MicroCash: Practical Concurrent Processing of Micropayments

Ghada Almashaqbeh[1⋆], Allison Bishop[2,3⋆⋆], and Justin Cappos[4]

[1] CacheCash Development Company, NY, USA ghada@cs.columbia.edu
[2] Columbia University, NY, USA allison@cs.columbia.edu
[3] Proof Trading, NY, USA
[4] New York University, NY, USA jcappos@nyu.edu

Abstract. Micropayments have a large number of potential applications. However, processing these small payments individually can be expensive, with transaction fees often exceeding the payment value itself. By aggregating the small transactions into a few larger ones, and using cryptocurrencies, today's decentralized probabilistic micropayment schemes can reduce these fees. Unfortunately, existing solutions force micropayments to be issued sequentially, thus to support fast issuance rates a customer needs a large number of escrows, which bloats the blockchain. Moreover, these schemes incur a large computation and bandwidth overhead, limiting their applicability in large-scale systems.

In this paper, we propose MicroCash, the first decentralized probabilistic framework that supports concurrent micropayments. MicroCash introduces a novel escrow setup that enables a customer to concurrently issue payment tickets at a fast rate using a *single* escrow. MicroCash is also cost effective because it allows for ticket exchange using only one round of communication, and it aggregates the micropayments using a non-interactive lottery protocol that requires only secure hashing and supports fixed winning rates. Our experiments show that MicroCash can process thousands of tickets per second, which is around 1.7-4.2x times the rate of a state-of-the-art sequential micropayment system. Moreover, MicroCash supports any ticket issue rate over any period using only one escrow, while the sequential scheme would need more than 1000 escrows per second to permit high rates. This enables our system to further reduce transaction fees and data on the blockchain by $\sim 50\%$.

1 Introduction

Micropayments, or payments in pennies or factions of pennies, have a large a number of potential applications as diverse as ad-free web surfing, online gaming, and peer-assisted service networks [19]. This paradigm allows participants to exchange monetary incentives at a small scale, e.g., pay per minute in online games. Such a fine-grained payment process has several advantages, including

⋆ Most work done while at Columbia supported by NSF CCF-1423306.
⋆⋆ Supported by NSF CCF-1423306 and NSF CNS-1552932.

a great deal of flexibility for customers who may stop a service at any time. In addition, it reduces the financial risks between mutually-distrusted participants, where there is no guarantee that a client will pay after being served, or that a server will deliver service when paid in advance.

However, processing these small payments individually can incur high transaction fees that exceed the payment value itself. For example, the average base cost of a debit or credit card transaction in the US is around 21 to 24 cents, and 23 to 42 cents [5,6], respectively. In cryptocurrencies such a fee could be even higher, e.g., above $1 in Bitcoin [3]. Beside this financial drawback, handling micropayments individually can impose a huge workload on the system, and may make the logs needed for accountability purposes unwieldy. Thus, there is a need for a payment aggregation mechanism that records fewer transactions while still compensating properly for the small payments received to date.

Probabilistic micropayment schemes have emerged as a solution that fits the criteria outlined above [23,21,18,22]. In these models, the total payment value is locked in an escrow and micropayments are issued as lottery tickets. Each ticket has a probability p of winning a lottery, and when it wins, produces a transaction of β currency units. This means that, on average, only one transaction is processed out of a batch of $1/p$ tickets. Unfortunately, these early proposals rely on a trusted party to audit the lottery and manage payments. Such a centralized approach may increase the deployment cost and limit the use of the payment service to systems with fully authenticated participants [14].

As cryptocurrencies evolved, a number of initiatives attempted to convert these schemes to distributed ones [19,14]. This was done by replacing the trusted party with the miners, and utilizing the blockchain to provide public verifiability of system operation. Yet, these approaches have several drawbacks that may hinder their usage in large-scale systems. First, they force a customer to issue micropayments sequentially using the same escrow. This is because in these schemes an escrow is only funded to pay only one winning lottery ticket. Hence, a new ticket cannot be issued until the merchant reports the lottery outcome and confirms that the previous one did not win. To issue tickets at a fast rate under this restriction, the customer needs to create a large number of escrows, which increases the amount of data on the blockchain and transaction fees. Second, these schemes rely on computationally-heavy cryptographic primitives [19,14], and several rounds of communication to exchange payments [14]. Such performance issues reduce the potential benefits of micropayments.

To address these issues, this paper introduces MicroCash, the first decentralized probabilistic framework that supports *concurrent* micropayments. MicroCash features a novel escrow setup that allows a customer to issue micropayments in parallel and at a fast rate using a *single* escrow that can pay many winning tickets. This is achieved by having the customer specify the total number of tickets it may issue, and provide an escrow balance that covers all winning tickets under its payment setup.

MicroCash is also cost effective because it introduces a lightweight non-interactive lottery protocol. This protocol requires only secure hashing and allows

a payment exchange using only one round of communication without demanding the merchant to report anything to the customer. Furthermore, this protocol is the first to eliminate the possibility that all lottery tickets may win or lose the lottery. Although the probability of these events is very small, the fear of paying much more than expected may discourage customers from using the system [18]. Moreover, accounting for the worst case when almost all tickets win requires a large escrow balance, which increases the collateral cost. Our protocol alleviates this concern by selecting an *exact* number of winning tickets each round (where a round is the time needed to mine a block on the blockchain). In particular, all tickets issued in the same round are tied to a *lottery draw* value in a future block on the blockchain. This value is used to determine the fixed-size set of winning tickets through an iterative hashing process. Lastly, the security of the system is enforced using both cryptographic and financial techniques. The latter requires a customer to create a penalty escrow that is revoked upon cheating, with a lower bound derived using a game theoretic modeling of the system.

To evaluate the efficiency of MicroCash, we test its performance against MICROPAY [19], a state-of-the-art sequential micropayment scheme. Our results show that a modest merchant machine in MicroCash is able to process 2,240 - 10,500 ticket/sec, which is around 1.7-4.2x times the rate in MICROPAY, with 60% reduction in the aggregated payment size. Furthermore, a modest customer machine in MicroCash is able to concurrently issue more than 33,000 ticket/sec using *one* escrow over any period, while MICROPAY requires the creation of more than 1000 escrows per second to support a comparable issue rate. This allows MicroCash to reduce transaction fees and amount of data on the blockchain in a video delivery and online gaming applications by $\sim 50\%$.

2 Related Work

Probabilistic Micropayments. The idea of probabilistic micropayments dates back to the seminal works of Wheeler [23] and Rivest [21,22]. In these schemes, a customer and a merchant run the lottery (on each ticket independently) by using a simple coin tossing protocol, with a chance of more, or less, winning tickets than expected. All of these schemes rely on a centralized bank to track and authorize payments. This imposes additional overhead on the users who have to establish business relationships with this bank. It also limits the use of the service to only fully authenticated users. Although they allow for concurrent micropayments [22], this centralization issue is viewed as one of the main reasons for the limited adoption of such solutions [14].

Cryptocurrency-based probabilistic micropayments can potentially overcome both the cost and efficiency problems inherent in earlier schemes. To the best of our knowledge, only two such schemes have been proposed to date in the literature, MICROPAY [19] and DAM [14].

MICROPAY translates what Rivest [21] proposed into an implementation on top of a cryptocurrency system. Instead of using an authorized bank account, any customer creates an escrow on the blockchain to issue lottery tickets. MICROPAY implements a similar interactive coin tossing protocol for the lottery,

and adds an alternative non-interactive version that reduces the communication complexity (a merchant still has to report the lottery result back to the customer). However, the latter is computationally-heavy since it requires public key cryptography-based operations and a non-interactive zero knowledge (NIZK) proof system. Moreover, MICROPAY only supports sequential micropayments as mentioned earlier. DAM shares similar constraints, but unlike the public MICROPAY it preserves user privacy by implementing anonymous micropayments.

We believe that the added blockchain transactions due to sequential payments, coupled with the high computation cost, point to the need for optimized approaches that support concurrent micropayments at a lower overhead. This need is the motivation behind building MicroCash.

Payment Channels and Networks. Payment channels were originally developed to handle micropayments in Bitcoin [2], where they rely on a similar concept of processing small payments locally. Later on, this paradigm was utilized to improve the scalability of cryptocurrencies [17,15,20], where off-chain processing is utilized to reduce on-chain traffic, and hence, increase the transaction throughput of the system.

In general, payment channels and networks require an escrow to be created between each pair of parties along the payment path. This may result in a higher collateral cost than probabilistic micropayments, since in the latter the same escrow can be used to pay several parties directly. These costs may indirectly push the network towards centralization [7] since only wealthy parties can afford multiple escrows to create payment paths. Thus, most users will rely on these parties, or hubs, to relay the off-chain transactions. In addition, each hub on the path charges a fee to relay payments. With micropayments, such a setup would be infeasible because these fees could be much larger than the payments themselves. Probabilistic approaches, on the other hand, eliminate any fees when exchanging lottery tickets. As a result, distributed probabilistic micropayments provide a better solution for handling small payments in cryptocurrency systems.

3 Threat Model

Processing off-chain transactions in distributed probabilistic micropayments creates the potential for various types of attacks. In this section, we outline a threat model capturing these attacks, which guided the design of MicroCash. In developing this model, we make the following assumptions:

- No trust is placed in any (insider or outsider) party.
- Participants are rational, meaning that they choose to follow the protocol, or deviate from it, based on what will maximize their utility gain.
- The underlying cryptocurrency scheme is secure in the sense that the majority of the mining power is honest. This means that the confirmed state of the blockchain contains only valid transactions, and that an attacker who tries to mutate or fork the blockchain will fail with overwhelming probability.
- Hash functions are modeled as random oracles, and the hash values of the blocks on the blockchain are modeled as a uniform distribution.

– Efficient adversaries cannot break the basic cryptographic building blocks (SHA256, digital signatures, etc.) with non-negligible probability.
– Communication between customers and merchants takes place over a channel that provides integrity, confidentiality, and authenticity, such as TLS/SSL.

We used the ABC framework [10] to build a comprehensive threat model for distributed probabilistic micropayment schemes[5]. During this process, we identified the assets to be protected in such systems, which include the escrows, the lottery tickets, and the lottery protocol. Then, by analyzing the security requirements of these assets, and examining more than 120 threat cases, we produced the following list of broad threat categories endemic to distributed probabilistic micropayments:

– **Escrow overdraft:** A customer creates a payment escrow insufficient for honoring the winning lottery tickets, or creates a penalty deposit that does not cover the cheating punishment imposed by the miners. Such a threat could stem from creating small balance escrows, or from front running attacks in which a customer withdraws the escrows before paying.
– **Unused-escrow withholding:** An attacker prevents or delays a customer from withdrawing its unused escrows. For example, merchants may delay claiming their winning lottery tickets to keep the payment escrow on hold.
– **Lottery manipulation:** An attacker attempts to influence the outcome of the lottery draw, and hence, bias the payment process.
– **Denial of service (DoS):** This is a large threat category that threatens any distributed system. This work focuses on attacks related to the payment process, like preventing a customer from creating escrows.
– **Duplicate ticket issuance:** A customer uses the same sequence number to issue several lottery tickets to different merchants. As this means creating more tickets than the escrow can cover, the customer obtains more service than it can pay for.
– **Invalid payments:** A malicious customer issues lottery tickets that do not comply with its payment setup or with the system specifications. Because these tickets will be rejected by the miners if they win the lottery, the customer can avoid paying merchants.

Note that dealing with malicious merchants who collect lottery tickets and do not deliver a service is outside the scope of MicroCash. The same is true for dealing with malicious customers who may obtain the service without paying. In this work, we are concerned with the payment scheme design, rather than how to exchange service for a payment, which is part of the application design.

In addition, MicroCash does not address payment anonymity (as in [14]). Addressing this issue securely, while preserving the low overhead of MicroCash, is a direction for our future work.

[5] A detailed documentation of this process is available online [8] and is based on the generic description of probabilistic micropayments as described in the introduction.

4 MicroCash Design

Having outlined the security threats to probabilistic micropayments, and the limitations of existing solutions, this section presents the design of MicroCash, a concurrent micropayment system that addresses these issues. We start with an overview of the system, followed by a more detailed technical description.

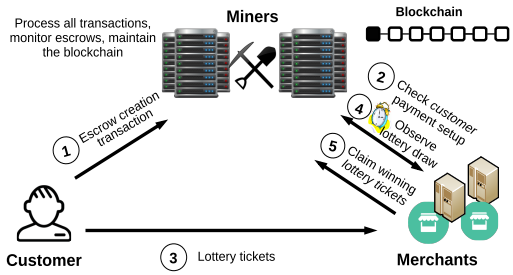

Fig. 1: Flow of operations in MicroCash.

A high level illustration of MicroCash, that also captures the remainder of this section's organization, is found in Figure 1. As shown, during the payment setup (**Step 1**, Section 4.1), each customer issues a transaction creating two escrows: payment and penalty. The customer uses the former to pay merchants in the form of lottery tickets, while the miners use the latter to financially punish this customer if it cheats. Merchants can check the escrow setup before transacting with the customer when the escrow transaction is confirmed on the blockchain (**Step 2**). In exchange for the delivered service, the customer issues lottery tickets according to a schedule that limits the number of tickets over a set period (**Step 3**, Section 4.2). To claim payments, a merchant holds a ticket until its lottery draw time, and determines if this ticket won based on a value derived from the block mined at that time (**Step 4**, Section 4.3). If it is a winning ticket, the merchant can claim currency from the customer's escrow during the ticket redemption period (**Step 5**, Section 4.4). This interaction continues until the end of the escrow lifetime. At that time, and after all issued tickets expire, the customer can spend any remaining funds.

4.1 Escrow Setup

MicroCash introduces a novel escrow setup that allows multiple winning tickets to be redeemed. This enables both concurrent ticket issuance and reduces the amount of escrow-related data. This setup also provides techniques to determine the escrow balance needed to cover all concurrent tickets, and to track the issuance of these tickets in a distributed way.

Escrow creation. As an off-chain payment scheme, MicroCash must ensure that customers can and will pay. This includes honoring winning tickets, and, if caught cheating, complying with a stipulated financial punishment. To satisfy these requirements, each customer must create payment and penalty escrows with sufficient funds to cover both eventualities.

Given that each payment escrow must be tied to a penalty escrow, a customer sets up both using one creation transaction. This transaction provides funds to be locked under each escrow balance, which we refer to as B_{escrow} (payment) and $B_{penalty}$ (penalty). It also configures a set of parameters that influence how these balances are computed, and how they are to be spent. These parameters, whose values are specified by the customer possibly after negotiating with the merchants, include the following:

- The lottery winning probability p.
- The currency value of a winning lottery ticket β.
- The ticket issue rate tkt_{rate}, which is the maximum number of tickets a customer is allowed to hand out per round. This is used to calculate which *ticket sequence numbers* are valid within each ticket issuing round.
- A lottery draw round length, denoted as $draw_{len}$, such that $draw_{len} \in \{1, \ldots, c\}$ for some small system parameter c. The customer has to configure $draw_{len}$, p, and tkt_{rate} in a way that makes $p\ tkt_{rate} draw_{len}$ of an integer value (this is the number of winning tickets in a lottery draw).
- The set of beneficiary merchants that can be paid using the escrow, where the size of this set is denoted as m.

Computing B_{escrow} and $B_{penalty}$ based on the above parameters proceeds as follows. To permit concurrent micropayments, B_{escrow} must be large enough to pay all winning tickets tied to an escrow. Given that each winning ticket has a value of β currency units, and that there are $p\ tkt_{rate} draw_{len}$ winning tickets per $draw_{len}$ rounds, B_{escrow} can be simply computed as follows (where l_{esc} is the escrow lifetime in rounds, and there are $l_{esc}/draw_{len}$ lottery draws)[6]:

$$B_{escrow} = \beta\ p\ tkt_{rate} l_{esc} \tag{1}$$

For $B_{penalty}$, we compute a lower bound for this deposit by using an economic analysis that quantifies the additional utility gain, or profit, a customer obtains by cheating. The profit is the monetary value of the service exchanged for invalid or duplicated tickets to merchants before cheating is detected, i.e., before any of these tickets wins the lottery and is claimed through the miners (assuming that merchants do not exchange any information about the received tickets). Thus, to make cheating unprofitable, and hence, unappealing to rational customers, $B_{penalty}$ is set to be at least equal to this additional utility as given by the following equation[7]:

$$B_{penalty} > (m-1)p\ \beta\ tkt_{rate} draw_{len} \left(\frac{1-p}{1-\rho^{-1}} + draw_{len} \Big((1-p)(d_{draw} - 1) + d_{redeem} \Big) \right) \tag{2}$$

[6] Compared to previous schemes [19,14], this is the same expected payment amount needed to cover the same number of winning tickets. However, since these works are sequential, they distribute this amount among multiple escrows instead of one.

[7] Compared to DAM [14], MicroCash's penalty escrow will be larger. This is because the cheating detection period in MicroCash is longer (several rounds until the lottery is run and a winning ticket is claimed). In DAM, the lottery is run over a ticket immediately when it is received, and a claim, if any, can be issued at the same time. Thus, assuming identical payment setup, $B_{penalty}$ in MicroCash is approximately $T_{\mathsf{MicroCash}}/T_{DAM}$ times the one in DAM, where T is the cheating detection period.

where d_{draw} is the lottery draw period in rounds, d_{redeem} is the ticket redemption period in rounds (more about these parameters in Section 4.3), and $\rho = \binom{a}{b}$ such that $a = tkt_{rate} draw_{len}$ and $b = (1-p)a$. The full details of deriving this bound are found in Section 5 in the full version of the paper [11].

Upon receiving the escrow creation transaction, the miners verify the correctness of a payment setup as follows. First, they check that the customer owns the input funds. Then, the miners use B_{escrow} to compute the escrow lifetime as $l_{esc} = \frac{B_{escrow}}{\beta\ p\ tkt_{rate}}$. After that, they check that both l_{esc} and $p\ tkt_{rate} draw_{len}$ are of integer values, $draw_{len}$ is within the allowed range, and that l_{esc} is multiples of $draw_{len}$. Lastly, the miners verify that $B_{penalty}$ satisfies the bound given above. If all these checks pass, the miners add the escrow transaction to the blockchain. Otherwise, they reject the escrow by dropping its transaction.

Escrow management. In MicroCash, the escrow funds can be spent only for a restricted set of transactions. These include claiming winning tickets, presenting proofs-of-cheating, and enabling a customer to spend its unused escrow funds.

To track the locked funds, miners maintain a state for each escrow in the system. This state includes the following:

- The ID of the escrow, which is a random value generated by the miner who adds the escrow creation transaction to the blockchain.
- The balances of both the payment and penalty escrows.
- The public key of the owner customer, which is used to verify all signed tickets that are issued using this escrow.
- The values of p, β, l_{esc}, tkt_{rate}, $draw_{len}$, and the set of beneficiary merchants (both the public key of each merchant and a corresponding index).
- An escrow refund time, denoted as t_{refund}, at which the customer can spend any remaining funds. Miners set this time to be equal to the expiry time of the tickets issued in the last round of an escrow lifetime.

Ticket issuance using an escrow must follow a schedule based upon the tickets' sequence numbers. That is, if an escrow supports a rate of tkt_{rate} tickets per round, then in the first round tickets with sequence numbers 0 to tkt_{rate}-1 may be issued. In the second round tickets with sequence numbers tkt_{rate} to $2tkt_{rate}$-1 can be issued, and so on until the last round of an escrow lifetime. Merchants will accept tickets in the current round with sequence numbers that follow this assignment schedule. As customers and merchants may have inconsistent views of the blockchain, and hence, may not agree on what is the current round, i.e., height of the blockchain, merchants will also accept tickets from the prior and next round, as long as these tickets use the correct sequence number range.

An example of a ticket issuing schedule is found in Figure 2. As shown, the escrow creation transaction is published at round 10 and confirmed at round 16. This escrow has $l_{esc} = 3$ rounds, and allows a ticket issue rate of 1000 tickets per round. Thus, the customer has 3 ticket issuing rounds, starting at round 17, with the sequence number ranges shown in the figure.

The miners update the escrow state based on the escrow related transactions (mentioned earlier) they process. For example, redeeming a winning ticket

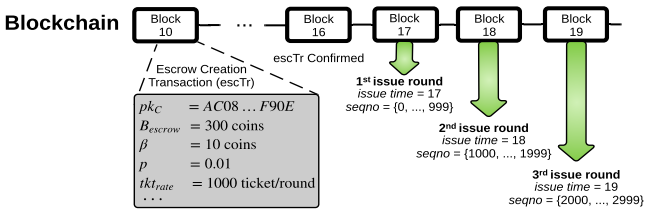

Fig. 2: An example of a ticket issuing schedule.

reduces B_{escrow} by β coins, and receiving a valid proof-of-cheating against the customer causes the miners to revoke the funds in $B_{penalty}$. All these transactions are logged on the blockchain, which permits anyone to validate the state.

The miners discard an escrow state once all tickets tied to this escrow expire. This happens at time t_{refund}, or when an escrow is broken after receiving a valid proof-of-cheating (discussed in Section 4.5). At that time, the customer may spend any remaining funds in its escrows.

4.2 Paying with Lottery Tickets

After the escrow is confirmed on the blockchain, a customer can start paying for service by giving merchants lottery tickets. A lottery ticket tkt_L is a structure containing several fields as follows:

$$tkt_L = index_M||id_{esc}||seqno||\sigma_C \tag{3}$$

where $index_M$ is the recipient merchant index as listed in the escrow state, id_{esc} is the escrow ID, *seqno* is the ticket sequence number, and σ_C is the customer's signature covering all the previous fields. The *seqno* field, along with id_{esc}, identifies a ticket, which also provides means for ticket tracking in the system. Note there is no need to include the escrow configuration or the parties' public keys in the ticket itself, these can be looked up on the blockchain using id_{esc}.

When issuing a ticket, the customer fills in the above fields and signs the ticket using its secret key tied to the public key used when creating the escrow. The ticket *seqno* can be any sequence number within the range assigned to the current ticket issue round. The customer can continue issuing lottery tickets, without waiting for the lottery results of previously issued ones, until it finishes all sequence numbers in this range. After that, it must wait the next round to generate more tickets.

Upon receiving a ticket, a merchant verifies it as follows:

- Check that the escrow is not broken.
- Check that its index $index_M$, that appears in the ticket, is identical to the one listed in the escrow state.
- Check the freshness of *seqno* (i.e., that no earlier ticket, associated to the same escrow, carries the same *seqno*).
- Verify that *seqno* is within the valid range based on the ticket issuing schedule. (As mentioned before, to handle inconsistencies in the blockchain view, tickets from the previous or next issuance round can be accepted.)
- Verify σ_C over the ticket using the customer's public key.

If any of the above checks, except the fourth one, fails, the merchant drops the ticket. On the other hand, if the ticket has an out-of-range sequence number (i.e., larger than the maximum sequence number allowed by the escrow), the recipient merchant can issue a proof-of-cheating that will cost the customer its penalty deposit. Otherwise, if all the above checks pass, the merchant accepts the ticket and keeps it until its lottery draw time.

4.3 The Lottery Protocol

MicroCash introduces a lightweight lottery protocol that relies solely on secure hashing. This protocol does not require any interaction between the customer and the merchant. Instead, it utilizes only the state of the blockchain, where the lottery draw outcome is determined by a value derived from the block mined at the lottery draw time.

To specify the lottery draw time, MicroCash defines two system parameters, d_{draw} and $draw_{len}$. d_{draw} represents the least number of rounds a ticket has to wait after its issue round (which we call t_{issue}) until it enters the lottery. $draw_{len}$ determines the number of consecutive ticket issuing rounds that all their lottery tickets enter the same lottery draw. Hence, if $draw_{len} = 1$, then the draw time t_{draw} of a ticket is computed as $t_{draw} = t_{issue} + d_{draw}$. On the other hand, if $draw_{len} > 1$, then t_{draw} of a ticket is t_{draw} of the last ticket issuing round in the contiguous set of rounds[8].

A clarifying example of determining the lottery draw time is found in Figure 3. As shown, starting with round 28, which the first ticket issuing round, each set of contiguous $draw_{len}$ rounds enter the lottery together. For example, all tickets issued in rounds 28, 29, and 30 enter the lottery at round 40, which is 10 rounds after the last ticket issue round in this set.

Whether a ticket wins or loses depends on a lottery draw value tied to the block mined at time t_{draw}. This value is computed using a simple verifiable delay function (VDF) [13] that is evaluated over this block. This evaluation takes a period of time, hence the name delay function, where this period is a system parameter. Consequently, when a miner mines the block at index t_{draw}, it cannot tell immediately which ticket will win or lose. This miner has to compute the VDF over this block to know the lottery draw outcome.

We instantiate this VDF using iterative hashing, where the number of iterations is set to a value that delays producing the output by the period specified in the system. In addition, we let the miners compute this function as part of the mining process. That is, when a miner mines a new block, it evaluates the VDF over the previous block. Therefore, the VDF value of the block at index t_{draw} appears on the blockchain when the block at index $t_{draw} + 1$ is mined.

Accordingly, in our protocol a merchant holds a ticket tkt_L until its lottery draw time t_{draw}. Then, after observing the VDF value of the block mined at that time, the miners, and any party, can compute the set of winning sequence numbers for that round as follows. First, the hash of the VDF value along with

[8] Since $draw_{len}$ affects t_{draw} of a ticket, MicroCash specifies a small interval for its possible values to prevent a customer from excessively delaying paying merchants.

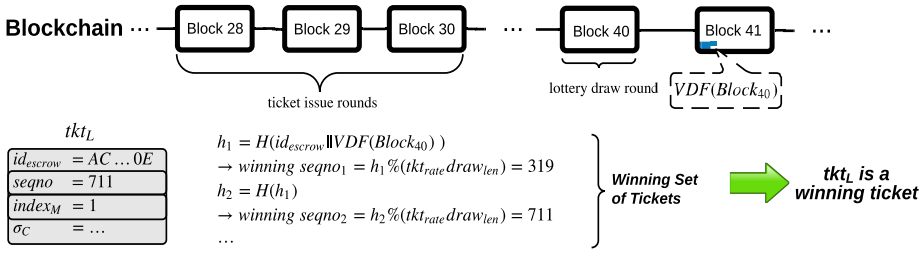

Fig. 3: Lottery draw example ($draw_{len} = 3$, $p = \frac{1}{300}$, $tkt_{rate} = 10^3$, and $d_{draw} = 10$).

the escrow ID is computed, which we call h_1, and then h_1 is mapped to a sequence number within the assigned range of the ticket issuing rounds tied to t_{draw}. To obtain the second winning sequence number, the hash of h_1 is computed to obtain h_2, and then h_2 is mapped to a sequence number in the given range. If a collision occurs, i.e., a previously seen number is produced, it is discarded and the process proceeds with hashing h_2 to obtain h_3, and so on. This continues until a set of distinct $p\, tkt_{rate} draw_{len}$ winning sequence numbers is drawn[9].

The previous process is clarified by the example depicted in Figure 3. As shown, tkt_L was issued in round 28, the first ticket issuing round, and hence, it entered the lottery at round 40. The VDF value of the block with index 40 appears inside block 41. By using this value, a set of winning sequence numbers has been chosen, based on which the ticket in the figure is a winning one because its sequence number is within this set.

Note that the lottery draw involves only values that are part of the escrow state. In other words, it relies on parameters that the issuing customer cannot manipulate, which do not include the merchant recipient address. This means that a ticket's chance of winning the lottery is not affected by who owns it. As such, if a customer issues tickets with duplicated sequence numbers to multiple merchants, all these tickets will win or lose together. If the tickets win, detecting cheating is trivial because merchants will publish their winning tickets to the blockchain to redeem the tickets.

4.4 Claiming Winning Tickets

After the lottery draw, a merchant can collect currency from the customer's escrow by redeeming its winning tickets (if any). This is done by issuing a redeem transaction that has the winning ticket as input, and has β coins directed to the merchant's address as output.

To allow the miners to resolve tickets and release escrow funds back to the customer in a reasonable timeframe, MicroCash specifies a redeem period for each ticket. This is done by defining a system parameter called d_{redeem} that determines the number of rounds during which a ticket can be redeemed. A ticket expires at time $t_{expire} = t_{draw} + d_{redeem}$. Thus, d_{redeem} must be set to a value that allows merchants to redeem their winning tickets.

[9] We design a version of this lottery protocol with independent ticket winning events in Appendix A in the full version [11]. This can be used in case it is infeasible in some applications to configure $p\, tkt_{rate} draw_{len}$ to be an integer.

After receiving a redeem transaction, the miners process it as follows:

- Check that the transaction complies with the system specifications.
- Verify the redeemed ticket as outlined in Section 4.2.
- Verify that the ticket is a winning one by checking that its sequence number is among the winning set drawn at time t_{draw} of this ticket.
- Check that the ticket has not expired.
- Verify the merchant's signature over the redeem transaction using the public key corresponding to $index_M$ found in the escrow state.
- Check that no other ticket with the same sequence number and tied to the same escrow has already been redeemed. If it is, this is a proof of duplicate ticket issuance and is used as a proof-of-cheating against the customer.

If all these checks pass, miners approve the redeem transaction and update the escrow state accordingly. Otherwise, they drop an invalid transaction and, if a proof-of-cheating is produced, revoke the customer's penalty deposit.

4.5 Processing Proof-of-cheating

A proof-of-cheating is a transaction any party who witnesses a cheating incident can present to the miners. In MicroCash, such an incident could be issuing tickets with out-of-range sequence numbers or issuing duplicated tickets. A signed ticket with an out-of-range sequence number or signed tickets with duplicated sequence numbers are publicly verifiable proofs against the issuing customer.

If cheating is verified, miners revoke the customer's penalty escrow tied to its payment escrow referenced in the ticket as follows. In case of ticket duplication, the miners first pay all duplicated winning tickets from the payment escrow, and then from the penalty deposit. Next, they publish an escrow break transaction containing the proof-of-cheating on the blockchain. This transaction burns the revoked penalty deposit rather than providing them to another party to eliminate the chance that this party may collude with the customer to receive those funds. Respecting the lower bound of $B_{penalty}$, as specified before, ensures that all the above cheating behaviors are less profitable than acting in an honest way. Hence, it makes such behaviors unappealing to rational customers.

5 Security and Game Theory Analysis

In this section, we analyze the resilience of MicroCash to the threats outlined in Section 3. To defend against these threats, our scheme utilizes cryptographic and financial approaches. Due to space constraints, this section presents a brief version of this analysis, but a more complete and detailed one can be found in Section 6 in the full version of the paper [11].

MicroCash addresses the *escrow overdraft* threat by using its escrow setup. The miners will reject any escrow with payment or penalty balances that do not satisfy the bounds defined earlier. Furthermore, no customer can perform a front running attack by withdrawing escrows before paying. This is because escrow fund release is triggered only by the receipt of a valid winning lottery ticket (for a payment escrow) or a valid proof-of-cheating (for a penalty escrow). In addition, a customer who tries to perform an indirect withdrawal by issuing

winning tickets to itself after observing the lottery draw outcome will also fail. As the ticket issue schedule specifies both issue and lottery draw time for each round, it will be too late to select only winning sequence numbers after t_{draw}. By that time, merchants have already received their tickets, and any unissued winning ticket that a customer may try to claim is covered by the escrow balance.

The *unused-escrow withholding* threat is also handled by MicroCash's escrow setup. When all tickets tied to an escrow expire, i.e., at time t_{refund}, the miners will allow the customer to spend the residual balance. This prevents locking unused escrow funds indefinitely on the blockchain.

The *lottery manipulation* threat is addressed by MicroCash's lottery draw mechanism. The draw outcome depends only on values that a customer cannot manipulate. These include a ticket sequence number, which must be within a predetermined range, the escrow ID that appears in the escrow state, and the VDF value of the block mined at time t_{draw}. The probability of predicting the latter is negligible (in the random oracle model and under the assumption that block hashes on the blockchain are modeled as a uniform distribution). Hence, a customer cannot know which ticket will win or lose in advance. Also, given that the VDF takes time to be computed, a miner who may perform selective mining (possibly in collusion with the customer) by evaluating the VDF first, and then announcing a favorable block, will have a low chance of publishing this block on the blockchain. This is because other miners will announce their newly mined blocks immediately, which will have higher probability of being adopted. As such, any lottery ticket has a fair chances of winning the lottery.

For *DoS*, which is a large threat category to any system, we limit our focus to cases related to the design of MicroCash. These include preventing customers from creating escrows, preventing merchants from claiming their winning tickets, or selectively relaying blocks based on their content. The case of miners disregarding specific transactions/blocks may take place when an attacker controls a substantial portion of the mining power, or when the attacker controls the network links and tries to isolate participants. Under the assumption that the majority of the mining power is honest, and by having each participant connect to a large number of miners, the impact of this threat can be reduced. To protect against selective relaying, techniques that allow propagating messages without disclosing their content can be employed, e.g., BloXroute [4]. Such mechanisms are independent of the design of MicroCash, and so it is up to the parties themselves to adopt suitable solutions.

MicroCash uses a detect-and-punish approach to financially mitigate the *duplicate and invalid ticket issuance* threats. Any party that detects any of these events can produce a proof-of-cheating against the issuing customer containing the duplicated or invalid tickets as a proof. Once such an incident is verified, miners burn the customer's penalty escrow as a punishment.

We compute the value of $B_{penalty}$ by using a game theoretic analysis in which we model the setup of MicroCash as a repeated game over the escrow lifetime[10].

[10] Although Chiesa et al. [14] present an economic analysis for the DAM penalty escrow, the derived bound cannot be used with MicroCash. This is due to the differences in

Then we quantify the the monetary value of the additional service a customer can obtain in exchange for the duplicated, or invalid, tickets during the cheating detection period. That is, cheating is detected when any of the duplicated tickets wins the lottery and is claimed through the miners, which happens in $d_{draw} + d_{expire}$ rounds after the ticket issue time. Thus, we set $B_{penalty}$ to be at least equal to service monetary value obtained during this period. Here, we only state our result while the full modeling and proof can be found in Section 5 in the full version [11].

Theorem 1. *For the game setup defined in Section 5 in the full version [11], issuing invalid or duplicated lottery tickets is less profitable in expectation than acting in an honest way if (where* $\rho = \binom{tkt_{\text{rate}}}{(1-p)tkt_{\text{rate}}}$*):*

$$B_{\text{penalty}} > (m-1)p\ \beta\ tkt_{\text{rate}}\text{draw}_{\text{len}}\left(\frac{1-p}{1-1/\rho} + \text{draw}_{\text{len}}\Big((1-p)(d_{\text{draw}}-1) + d_{\text{redeem}}\Big)\right)$$

6 Performance Evaluation

To understand the performance benefit of concurrent probabilistic micropayments, in this section we evaluate the computation, bandwidth, and payment setup costs of MicroCash. We implemented benchmarks for the functions used for generating tickets, verifying these tickets, and performing a lottery draw[11]. We used SHA256 for hashing, and for digital signatures, we tested the most widely used schemes: ECDSA over secp256k1, ECDSA over P-256, and EdDSA over Ed25519 [12]. To put our results in context, we compare our scheme with MICROPAY [19], particularly its fully decentralized version MICROPAY1 with its non-interactive lottery protocol. In implementing this protocol, we used the verifiable random function (VRF) construction introduced by Goldberg et al. [16].

For each of the tested schemes, we computed the rate at which customers, merchants, and miners can process lottery tickets. Also, we calculated the bandwidth overhead by reporting on the size of tickets when exchanged between the various parties. To evaluate the effect of micropayment concurrency, we computed the number of escrows a customer would need to support the ticket issue rate in each of the tested schemes. Lastly, we studied two real life applications, online content delivery and online gaming, to derive workload numbers and used them to quantify the overhead of processing micropayments in such applications.

Our experiments were implemented in C on an Intel Core i7-4600U CPU @ 2.1 GHz, with 4 MB cache and 8 GB RAM. Each of the payment processing functions was called 10^6 times. Due to space constraints, this section provides only a brief discussion, while a complete report can be found in the full version [11].

---

the system setup and the lottery timing, which affects the cheating detection period and the duplication decisions a customer can make.

[11] It should be noted that due to requiring a VDF evaluation and the new transaction types, MicroCash is not compatible with Bitcoin-like systems. For smart contract-based systems, if a periodic unbiased source of randomness exists to replace the VDF, then MicroCash can be implemented as a smart contract that uses this source for the lottery.

Table 1: Ticket processing rate (ticket / sec).

	MICROPAY			MicroCash		
	ECDSA (secp256k1)	**ECDSA (P-256)**	**EdDSA (Ed25519)**	**ECDSA (secp256k1)**	**ECDSA (P-256)**	**EdDSA (Ed25519)**
Customer	1,859	32,471	26,238	1,868	33,006	26,749
Merchant	1,328	2,399	2,561	2,249	10,505	8,473
Miner	1,340	2,448	2,617	2,241	10,345	8,368

Lottery ticket processing rate. Table 1 shows the ticket processing rates. Customers in both schemes generate tickets at comparable rates because the operations performed are almost identical in MicroCash and MICROPAY. Given that the heaviest operation in this process is signing a ticket, the generation rates improve by using an efficient digital signature scheme, where performance is boosted by around 17x and 14x when ECDSA (secp256k1) is replaced with ECDSA (P-256) and EdDSA (Ed25519), respectively.

The trend is different for merchants and miners. These parties in MicroCash are 1.7x, 4.2x, and 3.2x faster than in MICROPAY for the three digital signature schemes. This is because of the operations that miners and merchants need to perform when running and verifying the lottery draw outcome in each system. In MicroCash, this process involves only lightweight hash operations, while the lottery in MICROPAY requires evaluating a computationally-heavy VRF.

Furthermore, the table shows that merchants and miners in MicroCash benefit more from the efficiency of the digital signature scheme. This is because the heaviest operation these parties perform in MicroCash is verifying a customer's signature. However, in MICROPAY the bottleneck is evaluating a VRF and producing a correctness proof of the output the merchant side, and verifying this proof on the miner side. As shown in the table, MICROPAY obtains only around 1.9x improvement when replacing ECDSA (secp256k1) with any of the other two schemes. In contrast, MicroCash achieves around 4.7x and 3.8x improvement when replacing ECDSA (secp256k1) with ECDSA (P-256) or EdDSA (Ed25519), respectively.

Lottery ticket bandwidth overhead. In terms of bandwidth, MicroCash incurs less overhead than MICROPAY because its lottery tickets are smaller. A ticket sent from a customer to a merchant is 110 bytes in MicroCash, while it is 274 bytes in MICROPAY. A winning ticket sent from merchants to miners is also 110 bytes in MicroCash, while it is 355 bytes for MICROPAY because this ticket must be accompanied with a NIZK proof. This means that MicroCash incurs only 40% of the bandwidth overhead of MICROPAY between customers and merchants, and only 31% of the overhead between merchants and miners.

To put these numbers in context, we report on the transaction sizes in Bitcoin. The average size is around 500 bytes, where a transaction with one or two inputs is about 250 bytes [9]. Adding a winning ticket as one of these inputs produces a claim transaction with a size of 360 bytes in MicroCash, which is less than the average Bitcoin transaction size. On the other hand, in MICROPAY the size of a claim transaction will be 605 bytes, exceeding the average size.

Size of escrows on the blockchain. One major difference between concurrent and sequential micropayment schemes is that a customer in the latter needs a new escrow after each winning ticket, and to issue tickets in parallel at a fast rate, this customer has to create a large number of escrows. This dramatically increases the overhead since each of these escrows requires an individual creation transaction, paying a transaction fee, and logging on the blockchain.

For example, to support the ticket issue rates reported in Table 1, a MICROPAY customer would need a number of escrows that depends on the network latency and a merchant's ticket processing rate. Using the average US RTT of 31 ms [1], in the best case an escrow in MICROPAY can be used to issue 32 tickets per second (if none of these ticket win or only the last one wins). Therefore, a customer in MICROPAY would need 60, 1019, or 653 escrows per second to support the generation rates for signature schemes ECDSA (secp256k1), ECDSA (P-256), or EdDSA (Ed25519), respectively, as found in Table 1. On the other hand, a customer in MicroCash would need only *one* escrow with the proper balance to pay at any given ticket rate. As such, MicroCash dramatically reduces the amount of data logged on the blockchain.

Micropayments in real world applications. To ground our results in real world numbers, we examined two applications; online gaming and peer-assisted content delivery networks (CDNs). We computed the overhead of processing micropayments with parameter values derived from the service price and the application workload. This is done for three cases: Bitcoin with no micropayment scheme, Bitcoin with MICROPAY, and Bitcoin with MicroCash.

Our results confirm that MicroCash is cost efficient enough to be used in online gaming and content distribution. Since it is a concurrent scheme that allows issuing payments in parallel using a single escrow, MicroCash decreases the total data added to the blockchain by around 50% as compared to MICROPAY. The full details of this evaluation can be found in Section 7.3 in the full version [11].

7 Conclusions

In this paper, we introduce MicroCash, the first decentralized probabilistic framework that supports concurrent micropayments. The design of MicroCash features an escrow setup and ticket tracking mechanism that permit a customer to rapidly issue tickets in parallel using only *one* escrow. Moreover, MicroCash is cost effective, as it implements a non-interactive lottery protocol for micropayment aggregation that requires only secure hashing. When compared to the sequential scheme MICROPAY, MicroCash has substantially higher payment processing rates and much lower bandwidth and on-chain traffic. This demonstrates the viability of employing our scheme in large-scale micropayment applications.

References

1. *At&t Network Averages.* `https://ipnetwork.bgtmo.ip.att.net/pws/averages.html`.
2. Bitcoinj. `https://bitcoinj.github.io/working-with-micropayments`.

3. *BitInfoCharts, Bitcoin avg. transaction fee.* https://bitinfocharts.com/comparison/bitcoin-transactionfees.html.
4. *BloXroute: A Scalable Trustless Blockchain Distribution Network.* https://bloxroute.com/wp-content/uploads/2018/03/bloXroute-whitepaper.pdf.
5. *Board of Governers of the Federal Reserve System, press release June 2011.* https://www.federalreserve.gov/newsevents/pressreleases/bcreg20110629a.htm.
6. *Board of Governers of the Federal Reserve System, Regulation II.* https://www.federalreserve.gov/paymentsystems/regii-about.htm.
7. "lightning network will be highly centralized". https://cointelegraph.com/news/lightning-network-will-be-highly-centralized-gavin-andresen.
8. *Supplemental Material.* https://www.dropbox.com/s/799j92dnyz2bskk/microcashThreatModel.pdf?dl=0.
9. *TradeBlock: Analysis of Bitcoin Transaction Size Trends.* https://tradeblock.com/blog/analysis-of-bitcoin-transaction-size-trends.
10. Almashaqbeh, G., Bishop, A., and Cappos, J. Abc: A threat modeling framework for cryptocurrencies. In *IEEE INFOCOM Workshop on Cryptocurrencies and Blockchains for Distributed Systems (CryBlock)* (2019).
11. Almashaqbeh, G., Bishop, A., and Cappos, J. Microcash: Practical concurrent processing of micropayments. *arXiv preprint arXiv:1911.08520* (2019). https://arxiv.org/abs/1911.08520.
12. Bernstein, D. J., Duif, N., Lange, T., Schwabe, P., and Yang, B.-Y. High-speed high-security signatures. *Journal of Cryptographic Engineering 2*, 2 (2012).
13. Boneh, D., Bonneau, J., Bünz, B., and Fisch, B. Verifiable delay functions. In *Annual International Cryptology Conference* (2018), Springer.
14. Chiesa, A., Green, M., Liu, J., Miao, P., Miers, I., and Mishra, P. Decentralized anonymous micropayments. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2017), Springer, pp. 609–642.
15. Decker, C., and Wattenhofer, R. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems* (2015), Springer, pp. 3–18.
16. Goldberg, S., Naor, M., Papadopoulos, D., and Reyzin, L. Nsec5 from elliptic curves: Provably preventing dnssec zone enumeration with shorter responses. *IACR Cryptology ePrint Archive 2016* (2016), 83.
17. Heran, M., and Spilman, J. Bitcoin contracts. https://en.bitcoin.it/wiki/Contract, 2012.
18. Micali, S., and Rivest, R. L. Micropayments revisited. In *Cryptographers' Track at the RSA Conference* (2002), Springer, pp. 149–163.
19. Pass, R., and Shelat, A. Micropayments for decentralized currencies. In *CCS* (2015), ACM, pp. 207–218.
20. Poon, J., and Dryja, T. The bitcoin lightning network: Scalable off-chain instant payments. *Technical Report (draft)* (2015).
21. Rivest, R. L. Electronic lottery tickets as micropayments. In *International Conference on Financial Cryptography* (1997), Springer, pp. 307–314.
22. Rivest, R. L. Peppercoin micropayments. In *International Conference on Financial Cryptography* (2004), Springer, pp. 2–8.
23. Wheeler, D. Transactions using bets. In *International Workshop on Security Protocols* (1996), Springer, pp. 89–92.