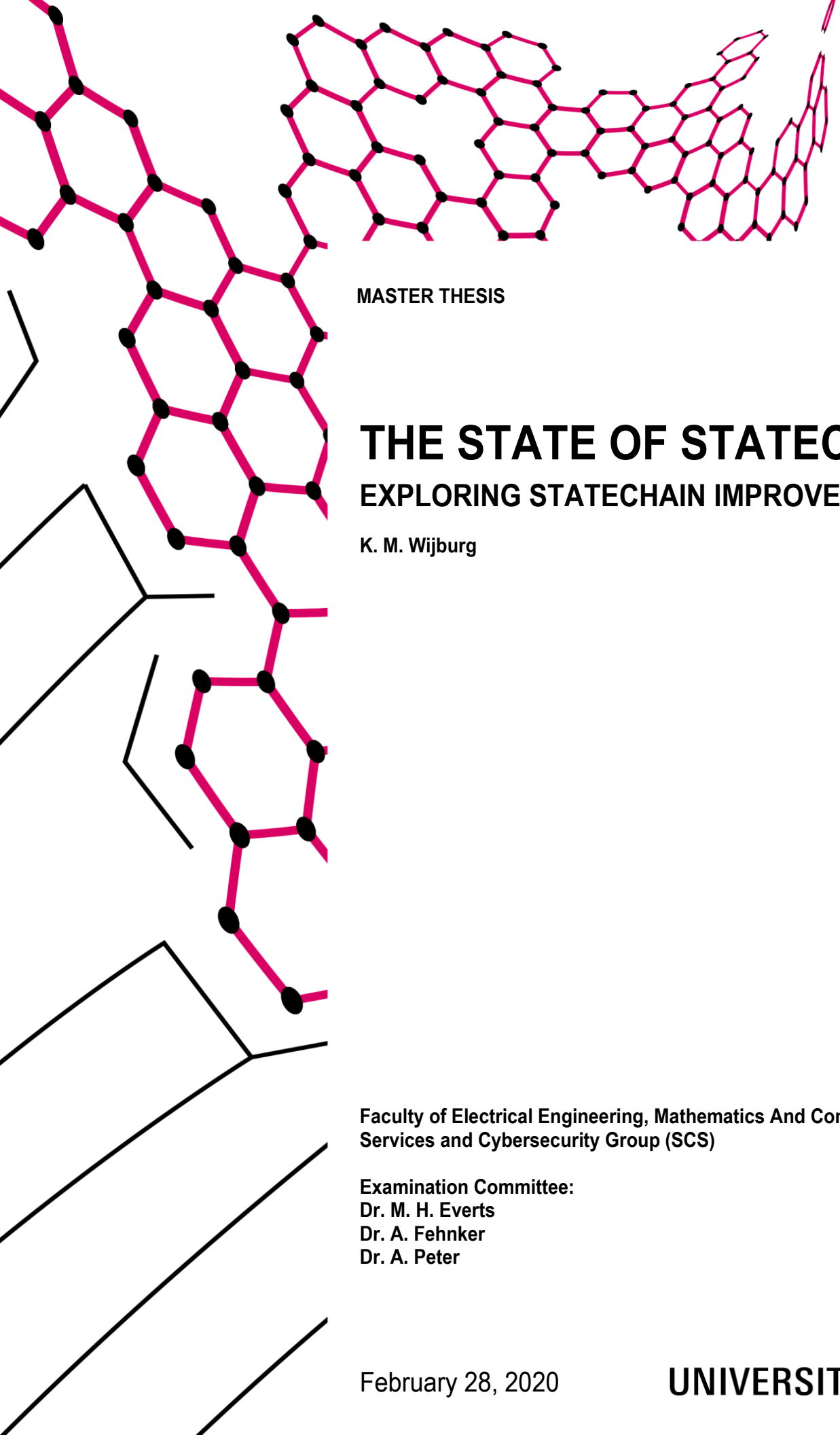

MASTER THESIS

THE STATE OF STATECHAINS

EXPLORING STATECHAIN IMPROVEMENTS

K. M. Wijburg

Faculty of Electrical Engineering, Mathematics And Computer Science (EEMCS)
Services and Cybersecurity Group (SCS)

Examination Committee:
Dr. M. H. Everts
Dr. A. Fehnker
Dr. A. Peter

February 28, 2020

UNIVERSITY OF TWENTE.

The State of Statechains

Exploring Statechain Improvements

Kasper Wijburg
University of Twente.
k.m.wijburg@student.utwente.nl

***Abstract*—Statechains are a second-layer solution for blockchains using a UTXO model. This paper poses a critical review of the Statechain protocol. It then explores State-accumulators as a possible solution to the scaling problems that Statechains suffer from, concluding that State-accumulators are a viable solution for the user side only. Additionally, the more recent blinded Statechain protocol is detailed and the Blinded Singular State Verification protocol is proposed as an alternative version that requires no unblinding operations.**

***Index Terms*—Blockchain, Statechains, State-accumulators, BSSV, Bitcoin, Second layer solution, UTXO**

I. Introduction

Over the last few years an increasing portion of purchases have been made over the internet [37]. These purchases are almost exclusively paid for using methods which are dependent on financial institutions to process the transactions. This results in a system which has the inherent weaknesses of requiring trust, having entities as single points of failure, and a complete lack of privacy.

In an attempt to solve the issue of privacy in digital payments, in 1982 David Chaum introduced eCash [12], [13]: a digital payment method which used cryptography to provide a high degree of privacy. However, this system was still reliant on a centralized institution, and failed to gain widespread adoption. Years later in 1998, Wei Dai created a concept for an "anonymous, distributed electronic cash system" called b-money [14]. This concept contained two proposals. The first proposal introduced proof-of-work [21] as a means for creating money. However, this proposal was seen as unimplementable due to its requirement of an unjammable, synchronous broadcast channel. In the second proposal, only a subset of the participants were required to keep accounts of the network, which would then be checked by all participants. However b-money was never implemented.

Then in 2008, Satoshi Nakamoto published a paper in which he proposed a new peer-to-peer electronic cash system: Bitcoin [26]. The protocol solved the issues of trust and having a single point of failure, however it only provided privacy in the form of pseudonymity. Additionally Bitcoin suffers from scalability issues, as well as requiring the passage of time before a transaction could be considered final with an acceptable degree of certainty.

Several solutions to the problems plaguing Bitcoin have been suggested. A prominent approach used in these solutions is to move as many transactions as possible off of the Bitcoin blockchain, while maintaining the advantages that Bitcoin provides. This type of solution is commonly referred to as a second layer solution.

Statechains [34] are a second layer solution that reduce the load on a blockchain by performing transactions through the off-chain transfer of an on-chain unspent transaction output (UTXO) [3]. They were first introduced by Ruben Somsen [34] and presented at *Scaling Bitcoin 2018 "Kaizen"* [35], where he highlights their functionality, some of their use cases, and several points of improvement for Statechains. Subsequently a blog post was written detailing a blinded Statechain which improves the privacy of the protocol [33]. However, both the original paper and the later blog post fail to critically assess the exact advantages and disadvantages of Statechains, both in a vacuum and when compared to other second layer solutions. This paper aims to provide such an assessment, as well as an exploration into several alternative versions of the Statechain protocol.

The paper is structured in the following manner. First, substantial background information is provided to ensure that Statechains, as well as the assessment and exploration, can be fully understood with only limited pre-existing knowledge in the field. We then provide a formal explanation of Statechains, including a detailed assessment of their use cases in the current blockchain environment. Emphasis is placed on a comparison with Lightning, the most prominent second layer solution at the time of writing this paper. Next, we present State-accumulators, a possible solution to the scalability problem of Statechains, where a onetime increase in storage space for the facilitating entity results in a constant size in storage space for the users. Then, the existing blinded Statechain protocol where the Statechain entity functions as a signing server is detailed. As an alternative, we present the Blinded Singular State Verification protocol (BSSV), a secure blinded Statechain where only the most current recent state is verified. Lastly, we conclude with several suggestions for future research to further improve the Statechain protocol.

II. Background

A. Blockchain

Blockchain technology is the driving force behind Bitcoin and other modern cryptocurrencies. In essence, a blockchain

is a distributed ledger which permanently records transactions between participants in a verifiable manner.

Blockchains can be public or permissioned. The former allows anyone to join the network of participants and partake in the blockchain, whereas the latter can only be accessed by authorized participants. Therefore, permissioned blockchains are not necessarily trustless. Since the focus of this paper is on improving Statechains in a trustless manner, all further mention of blockchain implicitly refers to public blockchains, though Statechains are not inherently incompatible with permissioned blockchains.

As the name suggests, a blockchain consists of blocks which are cryptographically chained together by having each block contain a hash of the previous block. In addition to this hash, a block also contains a timestamp and transaction data. The hash of the previous block causes the data to become permanent, since any change within a block would require all the subsequent blocks to be altered. Additionally, since a known hashing algorithm is used, all participants in the network can verify that the current state of the blockchain could only be reached with the provided set of transactions.

By itself, this construction only results in a permanent and verifiable system, as each participant in the network can provide their own transaction history and the corresponding blockchain. Therefore, a consensus mechanism is needed for the participants to agree on the current state of the blockchain, thereby enabling trustless and distributed storage. At the time of writing, the most popular consensus mechanisms for blockchain are proof-of-work (PoW) and proof-of-stake (PoS).

Within a PoW consensus protocol, participants aim to solve a puzzle. The first participant to successfully do so is allowed to create the next block. A requirement for such a puzzle is that it must be easy to verify compared to how difficult it is to solve. This allows other participants within the network to quickly validate the provided solution by the first solver.

On the other hand, within a PoS consensus protocol, the next participant allowed to create a block is randomly elected. The chance for a certain participant to be elected is based on their stake within the blockchain, for example the amount of cryptocurrency they hold. The theory behind this concept is that a participant with a high stake in the blockchain will act honestly, since if any cheating was detected the value of the stake would plummet. This construction enables the possibility to split transaction types that will never be dependent on one another into separate ledgers, thereby improving scalability without weakening the security [17]. This is not possible with PoW, as it would require the splitting of the computational rate of the network, resulting in multiple ledgers with weaker security. However, it can be argued that PoS is overall less secure than PoW, as it is considered more expensive to acquire 51% of the computational power within a network than it is to acquire 51% of the currency [20].

Through the addition of the consensus mechanism, a blockchain can function as a distributed ledger that does not suffer from the requirement of trust. However, for a blockchain to function as a monetary system, an additional feature is necessary: the ability to record ownership of values within the blockchain. There are two common approaches to implementing this feature within cryptocurrencies: the unspent transaction output (UTXO) model [3], and the account based model [4].

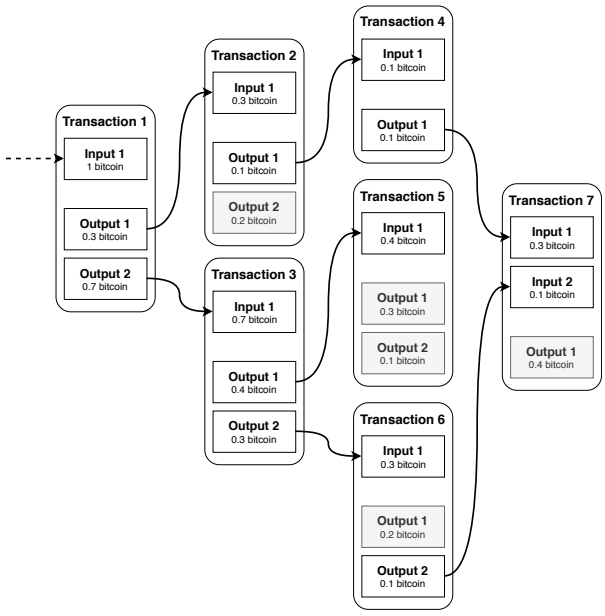

Fig. 1. A visualization of the UTXO model. The grey blocks are outputs which have no yet been spent and can be used as inputs for new transactions.

In a UTXO model, each transaction results in one or more outputs. Each of these outputs can be used once as an input for a later transaction by providing the private key that corresponds to the public key with which the output was locked. Each output that has not yet been used as an input is considered unspent (Figure 1). The value a person owns on a blockchain can be calculated by summing up all the unspent outputs for which they hold the private key.

The account based model on the other hand, simply stores a list of accounts and their corresponding values. When a transaction occurs from account A to account B, the amount transferred is subtracted from account A's value and added to account B's value. Similarly to the UTXO model, a private key that corresponds to the public key with which the account is locked, must be provided in order to spend from the account.

By combining the chain of blocks with the consensus mechanism and the ability to determine ownership of funds, blockchain technology appears to solve all the problems plaguing traditional digital currency systems. However, if naively implemented, blockchain technology suffers from several issues.

Firstly, the requirement to be able to validate all previous transactions requires the sharing of all transactions that have occurred in the history of the blockchain. This enables

any participant to trace the flow of cryptocurrency through pseudonymous addresses, thereby greatly increasing the risk of privacy leaks. Additionally, since the blockchain is ever growing, so is the amount of storage space required to store the entire transaction history, as well as the time required for a new participant to verify the blockchain, resulting in scalability problems.

Another issue is that transactions within a blockchain cannot be considered final immediately. Once a block is added to the chain it is quickly propagated through the network. Other participants validate the block and then add it to their own version of the blockchain. The older a block in the chain is, the more references it has by subsequent blocks, and therefore the less likely it is to be altered. Since recent blocks have fewer references by subsequent blocks, they can be overwritten more easily, either by malicious entities or honest participants that are out of sync. This creates the requirement for the passage of time before a block, and therefore the transactions within it, can be considered final with an acceptable degree of certainty.

Lastly, to facilitate features such as second layer solutions, the ability to program on the blockchain is necessary. Many blockchains include this feature, though it is not inherent to the concept of a blockchain. Within Bitcoin, this is limited to scripting which enables the specification of requirements for a transaction to be considered valid. Other blockchains, such as Ethereum [7], have opted to include a Turing complete programming language, enabling the creation of smart contracts on the blockchain.

B. Atomic Swap

An atomic swap is a protocol in which two or more parties perform an exchange atomically. A transaction is considered atomic if it either occurs completely or not at all. In a blockchain environment a transaction is considered atomic if the partial occurrence of a transaction only results in a loss for the party whose negligence resulted in the halt of the transaction, thereby enabling a trustless exchange between different blockchains. This can be achieved through hashed timelock contracts (HTLC). As the name suggests, an HTLC contains a timelock and a hash. When cryptocurrency is locked with an HTLC, it can be spent in one of two ways. Either the preimage of the hash is provided, or the time on the lock expires. The following steps detail an atomic swap where Alice is exchanging her bitcoin[1] with Bob's dogecoin [27]. All communication between Alice and Bob occurs off-chain through a secured channel, a visual representation can be found in Figure 2:

1) Alice and Bob agree on the amounts they are going to exchange. Alice provides Bob with her Dogecoin address, and Bob provides Alice with his Bitcoin address.
2) Alice uses secret value 'x' to generate $H(x) = X$, where H is a hash function. Bob does not know x.

[1]Many cryptocurrencies use the same name for their blockchain and a single unit of value, or coin, on this blockchain. Throughout this document, coins will be written with in all lower case letters, whereas the cryptocurrency and its blockchain will start with an upper case letter.

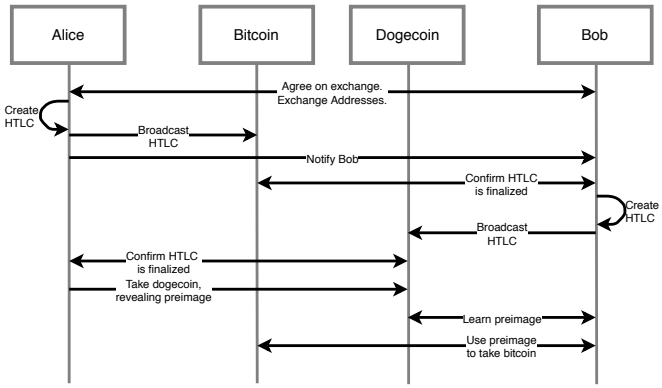

Fig. 2. Atomic swap protocol.

3) Alice creates an HTLC, using X for the hash, and a timelock which accounts for the finality time of the blockchains. She stores the bitcoin which is meant for Bob within the HTLC. If the preimage x is provided, the bitcoin can be sent to Bob's Bitcoin address. If the timelock expires the bitcoin can be sent back to Alice's Bitcoin address.
4) Alice broadcasts the HTLC to the Bitcoin blockchain and sends the transaction id (txid) to Bob.
5) Bob confirms that the HTLC is on the Bitcoin blockchain and is locked with X. He waits until the HTLC is finalized within the blockchain.
6) Bob creates an HTLC, using X for the hash despite not knowing x, and a timelock which is significantly shorter than that of Alice's HTLC. If the timelock is not significantly shorter, Alice can simply wait for the last second to take Bob's dogecoin and then reclaim her own bitcoins through the timelock in her HTLC. He stores the dogecoin which is meant for Alice within the HTLC. If the preimage x is provided, the dogecoin can be sent to Alice's Dogecoin address. If the timelock expires the dogecoin can be sent back to Bob's Dogecoin address.
7) Bob broadcasts the HTLC to the Dogecoin blockchain and notifies Alice he has done so. This completes the setup stage.
8) Alice waits for the HTLC on the Dogecoin blockchain to be finalized. She then uses her secret x to take the dogecoin, and in doing so puts x on the Dogecoin blockchain.
9) Bob reads x from the Dogecoin blockchain and uses it to take the bitcoin.

C. Scriptless Scripts

In 2016 a paper was dead-dropped on the *#bitcoin-wizards* IRC channel, detailing Mimblewimble [5], [22], a novel blockchain concept providing significantly better scaling than Bitcoin. However, in order to achieve this scalability, Mimblewimble made the trade-off of not supporting scripting of any kind, thereby severely limiting its practical use. To solve this issue, the concept of scriptless scripts was introduced [28].

Scriptless scripts are a method of embedding scripts within signatures. They ensure that signatures for transactions can only be created if the script is executed faithfully. Although scriptless scripts were originally developed for Mimblewimble, they can be used by any blockchain that uses a signature scheme that allows for linear operations. This includes the ECDSA signatures used by Bitcoin [25]. Using scriptless scripts in a blockchain such as Bitcoin, which already supports scripting, provides advantages with regards to privacy and scalability.

Scripts can be quite large. Since nodes on a blockchain network are required to validate all transactions, all nodes are required to download these scripts and validate them. With scriptless scripts, the script is part of the signature, therefore only the signature needs to be downloaded and verified. For example in a multi-party setting where the signatures of n participants are required for funds to be spent, using normal scripts n signatures would need to be validated. In a scriptless script setting, a single signature, that can only be created through the cooperation of n participants, would need to be validated. Due to the reduction in transaction size this results in improved scalability.

When normal scripts are executed on the blockchain, they are recorded there permanently. The details of the script can be analyzed by anyone and possibly be used to link transactions and break privacy. With scriptless scripts, the script is hidden within the signature, and only this signature gets recorded on the blockchain. This makes a scriptless transaction indistinguishable from an ordinary transaction.

Scriptless scripts are enabled by adaptor signatures, which are essentially partial signatures which prove that the receiver will learn certain information upon receiving the full signature [29]. The following is an example of adaptor signatures within the Schnorr encryption scheme which has linear properties by default.

C.1 Adaptor Signatures: Schnorr signatures are a form of elliptic curve cryptography. In the following formulas, lowercase letters represent numeric values and uppercase letters represent points on the curve, with the exception of H which is used for a hash function.

For a message m, with secret key x, random nonce r, and base point G on the elliptic curve, a Schnorr signature is created in the following manner:

$$\begin{aligned} P &= x \cdot G \\ R &= r \cdot G \\ e &= H(P||R||m) \\ s &= r + e \cdot x \end{aligned}$$

In this construction, P is the public key and R is the public representation of the nonce used for the transaction, note that x and r are difficult to calculate from P and R respectively, under the discrete log problem. The final signature (s, R), consists of both the signature and the public randomness, enabling verification that a specific message was signed by the owner of a specific public key.

In a situation where Alice wants to send value t to Bob by publishing a Schnorr signature, the signature scheme can be altered by adding the second, meaningful value t to r. Since this simply results in another random value it does not compromise the security of the signature or alter the construction.

$$\begin{aligned} (t+r) \cdot G &= t \cdot G + r \cdot G \\ T &= t \cdot G \\ e &= H(P||R+T||m) \\ s &= r + t + e \cdot x \end{aligned}$$

Alice now calculates the adaptor signature s', by leaving out t:

$$s' = r + e \cdot x$$

Note that T is still used in the calculation of e, and that s' is not a fully valid signature as t has been left out. Alice now sends (s', R, T) to Bob. Using this information Bob can first calculate e, and then use the information to validate that s' is a valid adaptor signature, which will reveal t to him once he learns the valid signature:

$$\begin{aligned} s' \cdot G &\stackrel{?}{=} (r + e \cdot x) \cdot G \\ &= r \cdot G + e \cdot x \cdot G \\ &= R + e \cdot P \end{aligned}$$

Now when Alice reveals valid signature s, Bob can simply subtract s' from s to learn t:

$$s - s' = (r + t + e \cdot x) - (r + e \cdot x) = t$$

To an observer monitoring the blockchain, this is indistinguishable from a normal transaction, as all that is posted onto the chain is the final signature $(s, (R + T))$.

C.2 Scriptless Contingency Payment: In order to extend the functionality, this section explains adaptor signatures in a 2-of-2 setting. The following is the construction of a signature over a transaction that transfers funds from Bob to Alice. Through the use of an adaptor signature, Alice uses this transaction to atomically sell secret t to Bob. First Alice and Bob exchange public keys P_A and P_B, and the public nonces R_A and R_B to be used for the transaction. Additionally, Alice sends T to Bob, where $T = t \cdot G$. Then hash commitments to Alice and Bob's public keys are made in order to avoid related-key attacks [38]. These commitments are then used to create a joint key $J(A, B)$ for the transaction:

$$\begin{aligned} P'_A &= H(H(P_A||P_B)||P_A) \cdot P_A \\ P'_B &= H(H(P_A||P_B)||P_B) \cdot P_B \\ J(A,B) &= P'_A + P'_B \end{aligned}$$

Note that Alice holds the private key for P'_A and Bob for P'_B:

$$x'_A = H(H(P_A||P_B)||P_A) \cdot x_A$$
$$x'_B = H(H(P_A||P_B)||P_B) \cdot x_B$$

Alice then creates the adaptor signature for the transaction:

$$e = H(J(A,B)||R_A + R_B + T||m)$$
$$s'_A = r_A + e \cdot x'_A$$

Bob now validates that the adaptor signature was created honestly. If this is the case, he knows that he will learn t once the aggregated signature is released. Upon validating he therefore creates his own signature and sends this to Alice.

$$s'_A \cdot G \stackrel{?}{=} R_A + e \cdot P'_A$$
$$s_B = r_B + e \cdot x'_B$$

Alice can now combine Bob's signature with her own, creating the aggregated signature $s_{A,B}$ needed to sign the transaction:

$$\begin{aligned} s_A &= r_A + t + e \cdot x'_A \\ s_{A,B} &= s_A + s_B \\ &= r_A + t + e \cdot x'_A + r_B + e \cdot x'_B \\ &= r_A + r_B + t + e \cdot (x'_A + x'_B) \end{aligned}$$

$$\begin{aligned} s_{A,B} \cdot G &= (r_A + r_B + t + e \cdot (x'_A + x'_B)) \cdot G \\ &= r_A \cdot G + r_B \cdot G + t \cdot G + e \cdot (x'_A \cdot G + x'_B \cdot G) \\ &= R_A + R_B + T + e \cdot J(A,B) \end{aligned}$$

Alice can now claim Bob's funds by broadcasting the aggregated signature to the blockchain. Once it has been included in a block, Bob can read the aggregated signature and use the adaptor signature to gain knowledge of t:

$$\begin{aligned} s_A &= s_{A,B} - s_B \\ s_A - s'_A &= (r_A + t + e \cdot x'_A) - (r_A + e \cdot x'_A) \\ &= t \end{aligned}$$

Despite this being a 2-of-2 transaction, all that is posted to the blockchain is a single signature. Therefore to an observer monitoring only the blockchain it looks identical to a basic transaction.

C.3 Scriptless Atomic Swap: The concept in the previous subsection can easily be extended to perform a scriptless atomic swap. Here, two transactions are created, both of which are 2-of-2 and require a signature from Alice and Bob. Each transaction contains the funds that are being swapped, one sending its funds to Alice, and the other one sending its funds to Bob. Once again Alice knows secret t and shares T with Bob. The steps described in the previous section, up to and including the verification of the adaptor signature by Bob, are performed for both transactions.

Bob then sends Alice the signature which is required for her to complete the transaction which grants her funds. Upon adding her own signature, Alice obtains the funds by broadcasting the signature to the blockchain. Bob can now determine t from the completed signature and the corresponding adaptor signature.

Bob can then complete the transaction which grants him funds without requiring help from Alice, since he knows the adaptor signature corresponding to Alice's signature for the transaction, and t.

$$s_A = s'_A + t$$
$$s_{A,B} = s_A + s_B$$

Since the transaction addresses have different public keys, and different nonces, the signatures for both transactions are completely different. This makes the transactions completely unlinkable, since all that is posted to the blockchain is the single aggregated signature confirming the transaction.

D. Accumulators

This section details the concept of cryptographic accumulators, which are one-way membership functions [10]. Accumulators have been suggested as a replacement for the Merkle trees currently used within Bitcoin and several other blockchains, with the goal of improving scalability [11]. Using accumulators, membership of a set can be proven without revealing the individual members of the set. As an example, the following is a description of the functioning of an RSA accumulator.

In order to create an RSA accumulator, the following building blocks are required:

- RSA modulus N, where $N = p \cdot q$ for two unique primes p and q.
- Hash function H, which maps an arbitrary value to a prime.
- Group $\mathbb{G} = \{QR_n \setminus \{1\}\}$, where QR_n is the group of quadratic residues modulo N
- Accumulator initializer $A_0 \in \mathbb{Z}_N$

Now a value x can be added to accumulator A_i, by raising the accumulator to the hash of that value:

$$A_{i+1} = A_i^{H(x)} \mod N$$

Similarly a value x can be removed from the accumulator A_i, by raising the accumulator to 1 divided by the hash of that value. Efficient computation of this removal requires a trapdoor or knowledge of the full set.

$$A_{i+1} = A_i^{\frac{1}{H(x)}} \mod N$$

Membership of value x within A can now be proven by providing inclusion proof $\Pi \in \mathbb{G}$, where $\Pi = A^{\frac{1}{H(x)}} \mod N$, and showing that $\Pi^{H(x)} = A$. If $H(x)$ was not in A this construction is impossible unless the prover has access to the RSA trapdoor information [11]. Therefore, RSA accumulators require a trusted setup, complicating their use in a trustless system.

An accumulator such as this can be used in a blockchain setting to record the full set of UTXOs instead of a Merkle

tree. Currently full nodes in a blockchain using Merkle trees are required to store the entire set of UTXOs. This is due to both the membership verification of a UTXO and the updating of the Merkle tree requiring knowledge of all values in the set. Accumulators on the other hand, only require the UTXO, the proof, and the accumulator itself to prove membership, and can be updated without knowledge of the other members of the set. This offers major scaling advantages as full blockchain nodes are now only required to store the accumulator instead of the entire set of UTXOs.

Another advantage of accumulators is the constant size of their inclusion proof. Since the other values within the set are not required to prove membership, the size of the proof does not scale with the size of the set. Whereas in a Merkle tree, the size of the proof depends on the depth of the tree. When a Merkle tree has more than 4000 members, the inclusion proof for accumulators becomes faster than one for the Merkle tree. Additionally it is possible to aggregate accumulator inclusion proofs using Shamir's Trick, making it possible to create a single proof for every transaction within a block [11].

E. Lightning Network

One of the first solutions to solving the blockchain scalability problem is the concept of performing transactions that are not recorded on the blockchain; this is referred to as an off-chain solution. One of the first off-chain solutions is the Lightning Network [31]. This network consists of payment channels between participants. At the time of writing, Lightning has a functioning beta implementation available for use with the Bitcoin blockchain [6].

E.1 Lightning Channels:

To create a Lightning payment channel, two participants create a contract on the blockchain within which they lock some cryptocurrency. This contract requires the signature of both participants to be spent from. Before this contract is actually uploaded on the blockchain, a participant that provides funds for the channel creates a refund transaction and demands that the counterparty signs this transaction. It can then be used to reclaim the funds, which are now locked on the blockchain, in the case where the counterparty vanishes before any transactions over the channel have occurred.

Once the channel has been successfully set up, funds can be exchanged across it by shifting the balance on the channel. This balance shift is done by creating valid blockchain transactions which spend from the channel contract, but not broadcasting these transactions to the blockchain. For example, if Alice provided 1 bitcoin for the channel and Bob provided nothing, the refund transaction would spend 1 bitcoin from the contract to Alice and nothing to Bob. Now if Alice wants to send 0.5 bitcoin to Bob, they create a new transaction which spends 0.5 bitcoin from the channel to Alice and 0.5 to Bob. If Bob now wants to send 0.1 bitcoin to Alice, they create yet another new transaction which spends 0.6 bitcoin from the channel to Alice and 0.4 to Bob (Figure 3.).

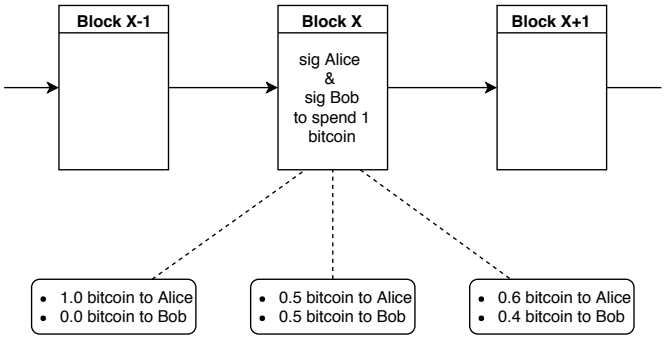

Fig. 3. Only the creation of the Lightning channel is recorded on the blockchain.

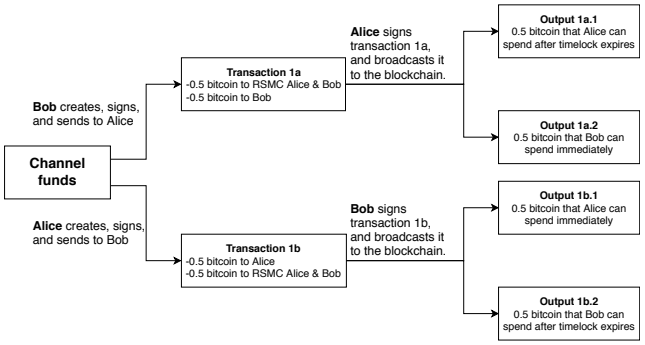

Fig. 4. Both participants have a transaction representing the current state, which they can upload without the cooperation of their counterparty.

These transactions need to be constructed in a way that either party can upload them to the blockchain, and dishonest behavior by broadcasting an older transaction is disincentivized.

The former can easily be achieved in the following manner: When the participants decide they want to alter the balance of the channel, both create the transaction and sign it, they then send their signed transaction to one another. In order to validate the transaction a participant can now simply sign the transaction already signed by their counterparty, and then upload it to the blockchain (Figure 4).

The latter requires a more complex construction which allows dishonest behavior to be punished. A participant needs a method to prove to the blockchain that their counterparty has behaved dishonestly by broadcasting an older transaction. This proof needs to occur before the cryptocurrency is spent from the contract, since the spending is irreversible. Therefore if the unilateral closing of a channel occurs, the party closing the channel must experience a waiting period before their cryptocurrency can be spent. This is accomplished by using the following construction: When a participant in the channel creates a new transaction, they ensure that the transaction output intended for their counterparty is a Revocable Sequence Maturity Contract (RSMC) [31]. This RSMC specifies that the cryptocurrency the output contains can be spent by the counterparty after a certain amount of time, or instantly by the participant if a Breach Rememdy Transaction (BRT) is pro-

vided. Such a BRT requires the signatures of both participants, and is provided to the counterparty for the previous transaction after a new transaction is created. The RSMC creates the waiting period for the counterparty to provide proof, and a valid BRT proves a previous transaction was uploaded. This construction is depicted in Figure 5.

When Alice and Bob decide to close the channel cooperatively they create a transaction which instantly pays both parties and sign it; this is then broadcast to the blockchain.

Using this protocol, Alice and Bob can rebalance the funds on the channel an infinite amount of times, in a trustless manner. Since only the creation and closure of the channel are required to be recorded on the blockchain, this provides great scaling benefits. Due to the fact that the transactions are fully valid blockchain transactions, the participants are guaranteed their bitcoin when they upload the transaction since no other source can spend from the contract. This means that the transactions can be considered final even before they are broadcast to the blockchain, allowing for instantaneous transfer of cryptocurrency.

Additionally, Lightning channels can be used to speed up atomic swaps, thereby improving blockchain interoperability. By creating two channels between them, one on each blockchain they wish to perform the exchange on, Alice and Bob can swap coins without having to wait for the finality of the transactions on the blockchain. Though they are still required to wait for the finality of the channel creation contracts to ensure that the on-chain funds are locked for use in the channel.

E.2 Watchtowers:

One downside of the construction described in the previous subsection, is that both participants of the channel need to continuously monitor the blockchain to ensure their counterparty is behaving honestly. In the situation where their counterparty broadcasts an old transaction, the participant needs to broadcast the BRT before the waiting period expires. This means the participant needs to constantly monitor the blockchain for old transactions and needs to be able to broadcast the BRT upon detecting the malicious behavior. Going offline for a period of time equal to or longer than the waiting period means risking your money being stolen. In order to solve this issue, the concept of channel watchtowers was introduced [8].

Watchtowers are third parties that are introduced to payment channels. They do not take part in the exchanging of funds but simply monitor the transactions that have occurred within the channel and check that no old transactions are uploaded onto the blockchain. If they are given the BRT they can broadcast it to the blockchain when they detect fraud.

Watchtowers do not necessarily infringe on privacy. The only data they need to receive from the channel in order to function, are the old transaction IDs, and optionally the BRT which corresponds to a specific transaction ID. The watchtower does not learn anything about the contents of the channel unless an old transaction is uploaded.

Continually monitoring the blockchain as well as storing transaction IDs cannot be done without incurring costs. Therefore incentives are required to motivate watchtowers to partake in a channel.

One possible business model is for watchtowers to charge a fee from the channel holders. However determining this fee can be very tricky. Since a channel can remain open indefinitely, and supports infinite transactions, the watchtower can end up indefinitely monitoring the blockchain and requires infinite storage space for transaction IDs. Therefore it would be required to continuously pay watchtowers while the channel persists as a form of subscription. However there is no way to ensure that the watchtower is actually performing the duties it promises, other than closing the channel with an old transaction.

Another possible solution is for BRTs to send a portion of the punishment funds to the watchtower that broadcast the BRT. This method ensures that watchtowers are only paid if they do their job properly. However, if watchtowers do their jobs properly, no rational channel participants would behave dishonestly, and therefore the watchtowers are not getting paid.

In order to solve these problems the concept of subscriptions and providing funding in the BRT can be combined. Instead of paying a watchtower a subscription fee directly, a channel participant creates a new fake channel with themselves on the blockchain. They then intentionally behave dishonestly by uploading an old transaction. Through this method, only watchtowers that are performing their functions properly are paid fees regularly. However, this method introduces numerous problems. Firstly, it increases the transaction count on the blockchain. Additionally, if watchtowers can keep track of which channel submits which transaction IDs they may decide to stop monitoring channels which have been open for a long duration of time. These channels are likely legitimate channels instead of ones used to pay watchtower fees, as creating a channel to pay a watchtower in the distant future is impractical as it requires the locking of funds. Lastly, there is no obligation for Lightning channel participants to pay watchtowers. It cannot be checked whether each participant that is using a watchtower's functionality is actually performing payments to the watchtower. This may cause participants to attempt to make free use of the watchtowers, possibly resulting in an unprofitable situation for the watchtowers, causing them to halt their services.

Furthermore, in a construction where a watchtower is paid for doing its job, the watchtower may be susceptible to bribery. A channel participant could offer the watchtower more money than the watchtower would get for submitting the BRT. To combat this possibility, the concept of a free market for watchtowers was proposed [8]. In this construction, old transaction IDs and their corresponding BRTs are broadcast onto a network of watchtowers instead of being given to a specific watchtower. This would require the dishonest party to offer each watchtower on the network a bribe greater than the reward for posting the BRT, quickly making bribery economically unfeasible. To combat watchtowers being part of

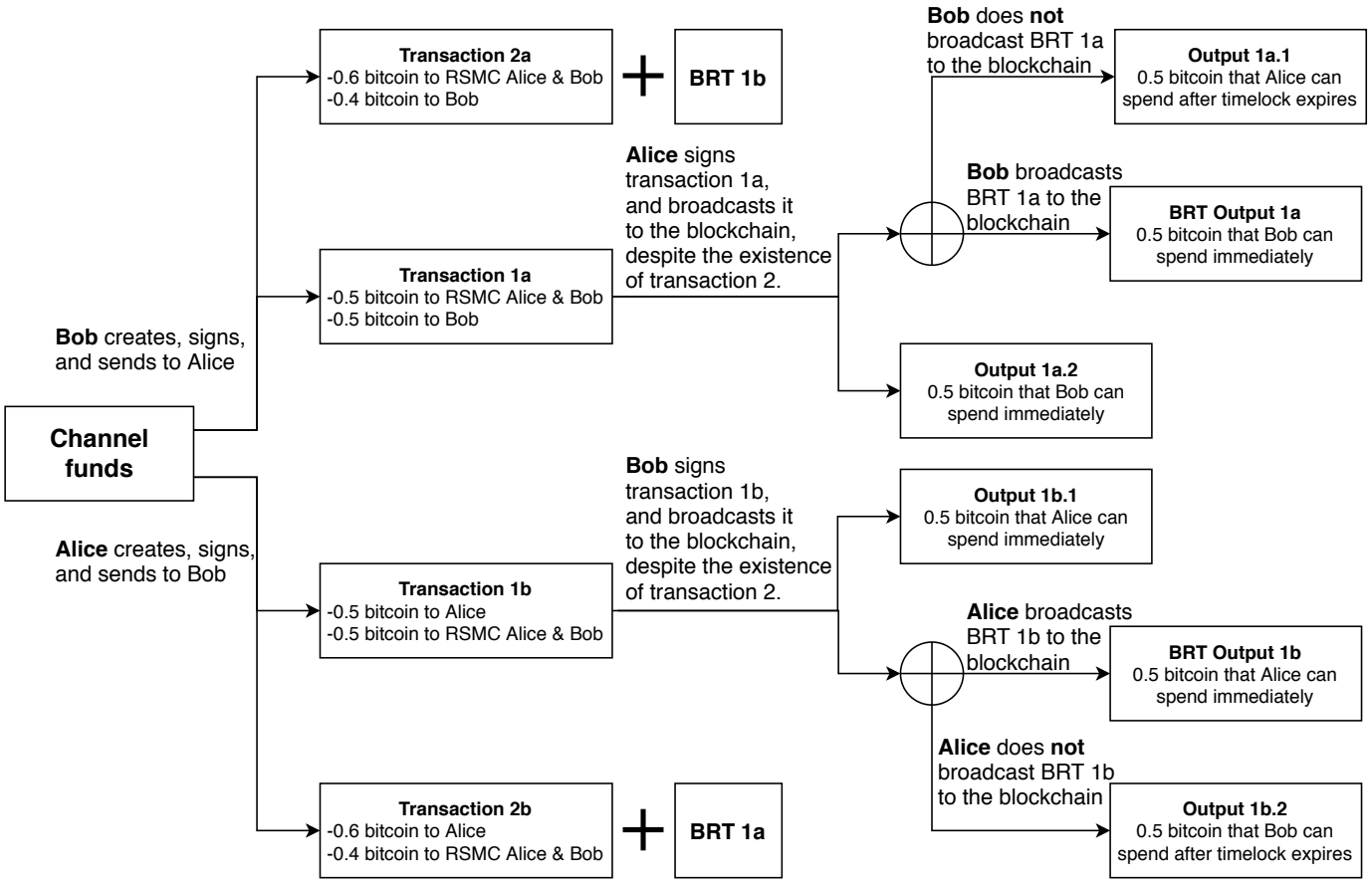

Fig. 5. A Lightning channel where two transactions have occurred. If a participant broadcasts the old state to the blockchain, the other participant can claim all the funds within the channel by broadcasting the BRT before the timelock expires.

the network but letting others do the actual work, the concept of BRTs granting payment to the watchtower that posts it is required. However the fastest watchtower is the only one that receives payment; in a situation where one watchtower vastly out performs the others it could quickly become unfeasible for others to perform their duties as watchtowers, resulting in a monopoly and reintroducing the issue of bribes.

E.3 eltoo:

eltoo is a drop-in replacement for the original state update mechanism used by Lightning, which maintains backward compatibility with the other parts of the stack of Lightning protocols [16].

In eltoo, every state of the channel is represented by a set of two transactions: an update transaction and a settlement transaction. The update transaction spends the contract's current output and creates a new output. The settlement transaction spends the newly created output and divides the funds according to the agreed-upon distribution. The outputs in eltoo have a script that allows a transaction to be directly attached to them, update transactions can be attached immediately, settlement transactions require a waiting period to have expired first. If an outdated settlement transaction is broadcast to the blockchain, before the waiting period has expired a newer settlement transaction can be broadcast. This new settlement transaction attaches to the output that the old transaction is attempting to spend. Since attempting to spend an output that has a transaction attached to it results in a double spend, the old settlement transaction is automatically invalidated.

By repeatedly invalidating the previous state, a long chain of transactions is created off-chain. However, this has the downside that broadcasting the final state to the blockchain requires the entire chain to be broadcast, effectively nullifying the scaling advantages provided by Lightning. To solve this issue, the eltoo protocol allows for the final update transaction to be directly linked to the channel creation contract. However, to enable this functionality the addition of a `SIGHASH_NOINPUT` flag to Bitcoin is necessary, as this allows transaction inputs to be bound to any transaction output with a matching script. In order to ensure that an update transaction cannot be replaced by an older update or spend transaction, the script checks that the state number of the new transaction is higher than the current one. Figure 6 details the format of an eltoo update script.

This construction provides fundamentally different tradeoffs compared to the classic Lightning channel construction. In eltoo there is no penalty for attempting to broadcast an older

```
OP_IF
  10 OP_CSV
  2 A_{s,i} B_{s,i} 2 OP_CHECKMULTISIGVERIFY
OP_ELSE
  <S_i + 1> OP_CHECKLOCKTIMEVERIFY
  2 A_u B_u 2 OP_CHECKMULTISIGVERIFY
OP_ENDIF
```

Fig. 6. The output script used for an eltoo update transaction.
Source: Adapted from [16].

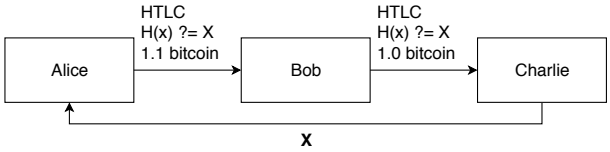

Fig. 7. Transferring 1 bitcoin from Alice to Charlie, through Bob.

channel state to the blockchain. Therefore there is no risk for malicious parties other than reputation loss. This may play an important role in the security of protocols that are built upon Lightning transactions. However, eltoo offers several functionality improvements over classic Lightning channels.

Firstly, eltoo allows for more advanced transaction constructions to be attached to the state of the channel. Instead of the channel contract being settled with two outputs, one for each participant, eltoo allows the channel to be settled with any number of outputs and more advanced constructions such as HTLCs.

Secondly, in the classic Lightning channel, a participant is required to keep an evergrowing list of BRTs, since they are required to prove the invalidity of an older transaction. Within eltoo a participant is only required to maintain the most recent version of channel state, since a more recent transaction automatically invalidates an older one.

Lastly, in a classical construction the fee to pay the miners upon broadcasting to the blockchain is determined upon the creation of the state. As the required fee for a transaction to be processed in a timely manner can shift vastly, it is undesirable for the fee to be set far ahead of broadcast time. By signing the update transaction outputs with SIGHASH_SINGLE[2], a participant that wants to broadcast the settlement transaction to the blockchain can dynamically add an extra input to the transaction, which can be used to provide a miner fee. This allows the participant to set the fee at the time of broadcasting, thereby being able to set a fee which is appropriate for the current state of the blockchain. This provides additional security benefits, as a current state with a too low fee may not be verified by the blockchain before the waiting period of a broadcasted outdated state has expired, allowing a dishonest participant to steal funds.

At the time of writing, a proposal to include SIGHASH_NOINPUT in Bitcoin has been submitted, but not yet accepted [15]. Therefore current implementations of Lightning use the construction as described in II-E.1, as eltoo can not yet be implemented. However, due to the benefits SIGHASH_NOINPUT offers, such as eltoo, it is likely it will be accepted in the near future.

F. Lightning Channel Networks

As described in the previous subsections, performing a Lightning transaction between Alice and Bob requires them to set up a channel on the blockchain first. This has the downside of needing to wait for the channel contract to be final on the blockchain, defeating the purpose of using Lightning for a transaction with a one-time counterparty. It may even make it unfeasible to use Lightning with every counterparty a participant regularly trades with, as it would require the participant to lock funds in contracts with each counterparty. A solution to this problem is using participants in Lightning channels as routing nodes [31].

In the situation where both Alice and Charlie have a Lightning channel open with Bob, but do not have a Lightning channel open with one another, they can still perform a Lightning transaction by routing funds through Bob. However, this in itself poses a problem. If Alice sends her money to Bob, there is no guarantee he will then forward it to Charlie. Likewise if Bob were to send funds to Charlie first, there is no guarantee he will be able to claim the funds back from Alice. Therefore a method is required which enforces the atomicity of the transaction. Once again HTLCs can be used to create this setup. Lets say Alice wants to send 1 bitcoin to Charlie. Charlie starts by generating a secret value x, he then hashes this value and sends the result X to Alice. Alice then uses this to create an HTLC using X as the hash, and locking 1 bitcoin plus 0.1 bitcoin fee for Bob's services as a routing node inside the HTLC. She then sends this HTLC to Bob and tells him Charlie knows the key to unlocking X. Bob then creates an HTLC using X as hash, and locking in 1 bitcoin of his own. He then sends this to Charlie. Charlie can take the bitcoin from Bob, since he knows x. In doing so, Charlie reveals x to Bob, which allows Bob to take the 1.1 bitcoin from Alice (Figure 7). Note that in order to ensure that Bob can claim the coins from Alice after giving his coins to Charlie, the timelock used by Alice must be longer than the timelock used by Bob.

This increases the on-chain privacy of the transaction, as the funds that are redeemed by Charlie on-chain could have been sent by any participant in the Lightning network. However, it naturally introduces privacy concerns off-chain, as Bob learns all the details of the transaction between Alice and Charlie. Therefore a routing protocol is implemented which makes use of onion routing [1], [32]. This ensures that each node that is routed through is only aware of the direct neighbors. However due to the HTLC construction, a global adversary or colluding nodes can correlate traffic through the use of the same hash X.

A multi-hop transfer can also be constructed using scriptless scripts, by combining the concepts described in II-C.2 with the concept described in this section. The HTLC transactions

[2]https://en.bitcoin.it/wiki/OP_CHECKSIG#Procedure_for_Hashtype_SIGHASH_SINGLE

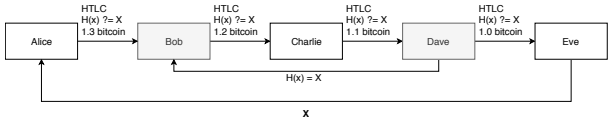

Fig. 8. A wormhole attack where Bob and Dave collude to steal the fee meant for Charlie.

used in the Lightning transfer are replaced by scriptless 2-of-2 transactions. The preimage x with hash X are replaced by secret t and curve point T. Upon setup, Charlie generates t and sends T to Alice. Alice then creates a scriptless contingency payment with Bob, requesting that Bob uses T as part of his signature. Bob now creates his adaptor signature and sends it to Alice. Once she has verified that it is valid, she sends her signature for the transaction to Bob. However since neither Alice nor Bob knows t, they cannot complete this transaction. Therefore Bob creates another scriptless contingency payment with Charlie, once again using T as part of the aggregated signature, he then requests that Charlie sends his adaptor signature. Bob verifies that the adaptor signature is valid and sends Charlie his signature. Since Charlie knows t he can construct the aggregated signature, thereby releasing the information Bob needs to determine t. This knowledge can then be used by Bob to complete the transaction between him and Alice.

Using basic scriptless scripts instead of standard script HTLCs allows us to mask the lock value X from a global adversary, but not from colluding nodes. By introducing new aspects to the protocol using scriptless scripts, the connection can be masked even from colluding nodes [24].

Knowledge that they are both part of the same transaction allows colluding nodes in the route to perform a wormhole attack [24] to steal the fees of intermediary nodes. Let's say Alice is sending 1.0 bitcoin to Eve, and routing this transaction through Bob, Charlie, and Dave, paying each of them 0.1 bitcoin for their services. However, Bob and Dave are colluding to steal Charlie's fee. When Eve takes her 1.0 bitcoin from Dave, Dave passes the secret x on to Bob, and simply lets his HTLC with Charlie expire, claiming his HTLC with Eve also expired; this makes it impossible for Charlie to claim the 1.2 bitcoin from Bob. Bob can now claim the 1.3 bitcoin from Alice using secret x, resulting in a net loss of 1.0 bitcoin for Dave and a gain of 1.3 bitcoin for Bob (Figure 8). Overall the colluding nodes are up 0.3 bitcoin, compared to the 0.2 bitcoin they would have gained from fees had they behaved honestly.

III. Statechains

Statechains [34] are an off-chain solution for blockchains that use UTXOs to determine cryptocurrency ownership. Instead of performing a transaction to transfer funds on the blockchain, Statechains perform off-chain transfers of UTXO ownership. In order for a Statechain to be usable on a blockchain, the blockchain must support eltoo [16] style channels, and must have a method for adaptor signatures to be

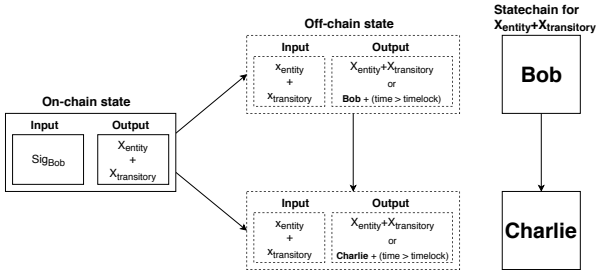

Fig. 9. Transferring ownership of the UTXO from Bob to Charlie, and updating the Statechain.

constructed. The examples in this section make use of Schnorr signatures, as they can easily be used to construct adaptor signatures.

A Statechain is maintained by a Statechain entity, that uses the Statechain as a public ledger to record the history of all the UTXOs under its management, and may receive fees for the services it provides. The role of a Statechain entity can be fulfilled by a single actor. However in order to provide better decentralization, the entity should be a federation which uses a consensus mechanism to determine what course of action to take. Any dishonest behavior by the Statechain entity can be proven using the Statechain. Therefore the current owner of the UTXO should store a copy of the Statechain, to prevent the Statechain entity from replacing the public ledger and feigning ignorance.

In order to deposit funds on a Statechain, participant Bob creates a channel with Statechain entity Alice, in the same manner as an eltoo channel would be constructed. Bob creates a transitory secret key $x_{transitory}$ and Alice creates a secret key x_{entity}. The funds are then locked, requiring both $x_{transitory}$ and x_{entity} to be spent, or defaulting back to Bob after a timelock expires. The timelock ensures that in the situation where the Statechain entity disappears or refuses cooperation, the current owner can still withdraw their coins on-chain. The eltoo construction ensures that an outdated state that is broadcast to the blockchain can be replaced by a newer state that is broadcast before the funds have been spent.

In order for Bob to send the funds to Charlie, the following three steps must occur:

- Alice and Bob update the state of the channel so that upon expiration of the timelock, the funds are sent to Charlie.
- Charlie obtains knowledge of $x_{transitory}$.
- The Statechain is updated to reflect the transfer.

After the transfer, both Bob and Charlie know the value of secret key $x_{transitory}$ and both can update the state or spend from the Statechain with the cooperation of Alice, since she knows secret key x_{entity}. However, Alice promises she will only cooperate with the current owner as listed in the Statechain. In the situation that Alice behaves dishonestly by colluding with a previous owner, this can be proven using the data within the Statechain. Likewise, the Statechain entity can disprove false accusations using the data within the Statechain.

In order for this transfer of ownership to be properly verifiable, Alice requires the signatures of both Bob and Charlie. Bob's signature to prove that he requested the transfer, and Charlie's signature to prove that he acknowledged receiving the updated state. In order to ensure that neither party can falsely claim their counterparty, or the Statechain entity, is cheating, these signatures need to be provided atomically. This can be achieved through adaptor signatures, using the following steps [34]:

1) Alice, Bob, and Charlie, each generate a secret, t, u, v respectively. They then exchange the public curve points $t \cdot G = T$, $u \cdot G = U$, and $v \cdot G = V$. From these values the shared adaptor value $T + U + V = N$ is calculated.
2) Bob and Charlie construct adaptor signature $s'_{B,C}$, which is missing N. This signature is sent to Alice.
3) Alice and Bob create adaptor signature $s'_{A,X}$, which is missing N, where A is the public key for x_{entity}, and X is the public key for $x_{transitory}$. This signature is sent to Charlie.
4) Bob reveals the nonce r_x which he used in $s'_{A,X}$ to Charlie through a secured channel.
5) Charlie reveals v to Bob through a secured channel.
6) Bob reveals $(v + u)$ to Alice though a secured channel.
7) Alice uses $(v + u)$, the adaptor signature $s'_{B,C}$ and t to construct a valid signature $s_{B,C}$ indicating the transfer of ownership was acknowledged by both Bob and Charlie. She then publishes this secret onto the statechain, thereby revealing $(t+u+v)$ to Charlie. Allowing him to complete $s_{A,X}$ and claim ownership of the UTXO.
8) Alice helps Charlie learn $x_{transitory}$ by providing him with her adaptor signature for $s_{A,X}$. Charlie can now calculate $x_{transitory} = \frac{s'_{A,X} - s'_A - r_x}{e}$

After step 7 Charlie can claim the funds in the UTXO by publishing the transaction to the blockchain and waiting for the timelock to expire. However, he does not yet have $x_{transitory}$, which is required for him to transfer ownership of the UTXO to a new party, or to claim the funds directly with Alice's cooperation. At this point, Bob has no further incentive to participate in the transaction, as $s_{B,C}$ has been published on the Statechain, proving he has performed the transaction. Therefore Bob cannot be relied upon to transfer $x_{transitory}$ to Charlie after this step. Alice on the other hand, has an incentive to assist Charlie in learning $x_{transitory}$, as she benefits from the continued existence of the Statechain.

A. Security

Within a Statechain, the user's funds can only be guaranteed as long as the Statechain entity or all previous owners are honest. However, from a game theory perspective, previous users have no incentive to remain honest. Therefore previous owners are assumed to be willing to cooperate with a malicious Statechain entity. On the other hand, a Statechain entity receives fees for maintaining a Statechain, and has an incentive to ensure the continued existence of its flow of income. Therefore, the Statechain entity is assumed to be honest. Despite the assumption that previous owners are malicious, the existence of $x_{transitory}$ is necessary to improve the security while the Statechain entity is honest. As not having knowledge of $x_{transitory}$ makes the Statechain entity incapable of seizing the funds within the channel, for example if it is commanded to do so by a government. Additionally, in the current legal system it may result in more lax regulations for Statechain entities, as they do not have direct ownership of the cryptocurrency, which they would have without $x_{transitory}$.

To ensure that the Statechain entity is incentivized to behave honestly, all malicious behavior by the Statechain entity must result in a loss of reputation. Therefore all malicious behavior must be provable. Consequently, it must not be possible to falsely accuse the Statechain entity of behaving dishonestly. Malicious behavior by the Statechain entity is specified as transferring ownership of the funds without the consent of the current owner. This can be done either by spending the UTXO on-chain, or by transferring ownership of the UTXO within the Statechain. This results in the following requirements:

1) It must be provable by the current owner that the Statechain entity has stolen the funds by spending the UTXO on chain.
2) It must be provable by the current owner that the Statechain entity has stolen the funds by transferring ownership of the UTXO within the Statechain.
3) It must be impossible for an uninvolved party to prove an honest Statechain entity is behaving maliciously.
4) It must be impossible for a previous owner to prove an honest Statechain entity is behaving maliciously.
5) It must be impossible for the current owner to prove an honest Statechain entity is behaving maliciously.

It can be shown that Statechains meet all these security requirements, under the assumption that the signature scheme that is used is existentially unforgeable for chosen-message attacks. The Schnorr signatures used within the example have been proven to be existentially unforgeable for chosen-message attacks in the random oracle model, under the assumption that the discrete logarithm problem is hard [30].

1-2) Every action taken by the Statechain entity is required to be signed off on by the owner at the time of that action. These signed confirmations are recorded within the Statechain. If the Statechain entity operates in a different manner than listed in the Statechain, malicious behavior can easily be proven. However, it is possible for the Statechain entity to cooperate with a previous owner to create a valid fork of the Statechain. This enables the Statechain entity to list the forked Statechain which matches its malicious behavior. Nevertheless this malicious behavior can be proven by the legitimate current owner by providing a valid spend transaction for the Statechain channel. Such a spend transaction can only be created with the cooperation of the Statechain entity, and the forked Statechain cannot encompass this transaction, as this requires a transfer request to have been signed by the legitimate current owner. Therefore, proving the

existence of a transaction which has not been recorded within the Statechain that is listed by the Statechain entity is a clear indication of fraud.

However, it should be noted that if the current owner does not monitor the Statechain, it is possible for a malicious Statechain entity to use this fork to scam multiple parties with a single UTXO. The forked Statechain appears valid to a new participant, enabling the Statechain entity and the cooperating previous owner to steal both the funds and the service or product offered by the new participant and paid for with the UTXO. From a game theory perspective this can be considered a serious flaw within the system. The loss for the legitimate current owner remains the same independent of the number of people that are scammed by the Statechain entity using the UTXO. Therefore, there is no direct incentive for the legitimate current owner to use resources to monitor the Statechain listed by the Statechain entity. The possibility of scamming a large number of participants using a single UTXO may cause a Statechain entity to risk its reputation more willingly.

3) For an honest Statechain entity, the Statechain will always correspond with the actions it is taking. Therefore, to accuse an honest Statechain entity of malicious behavior, the accuser must be able to produce a valid spend transaction for the Statechain channel. This is not possible for an uninvolved party without obtaining both the Statechain entity's secret key and $x_{transitory}$, as Schnorr signatures are existentially unforgeable.
4) Similarly to the previous point, the accuser is required to provide a valid spend transaction for the Statechain channel. In this case the accuser has such a transaction, as they were the owner of the UTXO within the channel at some previous point in time. However, since the Statechain entity is behaving honestly, a transfer requested signed by the accuser must be part of the Statechain that the Statechain entity is listing. By proving the existence of this signed transfer request within the Statechain, the accusation can easily be debunked: the Statechain entity being capable to forging such a signature is in conflict with the assumption that the discrete logarithm problem is hard.
5) An honest Statechain entity will always list the current owner as the current owner in its Statechain, and will not sign an on-chain spend transaction without a request to do so that is signed by the current owner. Therefore, it is impossible for the current owner to accuse the Statechain entity of performing malicious behavior.

B. Use cases

By moving transactions off-chain, the transaction speed and transaction throughput of a blockchain can greatly be increased. Additionally, since not all transactions have to be recorded on the blockchain, a significant amount of storage space within the blockchain can be saved. Therefore, second layer solutions are a much needed addition to the blockchain environment, and there are multiple use cases for Statechains. However, Statechains are not the only second layer solution and may not be the best suited for each application. In this subsection, Statechains are compared with Lightning [31] to identify use cases to which Statechains are more suited than Lightning.

At the time of writing, Lightning is the most popular off-chain solution for Bitcoin. However, Lightning suffers from several limitations that do not plague Statechains. On the other hand, Lightning enables several features which cannot be achieved with Statechains. Therefore, Statechains are not intended as competition to Lightning, but rather as complementary.

Firstly, funds on the Statechain are locked between the participant and the Statechain entity. This allows the funds to be directly transferred to any willing participant on the blockchain, whereas in Lightning, both participants required to be part of the same Lightning network. Therefore, it is likely that each participant has to create multiple channels on-chain to make proper use of Lightning. On the other hand, for Statechains the number of channels is determined not by the number of users, but by the value stored within each individual channel. In the short term, it is therefore likely that Statechains require fewer on-chain channels to be created to facilitate the transfer of an adequate amount of funds to the entire network of users.

Secondly, Statechains can instantly transfer any value that can be stored within a single UTXO, whereas Lightning is limited by the channel capacity between the transacting participants. Due to the Lightning's coverage currently being limited, it is unlikely a participant will find a path to another participant that can facilitate large sums. However, if the value being transferred is small, the costs of opening or closing the channel on-chain may outweigh the value being transferred. In this situation it is not desirable to create a Statechain channel, as the value within cannot reasonably be reclaimed on-chain. Therefore, Statechains can be employed to transfer values too large for the Lightning network to process, and Lightning can be used for smaller values that are unsuitable for Statechains.

Thirdly, Statechains can be used for the off-chain transfer of security tokens on the blockchain. Within the Lightning network, the transfer of funds depends on the fungibility of the funds. This prevents the transfer of unique security tokens as these are not fungible. On the other hand, Statechains directly transfer the ownership of the UTXO, and therefore the real world asset which the funds within this UTXO represent.

Fourthly, by transferring ownership of the UTXO to a channel contract (Figure 10), it is possible to open a channel within a Statechain. This allows the creation of a Lightning channel without being required to wait for the contract to be final on the blockchain. Additionally, a single Statechain could be used to create and close multiple Lightning channels over the course of its lifetime. This reduces the amount of transactions required on the blockchain even further, as it is no longer necessary to broadcast the unilateral closing and opening of Lightning channels to the blockchain.

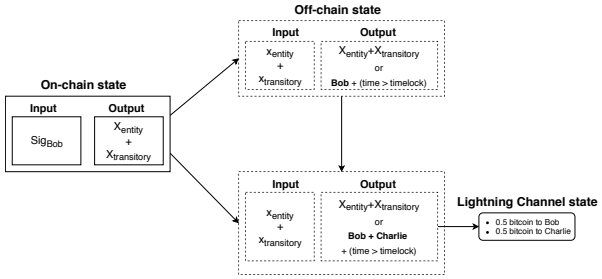

Fig. 10. Transferring ownership of the UTXO from Bob to a contract between Bob and Charlie, enabling Lightning transactions between them.

Lastly, Statechains have a few more general advantages over Lightning that do not result in a specific use case. Within Lightning, watchtowers are required to keep the channel secure if the user goes offline. Due to the existence of the Statechain entity this is not required within Statechains, as the Statechain entity has an incentive to monitor the blockchain itself as failure to do so could result in a loss of reputation. Another advantage that Statechains hold is that there is no need to rebalance channels[3]. Though channel rebalancing may be solved through wide spread adoption of the Lightning Network, this problem currently requires other methods of transferring value. This therefore often results in additional on-chain transactions that would not be required when using Statechains.

C. Criticism and Improvements

As described in the previous section, there are already several use cases for which Statechains are suitable. However, Statechains still have several flaws which may be improved, in this section several of these flaws are scrutinized and some possible areas of improvement are identified.

Statechains require a certain level of trust. The user trusts that the Statechain entity will behave honestly. This is enforced through the fees that a Statechain entity collects from its users, resulting in a loss of income should it be caught behaving maliciously. However, several factors must be taken into account when considering whether this assumption holds merit. It should be assumed that a malicious Statechain entity may collect several, if not all, of the transitory keys for the Statechains it controls. It cannot be proven that the Statechain entity has obtained these keys until it decides to use them for malicious behavior. As such, a patient Statechain entity could wait until it has complete control of all its Statechains before stealing the funds within. If the Statechain entity determines the future fees it may receive are less than the funds it may steal right now, it will naturally do so according to game theory. Therefore, the fees a Statechain entity should receive should be proportionate to the amount of funds within the channels it is facilitating. Naturally, this would result in high fees for the transfer of large funds, something that was identified as one of the main strong points of Statechains in subsection *B*. Therefore, this may harm the adoption of Statechains for this use case. Furthermore, fees based on the amount of funds within a channel may create a discrepancy in the importance the Statechain entity places on maintaining certain channels. The situation may occur where a significant portion of a Statechain entity's income results from a single channel. If at some point in time the owner of this channel desires to close it, the Statechain entity may be prompted to steal the funds it controls, as it is about to lose a significant portion of its income. This problem can be mitigated through a blind variant of the Statechain protocol. However, such a variant fails to completely solve the problems, as the Statechain entity can still learn the values it controls from previous owners who have nothing to lose by revealing this information to the Statechain entity. Blinded Statechains and their benefits and flaws are further discussed in section V.

Fees are integral to the security of Statechains. However, unlike Lightning, it is impossible to provide fees for the Statechain entity through the channel itself [34]. Within the Statechain protocol it is impossible to split UTXOs into smaller amounts. Mathematically it is possible to split the funds within the channel off-chain. However, this would create a massive overhead for the Statechain entity, requiring that it only cooperates in the signing of a new transaction if a partial owner only spends their partial amount and leaves all other values unchanged. Therefore, a separate construction is needed through which a Statechain entity receives fees.

Being unable to split the UTXO within the Statechain also severely limits the capabilities of Lightning channels on top of Statechains. Since the funds within a Lightning channel are divided between the two participants, it is necessary for the channel balance to be completely one-sided before the channel on top of the Statechain can be closed without closing the Statechain channel. In the situation where the Statechain channel is closed when the Lightning channel is closed, the on-chain gain of using Lightning on top of a Statechain is lost. Additionally, the security of this Lightning channel is equal to that of a Statechain, rather than that of Lightning, as a malicious Statechain entity and owner can steal the funds. The recipient of the Lightning channel must therefore be willing to participate in an environment that requires more trust than the blockchain or Lightning itself.

According to the Statechain protocol, Charlie is required to verify all the previous states within the Statechain before accepting the transfer. This way Charlie detects any irregularities within the Statechain before considering Bob's payment complete. However, it is possible for a malicious Statechain entity to cooperate with a malicious previous owner and create a fork of the Statechain as detailed in subsection *A*. It is

[3]While the Lightning Network is in its infancy, it is likely that a user will not be spending an equal amount of funds through a channel as they receive through that channel. For example, a user may pay their groceries through one channel, yet receive their salary through a different channel or outside of the network. This will result in a lopsided Lightning channel, which can now only be used to transfer funds back to the user.

impossible for Charlie to identify which Statechain is the legitimate one between the fork and the actual Statechain. Every scam a Statechain entity might pull has the same requirements, and carries the same risks for the entity as creating a fork. Therefore, it is sufficient for Charlie to simply verify that the most current state within the Statechain is a valid spend transaction for a corresponding on-chain channel, as verifying every state costs additional computation and does in actuality not increase the security of the protocol.

The ability to fork the Statechain is inherent to the Statechain protocol, as an owner cooperating with the Statechain should be able to create a new state to add to the end of the chain. However, there are possible improvements which can increase the chance of a forked Statechain being detected and reported, thereby further disincentivizing the Statechain entity to behave maliciously in this manner.

A forked Statechain is detected when the current legitimate owner realises that the Statechain listed by the entity has changed and that they are no longer listed as current owner. However, it is possible that instead of replacing the Statechain, the Statechain entity has listed the fork at a different location. Through this method it is possible for the Statechain entity to scam multiple parties, where none notice the malicious behavior until one of them attempts to close the on-chain channel.

Therefore, it is necessary to devise a method through which it can be assured that the Statechain entity can only list the Statechain for a specific on-chain channel in a single location. The simplest solution would be for the on-chain channel to reference the location where the corresponding Statechain is stored. If the ability to attach an arbitrary message to a transaction is not supported by the blockchain, this could also be achieved by transferring dust in the same transaction that is used to create the on-chain channel. This dust should be sent to an address that is the hash of the location the Statechain is stored. A potential owner can then simply trace the on-chain channel back to the transaction that created it and verify that the dust was sent to the correct address. However, this method has the downside that it creates additional transactions on the blockchain.

In the situation where a fork is made, this is only detected when a wronged owner realises the existence of a second Statechain for their UTXO. However an incentive is necessary for a scammed owner to report the malicious behavior. As from a game theory perspective they have nothing to gain by reporting a malicious Statechain entity and would be best off attempting to recuperate their losses by getting in on the scam. The owner cannot regain their funds since it is clear that the Statechain entity is being maliciously and therefore has no incentive to let them claim their funds on the blockchain. Reporting the Statechain entity will not result in the recuperation of funds for the owner. The only way they can recuperate their losses is by knowingly selling their compromised Statechain to an unsuspecting buyer.

Such a system can be constructed on a blockchain that supports smart contracts. Upon the creation of the Statechain, the Statechain entity is required to add the public key of the channel to the smart contract and to deposit some currency into the contract. If the contract is then provided with two states with the same state number that are signed with a key corresponding to the public key of the channel, the uploader is rewarded the deposited currency. Alternatively, if the smart contract confirms that the on-chain channel has been closed and a certain amount of time has expired, the Statechain entity can reclaim the funds.

Another flaw within the Statechain protocol is that Statechains themselves suffer from scaling problems similarly to the blockchain. Each time ownership of a Statechain is transferred, the Statechain increases in length. This results in an increase in storage space required, as well as the time required to verify the entire Statechain. However, the latter is only of consequence in the event that malicious behavior must be proven, as under normal circumstances only verifying the most recent state is sufficient. In an attempt to solve the former, section IV details State-accumulators.

A further improvement, suggested by Somsen [35], is to create a non-interactive version of Statechains. In its current form, the transferring of a UTXO over a Statechain requires all parties to be online. As on-chain transactions can be completed without the recipient being online, this can be considered a flaw within Statechains when compared to the blockchain itself. However, it cannot simply be left out, as without the recipients signature the Statechain entity could be wrongfully accused of malicious behavior. If a protocol were devised where the recipients signature can be safely excluded, a method would still need to be devised for the recipient to learn the transitory key at a later date, without the cooperation of the previous owner.

Lastly, Somsen has also suggested [35] the development is the usage of a hardware security module (HSM) to store the transitory key. Through this method it would become impossible for the Statechain entity to learn the transitory key as only the current owner has access to this key. Upon transferring ownership from Bob to Charlie, the key would be deleted from Bob's HSM. Such a construction would remove all requirement of trust in the Statechain entity. However, it would introduce the requirement to trust the HSM, which is far from ideal as Somsen admits in his presentation.

IV. State-accumulators

Each transfer of UTXO ownership within a Statechain increases the length of the chain. As the entire history of a Statechain is necessary to ensure malicious behavior can be proven, it is not possible to simply drop old states. Therefore, a possible solution is to use an alternative data structure to store the states. The following is an exploration into the possibility of State-accumulators, a conceptual replacement for the Statechain protocol, where the signatures are stored within an accumulator rather than a public ledger. Accumulators

were chosen as they are well known for their ability to remain constant in size despite containing a large assortment of values. Specifically, the explanations and examples in this section assume the use of a 2048-bit RSA accumulator [23] and Schnorr signatures. Through the advantages offered by cryptographic-accumulators, State-accumulators can have a constant size independent of the number of transfers that are made within the lifetime of the channel. The current legitimate owner only needs to store the State-accumulator, the transitory key, and their private invalidation key, all of which are constant in size. This invalidation key is necessary to ensure malicious behavior by the State-accumulator entity is provable, they are described in detail in subsection *A*. However, the State-accumulator entity is still required to store all previous states in addition to the State-accumulator, as inclusion and exclusion proofs for an accumulator cannot be securely created without this knowledge, or knowledge of the trapdoor information. Therefore this solution only improves the storage requirements for the participants of the State-accumulator, and not for the entities providing the service. Lastly, it should be noted that State-accumulators have the same requirements of a blockchain as Statechains: support for eltoo style channels and adaptor signatures.

A. Invalidation Keys

In addition to using an accumulator instead of a ledger, an obvious difference between Statechains and State-accumulators is that State-accumulators only list the current owner, rather than the entire transaction history of the channel. Due to the possibility of forking the Statechain, listing the history does not provide additional legitimacy to the current owner listed by the Statechain entity. However, the listing of this history is necessary for the legitimate current owner to prove a malicious Statechain entity has forked the Statechain. As shown in section III-A, this proof is achieved by having the legitimate current owner prove that the Statechain entity acknowledged them as the owner of the channel at some point in time, and by proving that the Statechain is missing a transfer request signed with his private key. In other words, the accuser must prove that they were the channel owner at some point in time, and that they have not relinquished this ownership.

Within a State-accumulator, proving ownership of the channel at some point in time can be achieved by the same method as used in Statechains: providing a valid on-chain spend transaction for the channel. However, proving that ownership has not been relinquished is difficult, as signed requests cannot simply be extracted from the accumulator. Therefore, it cannot simply be checked that none of the signed transfer requests within the accumulator are signed using the same private key as the proof of ownership. Thus, a participant claiming current ownership would be required to create an exclusion proof for every possible signed transfer request which would relinquish his ownership. This is computationally infeasible, as the number of possible receivers in the transfer request is equal to the public key space, minus the participant's own public key. Additionally, due to the nonce used in Schnorr signatures, there are many possible valid signatures for the same message and private keys.

A different method must therefore be devised to prove that ownership has not yet been relinquished. Simply removing values corresponding to previous owners from the accumulator is not a secure option, as the State-accumulator entity has knowledge of the legitimate current owner and can therefore remove them from the accumulator. Instead, a value is needed that can be added to the accumulator to invalidate an ownership proof belonging to a previous owner. If an exclusion proof can be created for this invalidation value, it is proof that the corresponding proof of ownership is the current one.

To ensure that a single exclusion proof is sufficient, the invalidation value must be predictable from the proof of ownership. However, if anyone can predict the invalidation value from the signed proof of ownership, a malicious State-accumulator entity can simply add the invalidation value to the accumulator without the cooperation of the current owner. Therefore, a proof of ownership must contain a computationally hiding commitment to the invalidation value, for which the committed value is revealed upon signing a request to transfer ownership of the channel. The commitment is referred to as the public invalidation key, and the value that is committed is referred to as the private invalidation key.

Lastly, in Statechains a valid on-chain spend transaction can be used as the proof of ownership. It may be difficult to insert the public invalidation key to such a transaction, as it must remain valid on-chain. Additionally, the participant receiving ownership of the channel is not directly involved in the creation of their on-chain spend transaction, further complicating the addition of the public invalidation key. Therefore, within State-accumulators the proof of ownership is shifted from a valid on-chain spend transaction, to the signed transfer request indicating the participant as the receiver. However, to enable the usage of the signed transfer requests as such, they must be signed by the State-accumulator entity in addition to the parties performing the transaction.

B. Protocol

A State-accumulator is specified where Bob is the original owner and Alice is the State-accumulator entity. As with a Statechain, the State-accumulator is created when Bob creates an eltoo style channel with Alice. A transaction which enables Bob to claim initial ownership of the UTXO is created. Additionally, Alice and Bob perform a multi-signature over a message containing both the UTXO on-chain txid and public invalidation key $u \cdot G = U$, for which Bob knows private invalidation key u. Alice adds this signature to the accumulator, and publicly lists the signature and invalidation value U, as she would for a Statechain.

In order for Bob to transfer ownership of this UTXO to Charlie, the following protocol is specified.

1) Charlie verifies $s_{A,B}$ listed by Alice, and verifies the existence of the channel on chain.
2) Alice generates adaptor secret t, and Charlie generates adaptor secret v, and private invalidation key w. They

then communicate their public curve points, $t \cdot G = T$, $v \cdot G = V$, $w \cdot G = W$, to each other and Bob. All three of them then compute $T + U + V = N$.

3) Bob and Charlie construct adaptor signature $s'_{B,C}$, which is missing N, over a message which contains the txid of the UTXO, as well as public invalidation key W. This signature is then sent to Alice.
4) Alice adds her own signature, creating adaptor signature $s'_{A,B,C}$, and sends this to Charlie.
5) Alice and the transitory key holder (in this case Bob) create adaptor signature $s'_{A,X}$, which is missing N. The full version of this signature enables spending from the on-chain channel. This signature is then sent to Charlie.
6) Bob reveals the secret nonce r_x which he used in $s'_{A,X}$ to Charlie through a secured channel.
7) Charlie verifies $s'_{A,B,C}$, $s'_{A,X}$, and r_x. Once all three values are verified, he reveals v to Bob through a secured channel.
8) Bob reveals $(u + v)$ to Alice through a secured channel.
9) Alice uses $(u + v)$, her adaptor secret t, and the adaptor signature $s'_{A,B,C}$, to construct a valid signature $s_{A,B,C}$. She then adds this signature to the accumulator and lists the signature publicly, replacing $s_{A,B}$. Additionally she replaces Bob's public invalidation key U with Charlie's public invalidation key W.
10) Charlie now reads $s_{A,B,C}$ and subtracts $s'_{A,B,C}$ from it, thereby learning $(t + u + v)$. He uses this knowledge to construct $s_{A,X}$ from $s'_{A,X}$.
11) Charlie now reveals v to Alice through a secured channel. Alice calculates u by subtracting v from $(u + v)$ which she received from Bob earlier. She then adds u to the accumulator.
12) Alice helps Charlie learn transitory key x by providing him with her adaptor signature s'_A which was used in $s_{A,X}$. Charlie calculates transitory key x from $s'_{A,X}$, s'_A, and r_x.

C. Security

The State-accumulator protocol is designed to meet the same security requirements as the Statechain protocol. Similarly to Statechains, it is assumed that previous owners are willing to cooperate with a malicious State-accumulator entity, that State-accumulator entities are disincentivized to behave maliciously, and that the discrete logarithm problem is hard. Additionally, the strong RSA assumption [9] is made to ensure the security of the protocol. This assumption is required to prove the security of RSA accumulators against computationally bound adversaries [23]. However, the security of RSA accumulators does not hold if the adversary is able to factorize the modulus used for the accumulator. Within the State-accumulator protocol, it is assumed that the State-accumulator entity has access to the factors of the modulus.

Additionally, the strong RSA assumption enables the hashing of arbitrary strings to prime numbers. This feature is necessary due to RSA accumulators only accepting prime numbers as inputs. However, the signatures which are to be added to the State-accumulator cannot be guaranteed to be primes. It has been shown [18] that there are secure methods to hash the prime numbers, under the strong RSA assumption.

The following requirements, similar to those for the Statechain protocol, are specified:

1) It must be provable by the current owner that the State-accumulator entity has stolen the funds by spending the UTXO on chain.
2) It must be provable by the current owner that the State-accumulator entity has stolen the funds by transferring ownership of the UTXO within the State-accumulator.
3) It must be impossible for an uninvolved party to prove an honest State-accumulator entity is behaving maliciously.
4) It must be impossible for a previous owner to prove an honest State-accumulator entity is behaving maliciously.
5) It must be impossible for the current owner to prove an honest State-accumulator entity is behaving maliciously.

It can be shown that the State-accumulator protocol meets these requirements under the specified security assumptions:

1-2) Every action the State-accumulator entity takes, must have a corresponding request signed by the owner at the time of that action. Similarly to Statechains, it is easy to prove malicious behavior of the State-accumulator entity if its current action does not match the currently publicly listed signed request. However, due to previous actions no longer being publicly listed, the State-accumulator entity can list a request signed with a random key it claims belongs to the current owner. This effectively results in the same situation as a forked Statechain, where the State-accumulator entity's actions appear to correspond with the publicly listed signed requests. Note that the State-accumulator entity does not need the cooperation of a previous owner to achieve the "fork" within the State-accumulator protocol. However, this does not diminish the security compared to Statechains, as the assumption is made that a previous owner is always willing to cooperate with a malicious Statechain entity.

A participant can prove current ownership of the channel by providing the signed transfer request in which the participant is the receiver, and by providing an exclusion proof of their private invalidation key for the State-accumulator. If the participant can provide this proof, and is not listed by the State-accumulator entity as the current owner, then the State-accumulator entity is behaving maliciously.

Determining the current owner's private invalidation key, from his public invalidation key, is equivalent to the elliptic curve discrete logarithm problem. Therefore it is considered impossible for the State-accumulator to correctly guess the private invalidation key and add it to the accumulator, under the assumption that the discrete logarithm problem is hard.

Note that to ensure that when creating the exclusion proof for the secret invalidation key, the accuser should do so using a zero knowledge proof, or after receiving a com-

mitment to the State-accumulator, signed by the State-accumulator entity. This is necessary to ensure that the State-accumulator entity cannot simply learn the private invalidation key once the proof has been posted, add it to the State-accumulator, and then claim that the accuser used an out-dated version of the State-accumulator for a false accusation.

3) For an honest State-accumulator entity, the listed most recent signed transfer request, will always correspond with the the most recent action taken by the State-accumulator entity. Therefore, to accuse an honest State-accumulator entity of malicious behavior, the uninvolved party must be capable of proving that it is in fact the legitimate current owner, as opposed to the current owner listed by the State-accumulator entity. To do so, the accuser must have a transfer request which has been signed by the State-accumulator entity. This is not possible for an uninvolved party without obtaining the State-accumulator entity's secret key, as Schnorr signatures are existentially unforgeable [30].
4) Similarly to the previous point, the accuser is required to prove that they are the current owner, as opposed to the owner listed by the State-accumulator entity. In this case the accuser has access to a transfer request signed by the State-accumulator entity, which indicates him as the receiver of the channel ownership. For this accusation to be valid, it must be shown that the State-accumulator does not contain the private invalidation key corresponding to the public invalidation key contained within the signed transfer request.

   The accuser must request the State-accumulator to provide a signed commitment to the current State-accumulator. An attempt can then be made to create an exclusion proof for the private invalidation key. However, succeeding in creating an exclusion proof for a value within the accumulator has been proven to be computationally infeasible if the strong RSA assumption holds [23].
5) An honest State-accumulator entity will always list the current owner as the the owner, and will not sign an on-chain spend transaction without a request to do so that is signed by the current owner. Therefore, it is impossible for the current owner to accuse the State-accumulator entity of performing malicious behavior.

Execution of the protocol may be aborted at any point in time, either due to a loss in connectivity with one or more of the participants, or due to malicious behavior by one or more parties involved. The following analyzes the consequences of aborting the protocol at each step of the State-accumulator protocols:

1-6) These steps can be considered the setup of the transfer. During any of these steps any party can decide to cancel the transfer by simply refusing to continue the next step. None of the adaptor signatures that have been exchanged can be completed without computing another parties secret. This is impossible under the assumption that the elliptic curve discrete logarithm problem is hard.
7) Upon revealing v to Bob, Charlie can no longer abort the process. Bob can continue the transaction at a later point in time.
8) Upon revealing $(u + v)$ to Alice, Bob can no longer abort the process without the cooperation of Alice. If Bob desires to abort the transfer after this step he must inform Alice of his intentions before she publishes the completed signature approving the transfer.
9) Charlie is now officially the owner of the UTXO and the process cannot be stopped. Bob no longer has an incentive to participate in the transfer process. Note that at this time, Bob's private invalidation key u is not yet part of the accumulator. Therefore Bob could use his proof of ownership to claim Alice is behaving maliciously. However, Alice still has signature $s_{A,B,C}$, which proves Bob requested the transfer.
10) Charlie can now redeem the UTXO on chain. However he cannot yet spend it off-chain as he does not yet have knowledge of the transitory key.
11) Alice now has knowledge of Bob's private invalidation key, thereby making it possible for her to safely discard $s_{B,C}$, when she no longer needs the signature to indicate Charlie as the current owner. However, Charlie still cannot spend the UTXO off-chain.
12) Charlie can now transfer ownership of the UTXO off-chain.

D. Results & Discussion

Using a State-accumulator, the current owner is required to store the State-accumulator, the transitory key, and their private invalidation key. Using 2048-bit RSA, the State-accumulator itself is 256 bytes in size. The private key within a Schnorr signature scheme is 32 bytes. Therefore, both the transitory key and the private invalidation key are 32 bytes long, resulting in a total of 320 bytes required storage space for the owner of the State-accumulator. A Schnorr signature is 64 bytes long, meaning that a State-accumulator is a more efficient method of storage as soon as the UTXO within a Statechain has been transferred 5 times.

The State-accumulator entity is required to store the State-accumulator, all of the signatures within the State-accumulator, the current public invalidation key, and the current state which is a Schnorr signature. This brings the total storage space required by the State-accumulator entity to $256 + (n \cdot 64) + 32 + 32$, where n is the number of time the UTXO has been transferred. Effectively this means that a State-accumulator entity must store 320 bytes more per channel than a Statechain entity.

State-accumulators require a one-time constant size increase in storage space by the facilitating entity, and greatly improve the amount of storage space required by the users. As the users are already required to provide a fee to the entity to ensure security this can be considered a valid trade-off. If adoption of Statechains spreads widely and they become used similarly to

how we use paper money, they may transfer ownership very often and it will quickly become cumbersome for an owner to keep the history of each of these bills stored.

Lastly, a downside of State-accumulators, is that through the use of RSA they suffer the requirement of a trusted setup. This forms a major problem as neither the State-accumulator entity, nor anyone else, should be allowed knowledge the RSA trapdoor information, as this knowledge would enable the party to create arbitrary proofs for the State-accumulator. Such an arbitrary proof could then be used to perform malicious behavior which cannot be proven. However, this can be mitigated through using RSA moduli for which noone knows the trapdoor values. Such RSA moduli can be found elsewhere, such as the RSA puzzles created by Ron Rivest [11].

V. Blind Statechains

Blind Statechains are Statechains where the signatures have been blinded from the Statechain entity, hiding the messages it is signing from the entity. Through blinded Statechains the privacy of Statechains can be greatly increased. Within an ordinary, not-blinded Statechain, the on-chain channel can easily be linked to the publicly listed Statechain, as well as every address that has had off-chain ownership of the UTXO. On the other hand, a blinded Statechain entity will only list the blinded signatures and is not even aware itself which channels it controls or for whom it is signing transactions. Neither the Statechain entity nor an observer can learn which on-chain addresses had off-chain control of the UTXO without unblinding the states. Additionally, since the Statechain entity is no longer aware of what it is signing, it is not aware whether it is signing a transaction or the closing of the channel, or the amount within the channel.

Intuitively it may seem that this property also increases the security of a Statechain. After all, it should be more difficult for a Statechain entity to accurately estimate its loss in income compared to its gain from stealing channel funds, when it has no knowledge of values contained within these channels. However, this property provides no extra security if a method exists for the Statechain entity to contact the previous owners of the channels it maintains. From a game theory perspective, a previous owner has nothing to lose in a situation where the Statechain entity behaves maliciously, and thus will be willing to unblind the Statechain for the Statechain entity if offered even a marginal incentive. The Statechain entity itself does not risk anything by reaching out to these previous owners, even if a previous owner decides not to cooperate there is no incentive for this previous owner to report the Statechain entity and they do not have any indisputable evidence. Admittedly, within a real world application the risk for this behavior by a Statechain entity will be much greater, as an accusation even without indisputable evidence could lead to a big loss in reputation.

In his blog post [33] and his messages to the bitcoin-dev mailing list [36] Somsen details a blind variant of Statechains where the Statechain entity functions as a signing server. In contrast to the original non-blind Statechain protocol this construction mandates that a new potential owner validates every previous state, otherwise security cannot be guaranteed. This raises the question whether it is feasible to create a blind variant of the Statechain protocol where validating only the most recent state provides the same amount of security as validating every previous state. The following subsection details Somsen's blinded Statechain where the Statechain entity functions as a signing server, subsection *B* explores the possibility of a blinded Statechain where only the most recent state needs to be verified.

A. Signing servers

A straightforward implementation of Blind Statechains can be achieved by using the Statechain entity as a blind signing server [36]. Within this setup the entity has minimal interaction with the users. It only provides two functions which users can call, as well as a public listing of each Statechain.

The first function which is provided is the request to form a new Statechain. The user submits their public key to the Statechain entity. The entity then generates a new private key and corresponding public key for itself for this Statechain. The entity then lists the submitted public key along with the its newly generated public key as a new Statechain. The generation of a new public key for the Statechain entity is necessary to ensure security, as using the same private key for the entity for each Statechain would make it possible for malicious users to attempt to hijack another users UTXO.

The second function enables users to transfer ownership of the Statechain. The function requires the caller to submit a blinded transaction and the new owners public key, signed with the callers public key. The entity first verifies that the signature is valid and corresponds with the public key of the current owner of the Statechain. If this is the case the entity then signs the blinded transaction with its public key belonging to this Statechain. The signed blinded transaction and the new owners public key are then appended to the publicly listed Statechain.

This setup offers a versatile functionality, as the entity can be used to sign a plethora of messages, not only Bitcoin transactions. However, due to the Statechain entity being unable to see the contents of the transactions, it is necessary for the receiving user to unblind and verify every previous transaction on the Statechain to ensure that he is not being cheated by a previous owner, whereas in non-blind Statechains it can be considered sufficient to validate only the most recent transaction. If setup properly, this unblinding can be performed by anyone who holds the transitory key [33]. This removes the necessity for each subsequent owner to store a chain of unblinding factors to enable verification of each state.

B. Blinded Singular State Verification

A major difference between ordinary, unblinded Statechains, and Blind Statechains as signing servers is the amount of steps required to verify the integrity of the Statechain. As described in section III, for an ordinary Statechain only verifying the most current transaction provides the same amount of security as verifying the entire chain. However, this is insufficient

when Blind Statechains are used as a signing server. This is due to the new possibility of a previous owner having behaved maliciously without the cooperation of the Statechain entity. Since the Statechain entity can no longer verify what it is signing, it cannot be held responsible for malicious behavior in previous states created by previous owners. This raises the question whether it is possible to design a blind Statechain where it is sufficient to only validate the most recent transaction in a timely manner, thereby possibly saving on computation time in a situation where a blind Statechain grows very large.

Firstly, it must be identified exactly which parts of the blinded transaction could be abused by a previous owner to perform a scam, assuming that a potential new owner only verifies the most recent state with the on-chain channel. In essence, a state is an eltoo style spend transaction for the on-chain state. Therefore, a state contains the following variables:

- *The address to transfer to.* The addresses used in previous states have no influence on the current or future states, therefore this variable can be kept hidden.
- *The amount of currency to be transferred.* An invalid transaction could be created by altering the amount of currency so it no longer matches the funds contained within the on-chain channel. However, this can easily be verified by comparing the state with the on-chain channel. Therefore, this value can be kept hidden from the Statechain entity.
- *The state number of this transaction.* A previous owner could have created a transaction,"$tx_{malicious}$" with itself, wherein the state number is set to the maximum. This malicious owner then creates another transaction with itself, this time with an ordinary state number. If a new potential owner only unblinds and verifies the most recent state, they are unaware that $tx_{malicious}$ exists and can be used to overwrite their on-chain transaction, due to its higher state number. The current owner could still reclaim the transaction by creating a new close transaction together with the Statechain entity. However, this reintroduces the issue of being required to be online for the funds to be secure, as the Statechain entity cannot create a new transaction without the cooperation of the owner.

To maintain accountability it is therefore necessary that the Statechain entity can verify the state number used within the transaction. Additionally, it is necessary for the entity to check that the transaction is indeed an eltoo style transaction, as a different type of transaction cannot be overwritten by newer states. However, for the sake of on-chain privacy, it is now necessary to reveal to the Statechain entity that the current owner is attempting to close the channel. Since a close transaction is a direct spend transaction, thereby hiding the fact that an off-chain channel was used.

Blind signatures are designed to both hide the message being signed and to provide unlinkability, that is to say that the signer cannot recognize who it signed the message for even if it ever verifies the unblinded message. However, the latter property is not necessary in this use case, as the Statechain entity is never required to verify the unblinded message. Therefore the same level of privacy can be achieved, by only hiding the message it is signing from the Statechain entity. A construction where only the message is hidden can be created using normal Schnorr signatures rather than blinded Schnorr. Within Schnorr signatures the challenge e is what is signed by the signer. As shown in section II-C.1, e is a hash whose preimage contains the message. Therefore, as long as the hash function is considered secure, the message can be hidden from the Statechain entity by computing e and sending it to the entity for signing. Note that while the possible values for the preimage are limited, as the transaction must still be valid on the blockchain, the possible input set is still sufficiently large as it encompasses all possible blockchain addresses.

However, the Statechain entity must still be able to verify that the correct state number was used within the message it is requested to sign, and that this message is an eltoo style transaction. Since the Statechain entity now only receives a hash, it cannot simply verify that the correct state number was used without knowing the rest of the contents of the preimage. However, knowing the message in its entirety would negate the desired blinding property. Therefore, a protocol where the owner provides the Statechain entity with a zero knowledge proof (ZKP) is necessary. Within this ZKP the owner must prove the following three statements:

1) $e = H(i)$
2) Message m in i is an eltoo style script
3) m contains the correct sequence number

Note that i is constructed in the same manner as an ordinary Schnorr 2-of-2 multisignature, i.e. the preimage of e shown in II-C.2 without the public adaptor value. Therefore, in addition to m, i must contain the public randomness, and a commitment to the public keys. If these values do not correspond to the keys and nonces used to sign e, the resulting signature will not be a valid transaction.

The following specifies the Blinded Singular State Verification protocol (BSSV) for Statechains, where Bob wants to transfer ownership of the Statechain to Charlie. Alice is the Statechain entity and currently has Bob's public key P_B listed as the current owner. Additionally, Alice has constructed a boolean circuit Φ, which takes the i, and a state number as input. Φ verifies that the second and third statements listed above are satisfied and then outputs the hash of i.

1) Charlie generates nonce r_x and sends his public randomness $R_x = r_x \cdot G$ and public key P_C to Bob.
2) Bob takes P_C and signs it with his private key corresponding to P_B. He then sends the signed P_C to Alice, requesting to transfer ownership of the Statechain to P_C.
3) Alice verifies that the message is signed by P_B, the current owner of the Statechain, and generates nonce r_A. She calculates $R_A = r_A \cdot G$ and sends R_A, the desired state number n, and boolean circuit Φ to Bob.
4) Bob uses n in his construction of the eltoo style transaction m. He then calculates challenge $e = H(i)$, where i

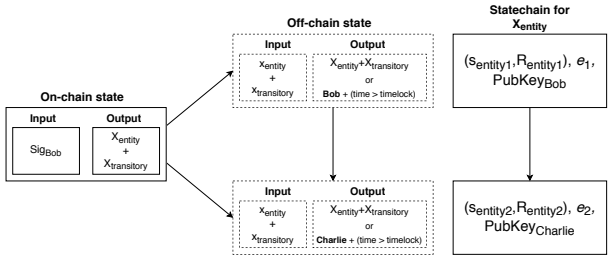

Fig. 11. Transferring ownership of the UTXO from Bob to Charlie, and updating the BSSV Statechain. The signatures listed within the Statechain are over the corresponding e. Note that PubKey_{Bob} and $\text{PubKey}_{Charlie}$ do not correspond with **Bob** and **Charlie** in the off-chain states, therefore neither the Statechain entity nor an observer of the Statechain can link these public keys to Bob or Charlie's on-chain addresses.

in constructed in the same manner as an ordinary Schnorr 2-of-2 multisignature, using both R_A and Charlie's R_x as public randomness values.

5) Bob constructs a ZKP Π that shows that Φ with inputs i and n, outputs e. Bob sends both e and π to Alice, as well as a signature over e signed with his private key corresponding to P_B. Note that this signature is not a partially signed spend transaction as it is signed with Bob's private key and not $x_{transitory}$.
6) Alice verifies Π and Bob's signature over e and creates her half of the signature $s_A = r_A + e \cdot x_{entity}$. She then adds s_A and both the signed e and P_C to the Statechain.
7) Bob takes s_A and constructs incomplete signature $s'_{A,X}$, which is missing r_x. He then sends $s'_{A,X}$ to Charlie.
8) Charlie completes $s'_{A,X}$ using r_x to create a valid signature $s_{A,X}$. He then takes s_A as listed by Alice, and uses it to calculate $x_{transitory} = \frac{s_{A,X} - s_A - r_x}{e}$.

Note that BSSV is based on the protocol where the Statechain entity functions as a signing server. Unlike non-blind Statechains, both blind variants rely on the current owner (Bob) transferring additional information to the new owner (Charlie). For the blind signing server protocol, this additional information is $x_{transitory}$, for BSSV this is the incomplete signature $s'_{A,X}$. In both cases Charlie cannot transfer ownership of the Statechain to a new owner or claim the UTXO on-chain without receiving this information from Bob, but is already considered to be the new owner by the Statechain entity. In the blind signing server protocol, this is due to the transaction listed by the Statechain entity being blinded, requiring knowledge of $x_{transitory}$ to unblind into a spendable transaction. On the other hand, BSSV does not list a blinded transaction, but rather a partially signed transaction which can only be verified with knowledge of e. This difference in design originates from the relative inefficiency of ZKPs. Using blinded Schnorr within BSSV would require Φ to prove an additional statement: e is the non-blinded version of e' which the Statechain entity is expected to sign.

It can be shown that BSSV meets all the security requirements set for original Statechain protocol:

1) It must be provable by the current owner that the Statechain entity has stolen the funds by spending the UTXO on chain.
2) It must be provable by the current owner that the Statechain entity has stolen the funds by transferring ownership of the UTXO within the Statechain.
3) It must be impossible for an uninvolved party to prove an honest Statechain entity is behaving maliciously.
4) It must be impossible for a previous owner to prove an honest Statechain entity is behaving maliciously.
5) It must be impossible for the current owner to prove an honest Statechain entity is behaving maliciously.

It can be shown that BSSV meets all of these requirements in identical fashion as shown in section III-A. This is enabled by steps 5 and 6 where the current owner signs e and sends this signature to the Statechain entity. This signed challenge functions as proof that the owner requested the Statechain entity to sign this challenge.

Lastly, it can be shown that the additional time required to compute a ZKP in BSSV is feasible for the transfer of funds. ZKBoo [19] is used as an example, as it can be used to create a ZKP for any boolean circuit and can provide a ZKP for the preimage of a SHA-256 hash within milliseconds. The runtime of ZKBoo is dictated by the number of *AND* gates within the boolean circuit.

Therefore, the amount of *AND* gates within the circuit required for BSSV must be determined. This circuit is required to prove the three statements listed earlier:

1) $e = H(i)$
2) Message m in i is an eltoo style script
3) m contains the correct sequence number

ZKBoo already provides an optimized implementation of a SHA-256 circuit [19]. This circuit contains 25344 *AND* gates. Therefore, it is only necessary to determine the number of *AND* gates required to prove statements 2 and 3. Both of these statements can be considered to be simple comparison operators. Their validity in a boolean circuit can therefore easily be represented by comparing each input bit with each bit of the desired value. Every such comparison is performed by an *XOR* gate combined with an *INV* gate. The output of every such comparison is then combined through *AND* gates.

Therefore, it is necessary to determine the exact bit-length of an eltoo style script as shown in figure 6. Since the state number is already included within the script, this does not provide any additional length. The following bit-lengths were determined from the Bitcoin Core source code [2]. Bitcoin opcodes are 1 byte in length. The input for *OP_CSV* is 5 bytes in length, and the input for *OP_CHECKMULTISIGVERIFY* and *OP_CHECKLOCKTIMEVERIFY* are 4 bytes in length. Therefore, an entire eltoo style script consists of 256 bits. This results in a total of 255 *AND* gates. This number is negligible compared to the number of *AND* gates already contained within the circuit for SHA-256. Additionally, the preimage also contains the the public keys, and public randomness

values, further increasing the size of the input of the hash function. This results in the SHA-256 circuit needing to be executed multiple times, as multiple rounds are required to compute the hash, thereby further reducing the effect of statements 2 and 3 on the total runtime required.

Consequently, the runtime can be considered to be determined by the time required to prove statement 1. As shown in the ZKBoo paper [19] this can be achieved within milliseconds. Thus, the ZKP used within BSSV will only result in a marginal increase in transaction time.

C. Results & Discussion

Blinded Statechains, where the Statechain entity operates as a signing server, and BSSV use different methods to enforce privacy. The former relies on blinded Schnorr signatures, whereas the latter depends on the security of a hash function. This difference leads to an interesting trade-off in what information can be obtained by different adversaries.

The blind signing server protocol is secure against passive onlookers. Even when closing the Statechain by spending the channel on-chain, an onlooker will not be able to differentiate this close transaction from an ordinary on-chain transaction. Therefore, it is impossible for such an onlooker to identify addresses that had ownership of this UTXO while it was transferred using the Statechain. It cannot even be determined that a Statechain was used at all. However, this protocol is susceptible to active adversaries who actively attempt to obtain this private information. It is necessary for a new owner to be able to unblind each previous State within the Statechain. As $x_{transitory}$ is used for this unblinding, any party with knowledge of $x_{transitory}$ can view each unblinded State and identify every on-chain address that had ownership at some point in time. An adversary with sufficient funds could actively pursue ownership of a large number of Statechains and learn their respective transitory keys. Other than the possible fees paid to the Statechain entity of each Statechain, this adversary can simply reclaim the funds it spent to obtain the Statechains by either closing the channel or transferring ownership of the Statechain in exchange for funds or a service. Therefore, the privacy provided by the blind signing server protocol cannot be considered adequate against an active adversary.

On the other hand, BSSV ensures that the Statechain entity and onlookers cannot identify who is participating in the Statechain without being provided the preimage for the signed transaction. Therefore, an active adversary who obtains ownership of the Statechain at a specific point in time will only be able to identify the participant whom they obtained ownership from, and the participant to whom they transfer ownership of the Statechain. However, BSSV cannot hide that a Statechain was used. As the final hash that is signed must be submitted to the blockchain to close the channel, the Statechain entity as well as passive onlookers can recognize this hash to be identical to the one listed by the Statechain. Therefore, the initial owner of the Statechain and the final owner of the Statechain can be identified, as well as the amount of transactions that were performed with this UTXO off-chain, but the on-chain addresses of the intermediary owners remain hidden.

Lastly, a downside of BSSV compared to the blind signing server protocol is the time required to perform a single transaction. While it has been shown that a BSSV transaction can be performed within an acceptable amount of time, the ZKP still takes a significantly larger amount of time than blinding and unblinding within blind Schnorr signatures. However, with each additional transaction, the blind signing server protocol requires another unblinding step. Therefore, the time required to perform a transaction will increase with the size of the Statechain. On the other hand, BSSV has a constant time requirement to perform a transaction.

VI. Conclusion & Future Work

Statechains are a second-layer solution for blockchains. Their security trade-off results in a higher amount of trust required than an on-chain transaction, or other second-layer solutions such as Lightning. Despite their flaws, the need to move transactions off the blockchain, and the benefits Statechains offer over Lightning creates valid use cases for Statechains, and therefore their improvement merits exploration. This paper has provided two such improvements: (1) State-accumulators, an alternative implementation of Statechains with improved scalability, and (2) BSSV, a protocol that enhances the privacy of Statechains.

Statechains would greatly benefit from a scheme that incentivizes a scammed owner to report a malicious Statechain entity. Such a construction is concisely described for blockchains that employ smart contracts. However, no such construction currently exists for blockchains that are more limited in their scripting capabilities, such as Bitcoin. Future research could greatly increase the security of Statechains from a game theory perspective by determining the possibility of such a construction.

Statechains themselves suffer from a scalability problem as they grow in size with each transfer until the channel is closed. In an attempt to solve this issue, the possibility of using accumulators rather than lists to store the states was explored. This solution solved the scalability issue for users but not for the State-accumulator entity, as it was still required to store all the previous states. This is considered an acceptable trade-off, as the State-accumulator entity collects fees from the users. Nonetheless, future research is recommended to explore the possibility of aggregating the data or the proofs stored by the State-accumulator entity, thereby reducing the storage space required by the State-accumulator entity.

Blinded Statchains can greatly improve the privacy of Statechains. A protocol where the Statechain entity functions as a signing server has been previously suggested. As opposed to nonblinded Statechains, this protocol requires every previous state to be verified before it can be considered secure. This raised the question whether it was possible to construct a

blinded Statechain variant where it is sufficient to only verify the most recent state. This paper presented BSSV, a protocol for blinded Statechains where a ZKP is used to provide the Statechain entity with the information necessary to ensure security despite only the most recent state being verified. However, BSSV and the the blind signing server protocol provide different levels of privacy against different types of adversaries. Future work is recommended to determine whether an efficient blinded Statechain variant can be constructed, where only verifying the most recent state provides sufficient security, while still providing the same level of privacy as the blinded signing server protocol.

Acknowledgements

In addition to the members of the examination committee, we extend our thanks to Tim Menapace for his guidance and supervision during this research.

References

[1] Basis of Lightning Technology #4: Onion Routing Protocol. https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md. Accessed: 2018-11-03.

[2] Bitcoin Core source code. https://github.com/bitcoin/bitcoin. Accessed: 2020-01-13.

[3] Bitcoin Developer Guide. https://bitcoin.org/en/developer-guide#block-chain-overview. Accessed: 2019-01-15.

[4] Ethereum Design Rationale - Accounts and not UTXOs. https://github.com/ethereum/wiki/wiki/Design-Rationale#accounts-and-not-utxos. Accessed: 2019-01-15.

[5] Grin, a MimbleWimble implementation. https://github.com/mimblewimble/grin. Accessed: 2018-11-13.

[6] Lightning Releases; lnd v0.5.1-beta. https://github.com/lightningnetwork/lnd/releases/tag/v0.5.1-beta. Accessed: 2018-12-06.

[7] A Next-Generation Smart Contract and Decentralized Application Platform. https://github.com/ethereum/wiki/wiki/White-Paper. Accessed: 2018-11-19.

[8] Georgia Avarikioti, Felix Laufenberg, Jakub Sliwinski, Yuyi Wang, Roger Wattenhofer, and Zeta Avarikioti. Incentivizing Payment Channel Watchtowers. Scaling Bitcoin Tokyo 2018, https://www.youtube.com/watch?time_continue=3880&v=nwSuctrzV7Y. Accessed: 2018-10-09.

[9] Niko Baric and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT*, 1997.

[10] Josh Benaloh and Michael de Mare. One-Way Accumulators: A Decentralized Alternative to Digital Signatures. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, pages 274–285, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[11] Benedikt Bünz, Benjamin Fisch, and Dan Boneh. A Scalable Drop in Replacement for Merkle Trees. Scaling Bitcoin Tokyo 2018, https://www.youtube.com/watch?time_continue=3520&v=IMzLa9B1_3E, October 2018. Accessed: 2018-10-09.

[12] David Chaum. Blind Signatures for Untraceable Payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology*, pages 199–203, Boston, MA, 1983. Springer US.

[13] David Chaum, Amos Fiat, and Moni Naor. Untraceable Electronic Cash. In *Conference on the Theory and Application of Cryptography*, pages 319–327. Springer, 1988.

[14] Wei Dai. b-money, an anonymous, distributed electronic cash system. http://www.weidai.com/bmoney.txt, 1998.

[15] Christian Decker. SIGHASH_NOINPUT. BIP118 https://github.com/bitcoin/bips/blob/master/bip-0118.mediawiki, February 2017.

[16] Christian Decker, Rusty Russell, and Olaoluwa Osuntokun. eltoo: A Simple Layer2 Protocol for Bitcoin. https://blockstream.com/eltoo.pdf. Accessed: 2018-11-15.

[17] A. Frederick Dudley. What are the scaling benefits of PoS vs PoW? Ethereum Stackexchange, https://ethereum.stackexchange.com/questions/1346/what-are-the-scalability-benefits-of-pos-vs-pow. Accessed: 2019-01-16.

[18] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In *IACR Cryptology ePrint Archive*, 1999.

[19] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *25th {usenix} security symposium ({usenix} security 16)*, pages 1069–1083, 2016.

[20] Robert Greenfield. Vulnerability: Proof of Work vs. proof of Stake. https://medium.com/@robertgreenfieldiv/vulnerability-proof-of-work-vs-proof-of-stake-f0c44807d18c. Accessed: 2019-01-16.

[21] Markus Jakobsson and Ari Juels. *Proofs of Work and Bread Pudding Protocols(Extended Abstract)*, pages 258–272. Springer US, Boston, MA, 1999.

[22] Tom Elvis Jedusor. Mimblewimble. #bitcoin-wizards IRC channel, https://scalingbitcoin.org/papers/mimblewimble.txt, July 2016.

[23] Jiangtao Li, Ninghui Li, and Rui Xue. Universal accumulators with efficient nonmembership proofs. In *Applied Cryptography and Network Security*, pages 253–269. Springer, 2007.

[24] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Privacy-preserving Multi-hop Locks for Blockchain Scalability and Interoperability. Cryptology ePrint Archive, Report 2018/472, 2018. https://eprint.iacr.org/2018/472.

[25] Pedro Moreno-Sanchez and Aniket Kate. Scriptless Scripts with ECDSA. https://lists.linuxfoundation.org/pipermail/lightning-dev/attachments/20180426/fe978423/attachment-0001.pdf, April 2018.

[26] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, October 2008.

[27] Shibetoshi Nakamoto and Jackson Palmer. Dogecoin. https://dogecoin.com/. Accessed: 2018-11-23.

[28] Andrew Poelstra. Scriptless Scripts. MIT Bitcoin Expo 2017 Day 1, https://www.youtube.com/watch?v=0mVOq1jaR1U&t=39m8s, March 2017. Accessed: 2018-10-03.

[29] Andrew Poelstra. Mimblewimble and Scriptless Scripts. L2 Summit, https://www.youtube.com/watch?v=jzoS0tPUAiQ&t=3h36m, March 2018.

[30] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 387–398. Springer, 1996.

[31] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. https://lightning.network/lightning-network-paper.pdf, January 2016. DRAFT Version 0.5.9.2.

[32] M.G. Reed, P.F. Syverson, and D.M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected areas in Communications*, 16(4):482–494, 1998.

[33] Ruben Somsen. Statechains: Non-custodial Off-chain Bitcoin Transfer. https://medium.com/@RubenSomsen/statechains-non-custodial-off-chain-bitcoin-transfer-1ae4845a4a39. Accessed: 2019-08-04.

[34] Ruben Somsen. Statechains: Off-chain Transfer of UTXO Ownership. https://seoulbitcoin.kr/img/statechains.pdf, October 2018.

[35] Ruben Somsen. Statechains: Off-chain Transfer of UTXO Ownership. Scaling Bitcoin Tokyo 2018, https://www.youtube.com/watch?time_continue=2857&v=FI9cwksTrQs, October 2018. Accessed: 2018-10-09.

[36] Ruben Somsen. Formalizing Blind Statechains as a minimalistic blind signing server. bitcoin-dev mailing list, https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2019-June/017005.html, June 2019. Accessed: 2019-07-28.

[37] Statista. E-commerce share of total global retail sales from 2015 to 2021. https://www.statista.com/statistics/534123/e-commerce-share-of-retail-sales-worldwide/, 2018. Accessed: 2018-10-25.

[38] Pieter Wuille. Schnorr signature in Bitcoin. Scaling Bitcoin Milan 2016, https://www.youtube.com/watch?v=_Z0ID-0DOnc&feature=youtu.be&t=2297, October 2016. Accessed: 2018-11-23.