone as long as he can access Internet. Compared with the wire transfer, telegraph transfer and credit transactions, Bitcoin has the desirable advantage of very low transaction cost. The transaction fee for each Bitcoin transaction is as low as

0.0001 BTCs (name of the Bitcoin currency)[1], which is about 4 US cents currently. In contrast, the payment system Paypal has a much higher cost compared with Bitcoin (5% plus $0.05).

Despite the low transaction fee of Bitcoin, it cannot be used as a micropayment solution for very low value payment transactions for two reasons. First, the transaction fee is not low enough for micropayments. Although several US cents is bearable for normal transactions, it is too much for small transactions of several cents or even fractions of a cent. Such micropayments of very low value are important for many applications. For example, a WiFi hot spot charges a user for every minute the user accesses Internet from this hot spot, or a mobile phone charging station charges a mobile phone for every minute that it has been charged. In these cases, the payment could be as little as several cents, which is comparable to the Bitcoin transaction fee.

Even worse, the transaction fee is expected to grow in the long run due to the diminishing mining reward. The mining reward is halved every four years and it becomes 0 after all 21 million coins have been all minted. To compensate the expense of the miners, the transaction fee will be inevitably increased. Currently, the mining reward is 25 BTCs, and the transaction fee is less than 1% of the mining reward for each block. To make the Bitcoin system secure, there should be enough transaction fee that motivates enough miners to maintain a high hashrate, such that an adversary cannot easily mine a block. As analyzed in [3], the adversary can successfully launch selfish mining attacks if their computation power is more than 1/4 of that of the whole Bitcoin mining power.

Secondly, the overhead for Bitcoin transactions is too high. One needs to wait for about 10 minutes on average for his transaction mined and logged in the blockchain, and waits for 60 minutes (6 succeeding blocks) to confirm the transaction. Transaction confirmation is required to avoid possible forks in the blockchain. Hence, it is neither cost-effective nor convenient to use Bitcoin directly as a micropayment method.

Since micropayments have wide applications, it is crucial for cryptocurrencies to implement cost-effective and fast micropayments. To this end, Pass and Shelat [8] have introduced a set of lottery-based micropayment schemes for Bitcoin, which successfully reduce transaction cost of micropayments to a very low level. In addition, their schemes are enhanced to be compatible with the current Bitcoin protocol by introducing an offline trusted third party. However, their proposal has a number of drawbacks such as high computation cost, inflexible payment and possible unfair exchange, which are undesirable for high-frequency micropayments. Therefore, we attempt to address these issues by designing an efficient, flexible and fair micropayment scheme for the Bitcoin system in this paper.

Inspired by the micropayment scheme PayWord by Rivest and Shamir [12], we design MicroBTC, a highly efficient, flexible and fair micropayment scheme for Bitcoin based on the hash chain technique. Though very simple in design,

[1] In rare cases, the transaction fee is 0, but the miners may refuse to include the transaction into a candidate block for mining.

MicroBTC is very efficient in both computation and communication. It incurs much less signature computation cost while enjoying the same level of transaction cost as the scheme in [8]. MicroBTC also enjoys flexibility in payment since the amount of payment need not to be fixed before running the protocol. More important, it achieves fairness for both participants in the sense that the largest possible loss of either participant is only one piece of micropayment.

We further enhance MicroBTC to relieve the payer from indefinite waiting by adding a refund transaction. We provide detailed security analysis of our scheme and implement it as an extension to Bitcoin. We evaluate the performance of our scheme with extensive experiments in Bitcoin, and the results show that our scheme is highly efficient: a micropayment session can be processed within about 73ms for a laptop, and over 90% micropayment sessions end with a ratio of transaction fee to transaction value lower than 0.08%.

The remainder of the paper is structured as follows. In the next session, we review related work in micropayment schemes and provide preliminaries on Bitcoin. Then we present the basic MicroBTC scheme and discuss the refund mechanism in Section 3. Discussion on possible extensions to MicroBTC and detailed analysis of MicroBTC's performance and security are provided in Section 4. We describe the implementation of our scheme and present experiment results for performance evaluation in Section 5.

2 Related Work and Preliminaries

In this section, we review related work on micropayment and provide a brief introduction of the Bitcoin transaction, which is the basis of our micropayment scheme. After that, we review the micropayment schemes proposed by Pass and Shelat [8] and point out several issues in their proposals.

2.1 Micropayment Schemes

Micropayments refer to payment transactions of very low value, several cents or even less than 1 cent. The key issues in micropayments are efficiency and cost. Many proposals have been put forward to reduce cost of micropayments. They generally follow two different strategies: the hash chain based approach [12,4,1] and the lottery-based approach [10,6,11].

The hash chain based approach [12] was introduced by Rivest and Shamir. The key idea is to generate a long hash chain and use each hash value as a piece of micropayment. Due to the one-wayness of the hash function, the payee cannot reverse the hash chain to obtain additional payment. This idea was also proposed by Hauser *et al.* [4] and Anderson *et al.* [1]. Jutla and Yung later enhanced the hash chain technique to the hash tree [5], allowing a payer to use parts of the hash tree to pay different payees.

The lottery-based approach [10] was also proposed by Rivest, and it adopts a probabilistic method to do micropayments. The payer issues lottery tickets to the payee with each ticket winning the lottery for probability p. Each lottery ticket pays $\frac{1}{p}X$ with probability p to the payee, so that the expectation of the amount received by the payee is X. For example, if $p = \frac{1}{1000}$ then each ticket pays $1000X$ with probability $\frac{1}{1000}$ to the payee. This approach has two drawbacks: the payer and the payee need frequent interactions, and the payer may pay more than X to the payee. These issues have been addressed by a follow-up work of Micali and Rivest [6] using a tamper-resistant hardware. Nevertheless, the lottery-based schemes are more computationally intensive than the hash chain based ones.

2.2 The Bitcoin Transaction

The Bitcoin system is a peer-to-peer network that maintains a distributed consensus ledger, which is known as the blockchain. The blockchain, as its name implies, is formed by a chain of blocks, and each block contains up to several thousand transactions. Essentially, each transaction records how many bitcoins are sent from which address(es) to which address(es). An address in Bitcoin is the hash of a public key (Pay-to-Public-Key-Hash, P2PKH) or a redeem script (Pay-to-Script-Hash, P2SH) specifying the redeem condition, and the private keys are used to spend the fund held in this address. The public key and the corresponding private key are randomly generated by a user, so that there is no linkage to the user's identity. To receive bitcoins from someone, one just needs to provide his address to that party, who can construct a transaction to send the fund.

A transaction contains one or more inputs[2] and one or more outputs. Each input contains an input script (called scriptSig) while each output contains an output script (called scriptPubKey). In the standard transaction, the input script consists of a signature and the output script verifies this signature and outputs a boolean result. A transaction is valid only if the output script outputs 1, i.e. the verification of the signature in the input script is successful.

A transaction in Bitcoin is in the following format:

$$\mathsf{T} = (\mathcal{I}_1, \mathcal{I}_2, ..., \mathcal{I}_m; \mathcal{O}_1, \mathcal{O}_2, ..., \mathcal{O}_n; \mathsf{t}_{\mathrm{lock}}),$$

where $\mathcal{I}_i$ is an input to the transaction, $\mathcal{O}_j$ is an output, and t_{lock} is the lock time, indicating the time before which the transaction is not final and cannot be recorded on the blockchain.

The input of the transaction is in the following format: $\mathcal{I}_i = (h_{\mathsf{T}_i}, n_{\mathsf{T}_i}, \mathsf{scriptSig}_i)$ where h_{T_i} is the hash of a previous transaction from which the bitcoins are sent, n_{T_i} is the output index of the previous transaction, and $\mathsf{scriptSig}_i$ is the input script generated by the sender. The scriptSig of a P2PKH transaction contains a signature over the entire transaction content and a public key, i.e. $\mathsf{scriptSig}_i = \{\sigma_i, pk_i\}$, while for a P2SH transaction scriptSig contains signatures and/or other data as well as a redeem script, i.e. $\mathsf{scriptSig}_i = \{\sigma_i, ..., \mathsf{redeemScript}_i\}$.

[2] There is a special transaction called *coinbase* transaction that does not have any input; this transaction creates mint bitcoins.

A transaction output is $\mathcal{O}_j = (v_j, \mathsf{scriptPubKey}_j)$ where v_j is the amount of BTCs sent to the receiver and $\mathsf{scriptPubKey}_j$ is the output script that verifies the transaction spending this output. The output script $\mathsf{scriptPubKey}_j$ of a P2PKH transaction contains a hash of a public key and a redeem condition, i.e. $\mathsf{scriptPubKey}_j = \{\pi_j, pkh_j\}$. The redeem condition demands that pkh_j equals the hash of the public key in the corresponding input script, and the signature in the input script is a valid signature w.r.t. the public key. Only if σ_i and pk_i from $\mathsf{scriptSig}_i$ satisfy $\pi_j(\sigma_i, pk_i, pkh_j) = 1$, the fund to be spent in this transaction can be transferred.

A P2SH transaction output $\mathsf{scriptPubKey}_j = \{\pi_j, \mathsf{scriptHash}_j\}$. π_j demands that $\mathsf{scriptHash}_j$ equals the hash of $\mathsf{redeemScript}_i$ in the corresponding $\mathsf{scriptSig}_i$. Furthermore, signatures and/or data in $\mathsf{scriptSig}_i$ should satisfy the redeem condition specified by $\mathsf{redeemScript}_i$. The $\mathsf{scriptHash}_j$ is seen as an address, just like the public key hash. Since P2SH transactions can implement more versatile redeem conditions using redeem scripts, they have been used to implement *multisig* transactions and complex contracts.

To prevent double spending, the Bitcoin system adopts the proof-of-work (PoW) mechanism to verify transactions. To be able to include a candidate block into the blockchain, the miners in the Bitcoin system need to find a random nonce for that block such that the hash of the candidate block is less than a threshold. With the PoW mechanism, the blockchain becomes a trustworthy public ledger so that everyone can check all recorded transactions, and hence double spending is effectively prevented.

2.3 Micropayment schemes for Bitcoin

Here we give a brief discussion on a micropayment scheme for Bitcoin by Pass and Shelat [8] and another micropayment scheme from Bitcoin Wiki [2].

Lottery-based Micropayment for Bitcoin. Three lottery-based micropayment schemes are proposed in [8], but only the first scheme, called MicroPay1, is a fully decentralized solution, while the other two still require a trusted third party.

MicroPay1 reduces the number of transactions to be recorded on the blockchain to 1/100 for winning probability $p = 1/100$. As a result, 99% of the transaction cost is saved compared to the original Bitcoin protocol. Unfortunately, MicroPay1 has several drawbacks in payment flexility, fairness and computation complexity. First, MicroPay1 requires a fixed final payment value, which is not flexible for micropayments. In many cases, the payment value cannot be determined beforehand, e.g. a user may not be able to determine how long he would access Internet, or a driver cannot determine how long his car will park.

Second, this protocol may be unfair for either the payer or the payee. For the payer, he may enjoy the service for 50 times while the payee does not obtain a winning lottery ticket. So the payer can just terminate the protocol and leave the payee unpaid after providing services. Or on the other hand, there is a non-negligible probability that the payee obtains a winning lottery ticket within the first 5 interactions, so the payee can refuse to provide service anymore and terminate the protocol. Lastly, this protocol is not efficient as the payer needs to compute a signature in the payment issuance for each micropayment.

Micropayment based on Adjusted Transactions. The Bitcoin community also suggests a micropayment solution for Bitcoin [2], which has been implemented in a Bitcoin project bitcoinj. The idea is for the payer to generate a series of transactions in sequential with each transaction allocating more payment to the payee than its proceeding one. All these transactions are signed by the payer, but only the last transaction will be published to the Bitcoin system. Consequently, the transaction fee is significantly reduced. A refund transaction is required to get back the payer's coins when the payee just runs away.

This solution is very flexible on payment in that any amount of payment can be processed with this approach. But its drawback is the payer has to sign all the transactions while the payee needs to verify every signature. Meanwhile, the payer has to send the signatures to the payee. Therefore, this solution is inefficient in both computation and communication.

Lightning network [9] extends the above idea to a network of payment channels, enabling instant, scalable, and low-cost micropayments. It is a high-level architecture and can use our proposal as the underlying payment mechanism.

3 The Proposed Scheme

In this section, we first present our basic hash chain based micropayment scheme, called MicroBTC. Then we provide an enhancement to make it free of indefinite transaction waiting. Our system involves three entities: the Bitcoin system, a payer and a payee. We assume that the Bitcoin system and the blockchain are trusted, while the payer and the payer are not. Both the payer and the payee always have access to the Bitcoin system to make transactions. From now on, we refer to the payer as Alice or A, and the payee as Bob or B.

3.1 An Overview of MicroBTC

The basic idea is inspired by the PayWord scheme due to Rivest and Shamir [12] that is based on the hash chain technique. The payer generates a long hash chain, e.g., a hash chain of length 10000 is as follows: $h_{10000} \rightarrow h_{9999} ... \rightarrow h_1 \rightarrow h_0$, where $h(\cdot)$ is a cryptographic hash function, $h_i = h(h_{i+1})$, h_0 is called the root of the chain. The payer signs h_0 and send it to the payee as a payment commitment. The payer discloses a hash value on the hash chain in sequence to the payee, starting from h_1. Then h_i represents the i-th piece of micropayments. At the end of the exchange, the payee has the latest hash value, say h_i, and its distance to the chain root, i.e. i, is exactly the number of micropayment pieces. In this example, the payer can pay up to 10000 pieces of micropayment to the payee. The value of each piece of micropayment can be determined by the payer and the payee beforehand. Due to the one-wayness of the hash function, the payee cannot infer from his received hash values to obtain undisclosed hash values.

Our scheme integrates the above hash chain-based micropayment approach with Bitcoin as follows. The payer generates a hash chain and inserts the chain root h_0 to a commitment transaction. The redeem condition of the commitment transaction is configured that the payee needs to present a hash value on the chain to claim a part or all of the fund. Then the payer signs the commitment transaction with her private key and publishes it to the Bitcoin system for verification. Shortly, this commitment transaction will be verified by the Bitcoin miners and logged into the blockchain. After seeing that the commitment transaction has been published on the blockchain, the payee can start to serve the payer and the payer continuously sends hash values on the hash chain to the payee. At the end of the service session, the last hash value (say h_i) received by the payee will be used to claim the entire payment in one shot.

To claim the micropayment, the payee needs to generate a claim transaction that refers the commitment transaction, and put the last received hash value (h_i) into the claim transaction. This claim transaction divides the fund in the commitment transaction into two parts: i pieces of micropayment goes to the payee and the remaining part is returned to the payer. Then the payee signs the claim transaction and publishes it into the Bitcoin system. Since the commitment transaction contains a condition saying that the payee is to be paid i pieces of micropayment if he presents h_i, Bitcoin miners will verify h_i and the fund is correctly divided between the payee and the payer. If the claim transaction is valid, it will be logged into the Bitcoin blockchain, hence the payee is correctly paid.

Since this hash chain-based micropayment approach relies only on hash function and signature algorithm as Bitcoin does, it can be seamlessly integrated into the Bitcoin system.

3.2 MicroBTC: The Hash Chain based Micropayment for Bitcoin

Generally, MicroBTC adopts the classical hash chain technique, but necessary adaption to the original Bitcoin transaction and protocol is crucial to ensure security of micropayments. The main challenge is to support flexible payment value for micropayments. Current Bitcoin protocol does not specify how an UTXO (unspent transaction output) is spent in a spending transaction, i.e. how many coins is paid to whom. All it cares is whether the owner can produce a valid signature. Our micropayment scheme needs to divide the payer's fund to the payee (the payment) and payer (the change). To achieve our goal, we design a special redeem condition to divide the fund.

We let the payer send her fund to an escrow address corresponding to the redeem condition, which constructs the commitment transaction. In order to claim the fund from the escrow address, the payee can only divide the fund according to the redeem condition. So finally, the payee will get his due payment while the payer will receive the change. In order to prevent the payee from delaying the claim transaction for too long, the payee is required to sign a refund transaction with a lock time set to a future time. If the payee does not publish a claim transaction before the lock time, the payer can publish this refund transaction to get back her fund. So to protect his income, the payee will publish a claim transaction before the lock time expires, which prevents the fund freezing issue.

MicroBTC consists of four steps, namely Micropayment Initiation, Micropayment Commitment, Micropayment Issuance, Micropayment Claim.

– **Micropayment Initiation**: First of all, the payer A and the payee B negotiate the value of each micropayment, and the payer estimates the maximum amount of payment in this session, e.g. V. The payer generates a long hash chain using a cryptographic hash function h: $h_N \to h_{N-1} ... \to h_1 \to h_0$, where $h_i = h(h_{i+1})$ and N is a sufficiently large integer. The payer creates an address (pk_A, sk_A) to receive the change, and the payee creates an address (pk_B, sk_B) to receive micropayments.
– **Micropayment Commitment**: The payer creates a MicroBTC redeem script $\mathsf{redeemScript}$ as well as $\mathsf{scriptHash}^{\text{esc}}$ using h_0, N, pk_A, pk_B and a predetermined maximum payment value V. The script $\mathsf{redeemScript}$ contains $\{h_0, V, N, pkh_A^{\text{esc}}, pkh_B^{\text{esc}}\}$ and specifies the micropayment redeem condition, which is to be described in micropayment claim step. Here $pkh_A^{\text{esc}} = h(pk_A)$ and $pkh_B^{\text{esc}} = h(pk_B)$.
Then the payer sends V BTCs to the address encoded from $\mathsf{scriptHash}^{\text{esc}}$ with a commitment transaction $\mathsf{T}^{\text{com}} = (*; \mathcal{O}^{\text{esc}}; 0)$. Wlog, we assume that this transaction has only one output and we do not care about the inputs in this transaction. Here $\mathcal{O}^{\text{esc}} = (V, \mathsf{scriptPubKey}^{\text{esc}}) = (V, \pi_{\text{esc}}, \mathsf{scriptHash}^{\text{esc}})$. The redeem condition π_{esc} just verifies validity of the redeem script. The payer publishes this transaction and it will be included in the blockchain. After this commitment transaction is recorded on the blockchain, the payee can be sure that the micropayment is ready for issuance.
– **Micropayment Issuance**: As agreed between the payer and the payee, the payee provides service to the payer, while the payer sends hash values on the hash chain one by one to the payee. Specifically, each time the payer sends (i, h_i) for $i = 1, 2, ..., N$ in sequential to the payee, who verifies that $h_i = h^i(h_0)$. If the verification is successful, the payee will continue to serve the payer; otherwise he stops serving the payer. On the other hand, the payer continues to issue hash values to the payee only if the payee provides her service. As such, the potential loss of both the payer and the payee is just one piece of micropayment, so the fairness is achieved in this sense.
– **Micropayment Claim**: When the service session is over, the payee can claim his payment using the latest hash value, say h_i, that he receives from the payer. Then he can create a claim transaction to divide the V BTCs between himself and the payer. The claim transaction is defined as follows:

$$\mathsf{T}^{\text{claim}} = (\mathcal{I}^{\text{esc}}; \mathcal{O}_A, \mathcal{O}_B; 0), \tag{1}$$

$$\mathcal{I}^{\text{esc}} = (h_{\mathsf{T}^{\text{esc}}}, n_{\mathsf{T}^{\text{esc}}}, \mathsf{scriptSig}^{\text{esc}}), \tag{2}$$

$$\mathsf{scriptSig}^{\text{esc}} = \{\sigma_B, pk_B, h_i, i, \mathsf{redeemScript}^{\text{esc}}\}, \tag{3}$$

$$\mathcal{O}_A = (v_A, \pi_A, pkh_A), \tag{4}$$

$$\mathcal{O}_B = (v_B, \pi_B, pkh_B). \tag{5}$$

Here $h_{\mathsf{T}^{\text{esc}}}$ is the hash of the escrow transaction, and $n_{\mathsf{T}^{\text{esc}}}$ is the index of the output that holds the V BTCs in the escrow transaction. v_A is the change returned to A and v_B is the payment to B. $v_A + v_B$ is equal to V minus transaction fee. pkh_A and pkh_B are the hashes of the public key of A and B respectively. $\mathcal{O}_A$ and $\mathcal{O}_B$ are standard P2PKH outputs corresponding to A and B respectively.
The claim transaction is first verified with the redeem condition π_{esc} ($\mathsf{redeemScript}^{\text{esc}}$, $\mathsf{scriptHash}^{\text{esc}}$): $\mathsf{scriptHash}^{\text{esc}} \stackrel{?}{=} \text{Hash}(\mathsf{redeemScript}^{\text{esc}})$. Then the redeem script $\mathsf{redeemScript}^{\text{esc}}$, which contains $\{h_0, V, N, pkh_A^{\text{esc}}, pkh_B^{\text{esc}}\}$, verifies the claim transaction as follows:

1. $h_i \stackrel{?}{=} h^i(h_0)$;
2. $v_B \stackrel{?}{=} i * V/N$, $v_A \stackrel{?}{=} V - v_B - \text{tx}_{\text{fee}}$;
3. $pkh_A \stackrel{?}{=} pkh_A^{\text{esc}}$, $pkh_B \stackrel{?}{=} pkh_B^{\text{esc}}$;
4. $pkh_B \stackrel{?}{=} h(pk_B)$;
5. σ_B is a valid signature over $\mathsf{T}^{\text{claim}}$ w.r.t. pk_B.

Here tx_{fee} is the transaction fee, about 0.0001 BTC.

There are several differences that distinguish the above transactions from the standard transactions in Bitcoin. First, the redeem condition in the commitment transaction takes as input more parameters than the standard transactions. To correctly divide the fund between the payer and the payee, the inputs to the redeem condition include the hash chain root h_0, the latest hash value h_i received by the payee and the length of the hash chain. These three parameters determine the final payment due to the payee, i.e. $i * V/N$ BTCs.

Second, the redeem condition in the commitment transaction has also specified conditions on the outputs in the claim transaction[3]. The first condition is that the fund should be correctly divided between the payer and the payee; the second condition specifies the receiving addresses must be pk_A and pk_B. However, there is no constraints on the redeem condition in the outputs of $\mathsf{T}^{\text{claim}}$.

In both transactions above, we set the lock time as 0, such that the transactions can be be accepted for mining at once.

3.3 MicroBTC without Indefinite Transaction Waiting

In our basic MicroBTC scheme, only after the payee publishes the claim transaction can the payer get back her change. If the payee delays the claim transaction publication, the payer may wait indefinitely for her change. In order to avoid this problem, we make use of the lock time in Bitcoin transactions to implement guaranteed change return within a pre-determined period. The idea is for the payee to sign another transaction, referred to as the *Refund* transaction, and send it to the payee. This refund transaction has a lock time set to a future time which allows enough time for the payee to claim his payment. This time is

[3] Note that the signature check operation for transaction verification, done with OP_CHECKSIG, needs access to the whole transaction. Since the outputs are a part of the transaction, it is natural for the redeem condition to have access to the outputs.

application-specific, e.g. 1 week for car parking or 1 day for WiFi access. Since before the lock time expires the refund transaction cannot be included into the blockchain, the payment to the payee is secured during this period.

The original MicroBTC is enhanced by adding two new steps: the refund setup before the Micropayment Commitment, and the refund claim as the (optional) last step.

– **Refund Setup**: The payer prepares a refund transaction with its lock time set to be a future time, e.g. 1 day from now. Essentially, this refund transaction is a special type of claim transaction with $h_i = h_0$. This transaction redeems all the fund in the escrow address (currently not published yet) to the payer:

$$\mathsf{T}^{\text{refund}} = (\mathcal{I}^{\text{esc}}; \mathcal{O}'_A, \mathcal{O}'_B; t_{\text{lock}}), \tag{6}$$

$$\mathcal{I}^{\text{esc}} = (h_{\mathsf{T}^{\text{esc}}}, n_{\mathsf{T}^{\text{esc}}}, \mathsf{scriptSig}^{\text{esc}}), \tag{7}$$

$$\mathsf{scriptSig}^{\text{esc}} = \{\sigma_B, pk_B, h_0, 0, \mathsf{redeemScript}^{\text{esc}}\}, \tag{8}$$

$$\mathcal{O}'_A = (V - \text{tx}_{\text{fee}}, \pi_A, pkh_A), \tag{9}$$

$$\mathcal{O}'_B = (0, \pi_B, pkh_B). \tag{10}$$

– **Refund Claim**: If the payer does not receives her change (if any) after the lock time t_{lock} expires, she publishes $\mathsf{T}^{\text{refund}}$ to the Bitcoin P2P network to get back her fund. Once this transaction is verified and included into the blockchain, the payer gets back all of her fund (minus necessary transaction fee) and the payee gets nothing.

4 Discussion and Analysis

In this section, we first discuss how MicroBTC can be further extended using hash tree and multi-dimensional hash chains. Then we discuss efficiency and flexibility of MicroBTC compared to existing schemes, and provide a brief security analysis of our scheme.

4.1 Extensions

Our hash chain based micropayment scheme can only pay a single payee with one hash chain. Multiple hash chains are required to pay multiple payees. This can be simply implemented by using multiple outputs in the escrow transactions. Each output pays a different payee, and each hash chain is independent from others. One transaction can pay as many payees as one wishes.

Alternatively, it can be extended using hash tree [5] and multi-dimensional hash chains [7] to pay different payees. This will further reduce the signature generation cost of the payer and transaction cost is also deduced due to fewer transactions. Here we give a brief description of these two techniques.

– PayTree [5] is a k-ary balance tree that pays the payee with its leaves. Each leaf is associated with a secret label that represent a unit of monetary value. It can be used with the hash chain micropayment schemes like PayWord. However, it is a bit complex for implementation.
– Multi-dimensional hash chain (MDHC) [7] is implemented with pair commutative hash functions. Each hash function represents a dimension, and one payee can be paid with a different dimension. But the multi-dimensional hash chain, implemented with RSA modular exponentiations, may not be very efficient.

4.2 Efficiency

Due to the hash chain technique, our scheme is more efficient in computation and communication than existing micropayment schemes for Bitcoin. We provide a comparison on computation and communication cost between our scheme and two existing schemes in [8] and [2]. Suppose there are n pieces of micropayments processed between the payer and the payee in a session, the computation and communication costs of different schemes are listed in Fig. 1. Our scheme shows great advantage in computation cost over the other two: the payer and the payee needs to compute a signature and verify a signature respectively, while the other two schemes require n signature generation and n verifications. Hash chain generation is very efficient, so it is ignored in the table.

All three schemes need n interactions in the micropayment session, so the communication overhead is $O(n)$. Nevertheless, the number of messages exchange in the micropayment session in our scheme is only half of that of MicroPay1.

Scheme	Comp. Cost		Comm. Cost
	Payer	Payee	
MicroPay1 [8]	n sig	n ver	$2n$ msgs
Bitcoin-MicroPay [2]	n sig	n ver	n msgs
MicroBTC	1 sig	1 ver	n msgs

Fig. 1. Efficiency comparison for a micropayment session with n interactions. sig denotes signature generation, ver denotes signature verification.

In addition, our scheme is more flexible than micropayment schemes in [8]. Our scheme can deal with the case where the amount of payment cannot be determined beforehand. Furthermore, our scheme can also make variable payments by disclosing multiple consecutive hash values (or just the last one of these hash values).

4.3 Security Analysis

Fairness. All micropayment schemes cannot achieve true fairness, while our scheme achieves fairness in the sense of very little potential loss. According to the specification of MicroBTC, the payer will stop sending more hash values if she cannot receive any service after sending out the lash hash value, and the payee will stop providing more service if he does not receive any more payment since the last time. So the potential loss of either participant is only one piece of micropayment, which is negligible for both the payer and the payee. Therefore, the proposed scheme can be seen as a fair micropayment scheme.

As discussed earlier, the micropayment schemes in [8] are not fair for either payer or payee.

Payment security. As we have assumed that there is a secure channel between the payer and the payee, the public keys exchanged between them are always secure and authenticated. Meanwhile, we assume that the Bitcoin system is trusted and can always be accessed by both participants. Attacks can only come from the payer or the payee, and they intend to obtain more financial benefits than specified by the protocol.

MicroBTC is secure because of the one-wayness of the hash function and the signature over the whole transaction, including the hash chain root and the output script for payment verification. To obtain illegal benefit, the payee needs to either reverse the hash chain or tamper the transactions without being detected. Thus, MicroBTC is just as secure as the standard transaction in Bitcoin, and the only difference is the payment processing logic.

Length of the Hash Chain. The payer may cheat on the length of the hash chain, but this will not benefit her at all. Suppose the payer generates a hash chain of length 10000, but she tell the payee that the length $N = 1000$. Then after 1000 times of service and micropayments, the payee thinks the total payment is reached, so stops servicing and proceeds to claim the payment. The remaining 9000 hash values are just useless for anyone. On the other hand, the payer can tell the payee the length $N = 20000$. Then after 10000 times of micropayments the payer cannot send more hash values so the payee stops service. The payer cannot gain any benefit either in this case.

5 Implementation and Performance Evaluation

The script language used in Bitcoin is not Turing-complete, and it does not support loops so as to avoid possibly high verification cost of transactions. Fortunately, hash computation is quite efficient so it is acceptable to extend Bitcoin script language to to implement MicroBTC.

We propose to extend Bitcoin scripting language with a new operation, say OP_CHECK_MICROBTC, to implement our algorithm. This new operation will take 7 inputs and outputs success or failure, as described below.

– OP_CHECK_MICROBTC$(h_i, i, h_0, V, N, pkh_A^{\text{esc}}, pkh_B^{\text{esc}}) \rightarrow$ (-/False): This operation takes 7 inputs from the top of the stack, parses them as $(h_i, i, h_0, V, N, pkh_A^{\text{esc}}, pkh_B^{\text{esc}})$. Then it retrieves the two output values (v_A, v_B) and two output addresses (pkh_A, pkh_B) from the two outputs. The verification literally follows the first 3 redeem conditions listed in the **Micropayment Claim** step in

Section 3. It returns nothing if the verification succeeds and false if the verification fails.

Accordingly, the scripts are as follows:

– scriptSig:
$$[\sigma_B]\ \ [pk_B]\ \ [h_i]\ \ [i]\ \ [\mathsf{redeemScript}_i], \tag{11}$$
where $\mathsf{redeemScript}_i$ is

$$[h_0]\ \ [V]\ \ [N]\ \ [pkh_A^{\mathrm{esc}}]\ \ [pkh_B^{\mathrm{esc}}]\ \ \mathsf{OP\_CHECK\_MICROBTC}\ \ \mathsf{OP\_DUP}$$
$$\mathsf{OP\_HASH160}\ \ [pkh_B^{\mathrm{esc}}]\ \ \mathsf{OP\_EQUALVERIFY}\ \ \mathsf{OP\_CHECKSIG}$$

– scriptPubKey:
$$\mathsf{OP\_HASH160}\ \ [\mathsf{scriptHash}_j]\ \ \mathsf{OP\_EQUAL} \tag{12}$$

The size of the redeem script is only 109 bytes, assuming V is 8 bytes and N is 4 bytes. It checks that the fund is correctly divided according to the hash value h_i and the signature is valid. The complete script verification process can be found in Appendix.

Implementation. We have implemented MicroBTC based on the source codes of the latest Bitcoin Core (version 0.12.0), which is the reference implementation of the Bitcoin protocol. Three different types of transactions are supported in our implementation: the commitment transaction (a normal P2SH transaction), the claim transaction (a special P2SH transaction, the MicroBTC transaction), and the refund transaction (also a MicroBTC transaction). The Bitcoin Core source codes is revised to enable it to create, sign and verify these MicroBTC transactions. Our implementation has the following functions:

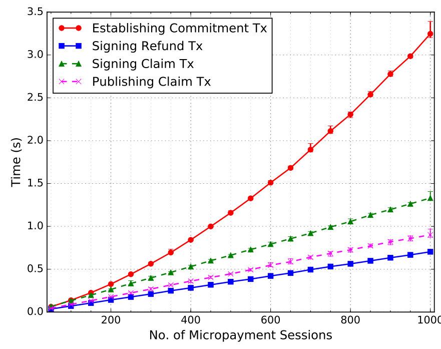
– Creating the escrow address and redeem script for the MicroBTC commitment transaction, then the payer's fund can be sent to the address to make the micropayment commitment;
– Creating an unsigned MicroBTC claim transaction based on the hash value provided by the payee;
– Signing the MicroBTC claim transaction with the payee's private key and published for micropayment claim;
– Creating/signing a MicroBTC refund transaction sending the fund back to the payer.

Experiment design. We conduct a series of tests under the regression test mode which enables us to mine blocks as fast as we want. Each test consists of the following steps:

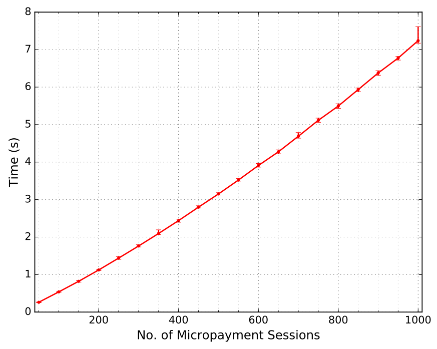
1. Preparation: we create n hash chains of length 10000, and from each hash chain root we generate a MicroBTC escrow address for a micropayment session.
2. Commitment: For each micropayment session, we send 2 BTCs (thus each piece of micropayment is 0.0002 BTC) to the corresponding escrow address from one of our UTXOs. This normal P2SH transaction is the commitment transaction, and only the payee can spend this fund with a valid hash value.

3. Refund: We establish a micropayment refund transaction using the hash chain root h_0, sending the fund (minus transaction fee) to the payer. This transaction has a lock_time set as 100 blocks from now, and it is signed by the payee's private key. The signed transaction is supposed to be returned to the payer, but it is not accepted by the miners before the lock_time expires.
4. Claim: We choose a random number $i (0 < i < 10000)$ and obtain the corresponding hash value h_i, and suppose they are the last items received by the payee. We create a MicroBTC claim transaction from the hash value h_i and i. Then it is signed by the payee's private key.
5. Publishing: The claim transaction is published to the Bitcoin P2P network and verified by the miners.

We conduct the experiments on a laptop equipped with Intel Core i3-4030U CPU(1.90GHz, 3M cache), 8G RAM and running 64-bit Ubuntu 15.10. In our experiment, we run the above test with $n = 50, 100, ..., 1000$ for 20 times to obtain the final performance results. On this experiment platform, the ECDSA signature generation over secp256k1 (used in the Bitcoin protocol) costs 101.60 μs, while signature verification time is 12.65 μs.



(a) Computation cost of MicroBTC

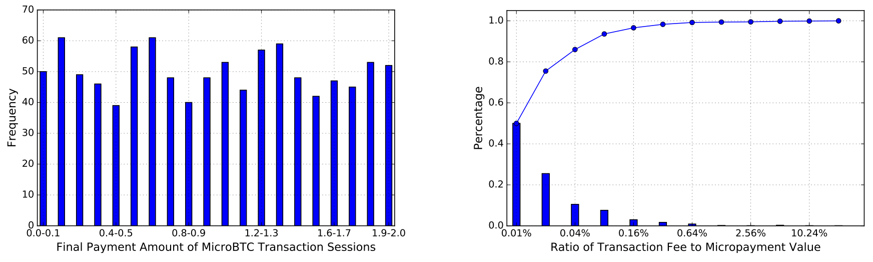


(b) Overall computation cost

Fig. 2. Computation cost of MicroBTC

Fig. 2(a) shows the computation time for different operations. Establishing a commitment transaction requires 1 signature generation and 2 signature verifications, so it takes more time than other operations. Since it has to find a UTXO to make the commitment, it takes longer and longer time when the number of transactions in the blockchain increases. Signing a transaction, either the claim transaction or the refund transaction, needs 1 signature generation and 1 signature verification. Since the refund transaction is much simpler than the claim transaction, so signing the refund transaction is faster. Publishing the signed claim transaction to Bitcoin miners requires to verify the signature in the claim signature as well as check its validity according to the Bitcoin protocol, so it takes longer time than signing a refund transaction. Nevertheless, each operation can be finished in less than 4ms. Fig. 2(b) shows the overall computation overhead of the tests (without micropayment issuance). It shows that the average overhead for one micropayment session is about 73ms, which is very modest for current laptops and PCs.

Fig. 3(a) illustrates the distribution of the final payment value of micropayment sessions for a test with 1000 sessions. Since a uniform random number generator is used to create micropayment sessions, the micropayment values approximately conform to the uniform distribution. We analyze the transaction fee ratio, which is the ratio of transaction fee to transaction values, in Fig. 3(b). It shows that the transaction fee ratio is only 0.01% for half of the micropayment sessions (assume transaction fee is 0.0001 BTC), and more than 90% of micropayment sessions has a transaction fee ratio less than 0.08%.



(a) Micropayment Value Distribution (b) Transaction Fee Ratio

Fig. 3. Distribution of Transaction Values and Transaction Fee Ratio

6 Conclusion

In this paper we have designed an efficient, flexible and fair micropayment scheme called MicroBTC for Bitcoin and other blockchain-based cryptocurrencies. MicroBTC is designed by integrating hash chains with the redeem scripts in Bitcoin transactions. Compared with existing micropayment solutions, MicroBTC is more efficient in computation, more flexible in payment value, and it also achieves better fairness for micropayments. We have implemented MicroBTC based on the latest Bitcoin Core source code by adding a new transaction type and a new operation. The experimental results show that MicroBTC is very efficient in computation.

References

1. Anderson, R., Manifavas, H., Sutherland, C.: Netcard - a practical electronic cash system. In: Proc. of the 4th Security Protocols Workshop. pp. 69–87 (1996)
2. Bitcoin Wiki: Rapidly-adjusted (micro)payments to a pre-determined party, `https://en.bitcoin.it/wiki/Contract`
3. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: Proc. of FC'14 (2014)
4. Hauser, R., Steiner, M., Waidne, M.: Micropayments based on iKP. In: Proc. of Worldwide Congress on Computer and Communications Security Protocol (1996)

5. Jutla, C., Yung, M.: Paytree: "amortised-signature" for flexible micropayments. In: Proc. of the 2nd USENIX Association Workshop on Electronic Commerce. pp. 213–221 (1996)
6. Micali, S., Rivest, R.: Micropayments revisted. In: Proc. of CT-RSA'02 (2002)
7. Nguyen, Q.S.: Multi-dimensional hash chains and application to micropayment schemes. In: Proc. of WCC'06, LNCS 3969. pp. 218–228 (2006)
8. Pass, R., Shelat, A.: Micropayments for Decentralized Currencies. In: Proc. of ACM CCS'15 (2015)
9. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2015), `https://lightning.network/lightning-network-paper.pdf`
10. Rivest, R.: Electronic lottery tickets as micropayments. In: Proc. of Financial Cryptography (1997)
11. Rivest, R.: Peppercoin micropayments. In: Proc. of Financial Cryptography. pp. 2–8 (2004)
12. Rivest, R., Shamir, A.: Payword and micromint: two simple micropayment schemes. In: Proc. of the 4th Security Protocols Workshop. pp. 69–87 (1996)

Appendix

The execution of MicroBTC scripts is illustrated in Table 1. Note that from step 5 to step 6, the redeem script is expanded and duplicated to execute with the stack.

Table 1. Script Execution Process with OP_CHECK_MICROBTC

Stack	Scripts
	$[\sigma_B]$ $[pk_B]$ $[h_i]$ $[i]$ [redeemScript$_i$] OP_HASH160 [scriptHash$_j$] OP_EQUAL
$[\sigma_B]$ $[pk_B]$ $[h_i]$ $[i]$ [redeemScript$_i$]	OP_HASH160 [scriptHash$_j$] OP_EQUAL
$[\sigma_B]$ $[pk_B]$ $[h_i]$ $[i]$ [scriptHash$'_i$]	[scriptHash$_j$] OP_EQUAL
$[\sigma_B]$ $[pk_B]$ $[h_i]$ $[i]$ [scriptHash$'_i$] [scriptHash$_j$]	OP_EQUAL
$[\sigma_B]$ $[pk_B]$ $[h_i]$ $[i]$ True	
$[\sigma_B]$ $[pk_B]$ $[h_i]$ $[i]$	$[h_0]$ $[V]$ $[N]$ $[pkh_A^{\text{esc}}]$ $[pkh_B^{\text{esc}}]$ OP_CHECK_MICROBTC OP_DUP OP_HASH160 $[pkh_B^{\text{esc}}]$ OP_EQUALVERIFY OP_CHECKSIG
$[\sigma_B]$ $[pk_B]$ $[h_i]$ $[i]$ $[h_0]$ $[V]$ $[N]$ $[pkh_A^{\text{esc}}]$ $[pkh_B^{\text{esc}}]$	OP_CHECK_MICROBTC OP_DUP OP_HASH160 $[pkh_B^{\text{esc}}]$ OP_EQUALVERIFY OP_CHECKSIG
$[\sigma_B]$ $[pk_B]$	OP_DUP OP_HASH160 $[pkh_B^{\text{esc}}]$ OP_EQUALVERIFY OP_CHECKSIG
$[\sigma_B]$ $[pk_B]$ $[pk_B]$	OP_HASH160 $[pkh_B^{\text{esc}}]$ OP_EQUALVERIFY OP_CHECKSIG
$[\sigma_B]$ $[pk_B]$ $[p\hat{k}h_B]$	$[pkh_B^{\text{esc}}]$ OP_EQUALVERIFY OP_CHECKSIG
$[\sigma_B]$ $[pk_B]$ $[p\hat{k}h_B]$ $[pkh_B^{\text{esc}}]$	OP_EQUALVERIFY OP_CHECKSIG
$[\sigma_B]$ $[pk_B]$	OP_CHECKSIG
True	