Acebusters
A real-time Poker Application on the Ethereum Blockchain

Johann Barbie
Email: johann@acebusters.com

***Abstract*—We present a payment channel concept for up to 10 participants and an efficient protocol to rotate in and out of this multiparty state channel including secure cash distribution. Of this, a busy cash-game poker table with players sitting down and standing up is a prime example.**

Our protocol has an initial phase where a participant interacts with the blockchain to join a state channel. After that, new and previous participants need to communicate only with each other over the course of many message exchanges. At any time a participant can signal intention to leave the state channel and eventually rotate out from the channel without interrupting the ongoing message flow or affecting the security of funds.

We describe the stateful contract model and the security notions that our protocol accomplishes. Moreover, we provide a reference implementation as smart contract in Solidity.

In comparison to existing payment channels, our multiparty state channels are more efficient and have additional features.

Keywords*—*state channels, smart contract, blockchain, poker.

I. Introduction

In 2011, Poker Stars and Full Tilt Poker, the biggest poker rooms at the time, were shut down due to bank fraud and money laundering. Players were unable to access their funds, and the day is remembered as ”Black Friday”[1]. This and other scandals make players question the trust model of a traditional online poker rooms. The poker room operators manage funds on behalf of users as well as shuffle and keep the cards secret, hence, posing as a trusted third party.

Shamir et al. [2] introduced mental poker, a set of cryptographic problems to play a fair hand of poker over distance without a trusted third party [3]. This work gave rise to a subfield of cryptography called multiparty computation. The latter enables parties to jointly compute a function over their inputs while keeping those inputs private [4]. This could be the solution for a provably fair shuffle guaranteeing the secrecy of the cards without a need of a trusted third party. However, Cleve demonstrated that a fair multiparty computation without an honest majority is impossible [5].

The basic approach to managing funds and transfers without a trusted third party has been introduced by Satoshi Nakamoto as Bitcoin [6], a peer-to-peer electronic cash system. Blockchain technology like Bitcoin could have prevented the ”Black Friday”, if players would be able to use it to manage their funds. Unfortunately, interacting with a proof-of-work based blockchain like Bitcoin entails waiting times, which would make real-time games like poker prohibitively slow.

In recent years, the academic study of decentralized cryptocurrencies gave rise to a line of research that seeks to impose fairness in multiparty computation by means of monetary penalties. Kumaresan and Bentov show a scheme in which the parties run an initial setup phase requiring interaction with the Bitcoin blockchain, but thereafter engage in many fair secure computation executions, communicating only among themselves for as long as all parties are honest [7].

It seems as if the mere performance optimization of blockchain could incentivise a fair execution of the game through multiparty computation as well as make the trusted third party redundant regarding the management of the players' funds. Bloomer and Nishimura introduce the first optimization in this regard to Bitcoin with payment channels, circumventing the timing restrictions of blockchains [8]. This initial proposal for payment channels is limited to only two participants.

Bentov et al. introduce efficient protocols for amortized secure multiparty computation in combination with smart contracts, allowing first real-time poker games [9].

To our knowledge all existing multiparty computation protocols require all participants to initiate the game at the same time. While all participants might be available at the start of a tournament, in cash-games participants can sit down or stand up from a table at any time.

A. Our Contributions

We extend existing schemes with the following features:

Asynchronous channel participation: all participants have the ability to join and leave the channel at any time without affecting message flow or security of funds.

Ability to rebuy: when a player runs out of money, he is kept in the channel and has the ability to rebuy by interacting with the blockchain.

As a consequence, we are the first to provide a working implementation that has an acceptable user experience for cash-game tables.

We do not present a complete integration of a multiparty computation protocol with the multiparty state channel. This is due to the high cost of verification [9] of the multiparty computation proof on the Ethereum blockchain, which makes an implementation impractical.

As a temporary solution before the implementation of multiparty computation, the evaluation of the hands is performed by an oracle. An oracle is an external service that has special permissions in the smart contracts to provide data.

II. Extending the State Channel

We explain the operation of a two-party state channel and then extend the functionality to multiple parties. A data structure that models the cash distribution after each hand is introduced. Lastly, a leave protocol is described that protects participants from accepting uncovered bets.

A. State Channels

State channels are a way to think about blockchain interactions which could occur on-chain, but instead get conducted off the chain, without increasing the risk of any participant. The most well known example of this strategy is the idea of payment channels in Bitcoin, which allow for instant fee-less payments to be sent directly between two parties [10].

The life-cycle of a state channel can be broken down into steps as depicted in figure 1:

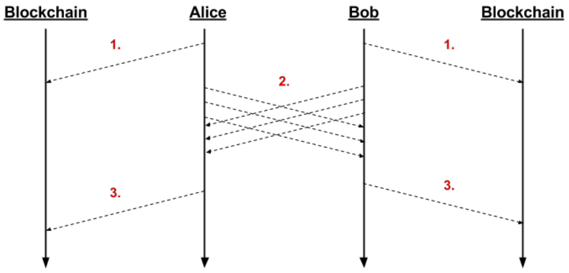

Fig. 1. A state channel netted by settlement.

1) Alice and Bob lock up some tokens by sending a transaction to the payment channel contract.
2) They can internally transfer value by signing receipts about new commitments and exchanging them. The receipts have to carry increasing sequence numbers and signatures.
3) Alice and Bob can release the bonded tokens by mutually agreeing (signing) on a resolution.

Several reasons might hinder Alice and Bob from agreeing on a resolution, like malicious intentions or network connectivity. In this case the channel is closed through dispute, as depicted in figure 2.

1) Alice and Bob lock up some tokens by sending a transaction to the payment channel contract.
2) They can internally transfer value by signing receipts about new commitments and exchanging them. The receipts have to carry increasing sequence numbers and signatures.
3) If Alice or Bob stop cooperating, the latest receipt can always be submitted to the blockchain. This will trigger a challenge period.
4) During the challenge period each participant is incentivised to submit the receipt for the highest commitment of his opponent.

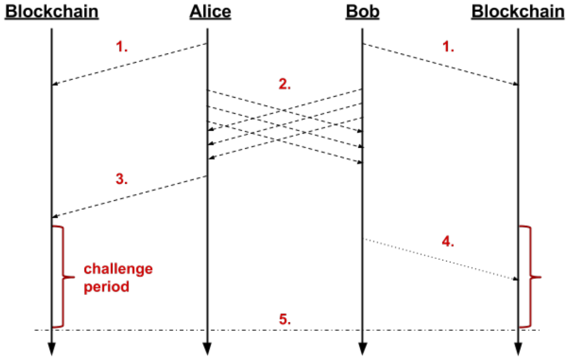

Fig. 2. A state channel netted after dispute.

5) After the challenge period expires, the bonded tokens are released to Alice and Bob by the ratio of the receipts with highest sequence numbers.

In effect, a state channel buffers the operations that would otherwise be written to the blockchain. Using a state channel decouples an application from the liveliness constraints of the blockchain without relaxing the security assumptions.

B. multiparty State Channels

An interaction of multiple participants could be modeled as a set of bilateral state channels from each player to each player. Yet, as the distribution of cash is not known beforehand, each channel would need to be funded with the full amount, resulting in a requirement of security deposits of $O(n^2)$ tokens.

To be able to operate the channel with multiple parties and $O(n)$ size of deposits, we define a commitment data structure as shown in figure 3.

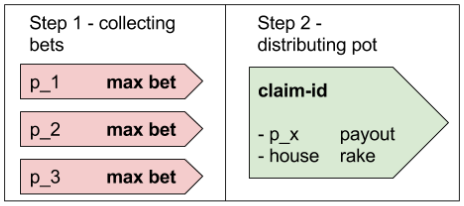

Fig. 3. Commitments and distribution

Commitments by participants carry a value, but do not have a specific recipient. Each higher value commitment by a participant replaces a lower one. When results of secure computation become available an oracle evaluates the data and issue a corresponding distribution receipt, reassigning the values from the participants' commitments to the winner(s) deducted from the multiparty computation.

Multiple rounds of commitments and distributions (hands played) can happen among the participants of the multiparty state channel. These rounds are numbered by a strictly increasing identifier (handId). Figure 4 depicts that, as long as all parties agree on a final settlement, the protocol is equivalent to the two-party channel:

1) Alice, Bob and Charlie lock up some tokens by sending a transaction to the multiparty state channel contract.

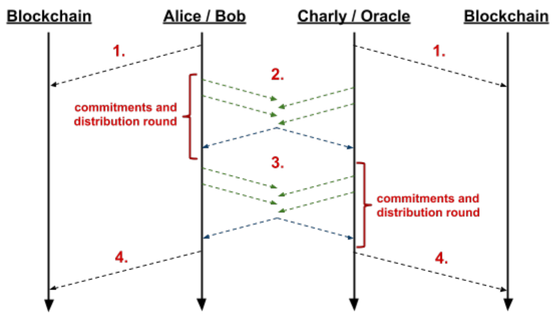
Blockchain
Alice / Bob
Charly / Oracle
Blockchain
1.
1.
2.
commitments and distribution round
3.
commitments and distribution round
4.
4.

Fig. 4. A multiparty state channel netted through settlement.

2) A round of commitment and distribution executes. In the round, the participants repeatedly bet value by signing receipts about new commitments and broadcasting them. The round is closed by the distribution receipt, which awards the committed value to the winner(s).
3) participants can have multiple further rounds of commitment and distribution, as long as at least 2 participants stay solvent.
4) Alice, Bob and Charlie mutually agree on a resolution. The resolution sums up the value transferred over all rounds performed since the start of the channel. The smart contract evaluates the resolution and updates the balances of all participants.

Whenever the balances are update also the latest handId is updated in the state of the contract. This process is called netting. The netting can happen when all parties agree through resolution, or when there is disagreement, through dispute. The dispute is executed as depicted in figure 5.

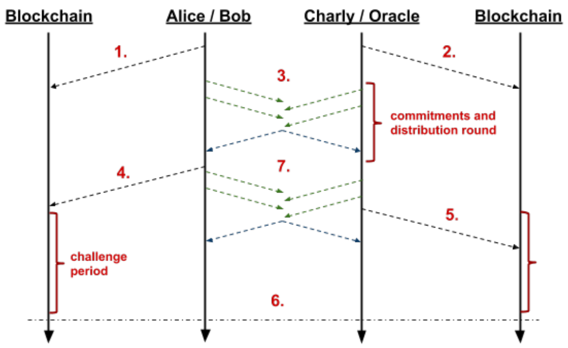

Fig. 5. A multiparty state channel netted after dispute.

1) Alice, Bob lock up some tokens by sending a transaction to the multiparty state channel contract.
2) Charlie can join the channel later by also sending a transaction to the contract and also locking up some tokens.
3) A round of commitment and distribution executes. In the round, the participants repeatedly bet value by signing receipts about new commitments and broadcasting them. The round is closed by the distribution receipt, which awards the committed value to the winner(s).
4) Bob signals to leave the state channel, but is unable to sign the resolution due to network issues. The challenge period is triggered in the contract.
5) The participants submit receipts of commitments and distributions to the contract.
6) The smart contract sums up the receipts over all rounds since the last netting and updates the balances of all participants. Once the netting progresses beyond the handId at which Bob signaled leaving, he is payed out and removed from the contract.
7) During the challenge period remaining solvent players can continue to execute rounds of commitments and distributions.

C. Asynchronous participation

Rotation into a state channel requires no special protocol, once the deposit is placed, the participant can be interacted with.

Rotation out of a channel has the potential risk that a remaining party could be handed uncovered commitments by the leaving party after it has withdrawn its deposit already. To guard against this situation, we extend the on-chain state with the variables:

lastNettedHand (LNR): describes the id of the last hand at which the channel has been netted. this is also the hand at which the balance of the participant is valid.

exitHand: each participant maintains this flag, which is initialized with 0 and stays unchanged until the participant signals to leave. At that point the participant requests a leave receipt from the oracle, which identifies the latest hand id in the channel. This receipt is submitted to the chain and sets the exit Hand of the participant.

NettingRequestHand (NRH): This variable tracks the id of the latest hand that has completed in the state channel. Yet, this variable is only updated in the contract if a participant of the state channel requests to leave.

NettingRequestTime (NRT): This variable is set to the current time simultaneously to NRH. It will be used to to measure the challenge period if a dispute arises.

Table I shows the state transition of the channel from having an empty seat, to a player joining and playing, to netting and leave.

TABLE I. STATE TRANSITION OF SEAT AND TABLE.

	seat x				
	balance	exitHand	LNH	NRH	NRT
1) empty	0	0	y	y	0
2) join	100	0	y	y	0
3) play	100	0	y	y	0
4) request leave	100	z	y	z	999
5) netting	80	z	z	z	999
6) leave	0	0	z	z	0

Once the exitHand flag of a leaving participant is set in the contract, other participants can clearly judge to accept or reject receipts in the current round. When, either through agreement, or dispute, the channel is netted beyond a participants exitHand (LNH ¿= exitHand), the participant is payed out and removed from the channel.

III. Implementation in Solidity

Each poker table implements a multiparty state channel as a smart contract in Solidity. It consists of data declarations and public and private functions. public function can further be distinguished by constant functions, that only read from the contract storage, and non-constant functions, that can be invoked through transactions to change the state of the contract storage.

A transaction that invokes a contract function carries the function name and the arguments passed to the function. The transaction also contains a fee, which roughly corresponds to the computational cost of executing the operation in the smart contract.

In appendix A, we provide our reference implementation of smart contract code for multiparty state channels.

A. Data Structures

In addition to the state variables (LNR, exitHand, NRH, NRT), which have been defined in the previous section, two important structures are Seats and Hands, which we will examine now.

Each table holds an array of 2 to 10 seats. Each seat can hold only one player at a time, no lap dances allowed. Each Seat element is implemented as a struct with the following 4 attributes:

```
struct Seat {
  address senderAddr;
  address signerAddr;
  uint256 amount;
  uint256 exitHand;
}
Seat[] public seats;
```

senderAddr: The sender address is the address of the wallet used by the player to join the table and holding the player's bankroll.

signerAddr: The signer address is the address of an additional private key created and used under the control of the player to sign receipts about commitments during betting.

amount: The amount stores the value of tokens that the player chose to transfer to the state channel to be used in bets during the game.

exitHand: The exitHand parameter is used to signal the intention to leave the state channel to other participants. Hence, it is kept at a value of 0, until the player signals to leave the channel. Once the player signals to leave the channel, the exitHand will take the id of the latest betting round executed in the state channel in which the player participated.

An additional data structure is maintained in the smart contract which is only used in the case of dispute. An array is used to collect all receipts of commitments and distributions for each betting round. Each Hand element is implemented as a struct with the following 3 attributes:

```
struct Hand {
  mapping (address => uint256) ins;
  uint256 claimCount;
  mapping (address => uint256) outs;
}
Hand[] public hands;
```

ins: This mapping holds the commitment value of the receipt with the highest sequence number for each player.

claimCount: The claimCount stores the sequence number of the distribution receipt.

outs: This mapping is filled by the distribution generated by the oracle at the end of each hand. It holds the value(s) of payment(s) for the winner(s) of this hand.

B. Functions

The contract implements the following functions:

tokenFallback(): This function allows new players to join the table by submitting tokens into the control of the smart contract. Already seated players can use this function to rebuy, increasing their balance at the table.

leave(): The leave function allows players to signal their intention to leave the table. This function can only be invoked by an active player once. It will trigger a netting request, notifying all other participants about the necessity to settle on the new balances at the end of the current betting round.

settle(): The settle function receives a resolution, which holds the new balances of all players at a specific hand. The resolution needs to be signed by all players and the oracle alike, to be valid and accepted into the smart contract. A valid resolution will trigger a netting, updating all players' balances and removing players that are signaling to leave the table, if the netting exceeded the players' exitHand ID.

submit(): The submit function can be called by anyone, receiving an array of receipts of commitments and distributions. Each receipt is evaluated by sequenceId and signature, then written into contract storage. Receipt are only accepted into contract storage if the state channel is in a dispute resolution state ($LNH < NRH$).

net(): The net function sums up all receipts of commitments and distributions that have been collected in the challenge period. It can only be invoked once the time of the challenge period expired.

C. Joining a Channel

With ERC20 a token standard has evolved that describes the functions and events for an Ethereum token contract to implement. With thransfer() token can be transferred one account to another. Unfortunately, ERC20 still has a couple of disadvantages, that hinder the interoperability with smart contracts that act as trust-free agents for escrow or exchange to the token, like our multi-party state channels:

1) A transaction depositing tokens into a smart contract requires two separate transactions: The depositor has to call approve() on token contract, only then the receiver contract can invoke transferFrom() to pull the tokens into his account.
2) Inability of handling incoming token transactions: ERC20 token transaction is a call of the transfer() function inside token contract. The ERC20 token contract is not notifying the receiver that a transaction occurred. Also there is no way to handle incoming token transactions for contracts and no way to reject any non-supported tokens.

These disadvantages would force users to send multiple transactions to join tables and would increase gas usage through excessive inter-contract communication.

We have adopted ERC223, which solves these disadvantages, by implementing a standard function to handle token transfers that is called from token contract when a token holder is sending tokens. This function works like the fallback function for Ether transactions:

```
function tokenFallback(
    address _from,
    uint256 _value,
    bytes _data
  );
```

The table contract implement this function to accept NTZ transaction when players join the table and maps the parameters as follows:

_from is the token sender
_value is amount of incoming tokens
_data the first 20 bytes carry the signerAddress. SignerAddress is the address of the private key that the players will use to sign receipts, .

By implementing the ERC223 token standard players can join tables by sending a single transaction, extending the interoperability of our poker tables to all ERC223 supporting wallets.

IV. Conclusion

The combination of standard secure computation and stateful contracts enables fair protocols that are highly efficient, since the interaction with the cryptocurrency network is kept to a minimum. The same is also true for secure cash distribution protocols, and in particular the efficient poker protocol that we provide. Stateful contracts can also be valuable for protocols that do not require secure computation, as demonstrated by our design of a duplex payment channel.

Appendix A
Proof of the First Zonklar Equation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Acknowledgment

The authors would like to thank...

References

[1] U.S. Treasury Department, "Unlawful internet gambling enforcement act," 2011. [Online]. Available: http://www.ots.treas.gov/_files/422372.pdf

[2] A. Shamir, R. L. Rivest, and L. M. Adleman, "Mental poker," *The Mathematical Garden*, pp. 37–43, 1981. [Online]. Available: https://people.csail.mit.edu/rivest/ShamirRivestAdleman-MentalPoker.pdf

[3] Wikipedia, "Mental poker." [Online]. Available: https://en.wikipedia.org/wiki/Mental_poker

[4] ——, "Secure multi-party computation." [Online]. Available: https://en.wikipedia.org/wiki/Secure_multi-party_computation

[5] R. Cleve, "Limits on the security of coin flips when half the processors are faulty," in *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*. New York, NY, USA: ACM, 1986, pp. 364–369. [Online]. Available: http://doi.acm.org/10.1145/12130.12168

[6] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: http://bitcoin.org/bitcoin.pdf

[7] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," *IACR Cryptology ePrint Archive*, vol. 2014, p. 129, 2014.

[8] B. Bloomer and T. Nishimura, "Bitcoin micropayment channels," 2014. [Online]. Available: https://www.ischool.berkeley.edu/sites/default/files/student_projects/finalprojectrepor

[9] I. Bentov, R. Kumaresan, and A. Miller, "Instantaneous decentralized poker." *CoRR*, vol. abs/1701.06726, 2017. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1701.html

[10] J. Coleman, "State channels," 2015. [Online]. Available: http://www.jeffcoleman.ca/state-channels/