Garrison: A Novel Watchtower Scheme for Bitcoin

Arash Mirzaei, Amin Sakzad, Jiangshan Yu, and Ron Steinfeld

Faculty of Information Technology, Monash University, Melbourne, Australia
{arash.mirzaei,amin.sakzad,jiangshan.yu,ron.steinfeld}@monash.edu

Abstract. In this paper, we propose Garrison, which is a payment channel with watchtower for Bitcoin. For this scheme, the storage requirements of both channel parties and their watchtower would be $\mathcal{O}(log(N))$ with N being the maximum number of channel updates. Furthermore, using properties of the adaptor signature, Garrison avoids state duplication, meaning that both parties store the same version of transactions for each state and hence the number of transactions does not exponentially increase with the number of applications on top of the channel.

1 Introduction

Scalability of blockchains is a grand open challenge limiting the adoption of blockchain technologies. Payment channel is one of the promising techniques to mitigate this issue. To establish a payment channel, two parties lock their funds in a 2-of-2 multisignature address on the blockchain. Then, parties privately carry out payments among themselves and update the channel's state by exchanging off-chain transactions. Finally, each party can close the channel by publishing the last state of the channel on-chain.

However, since the channel parties are generally untrusted and blockchain miners are unaware of the off-chain transactions, a mechanism must be adopted to prevent cheating parties from publishing an old state. In Lightning network, Raiden network and many other proposals, when a channel party publishes a channel state on the blockchain, a period called *revocation period* starts, in which their counter-party can provide some evidence proving that the published state is invalid and hence prevent the channel from getting finalized with an old state.

However, the revocation process works based on the assumption that the parties are always online and synced with the blockchain to detect malicious behaviour. This requirement has always been a drawback for payment channels because it can be practically violated due to crash failures or DoS attacks on the channel party. *Watchtower* was introduced to relax this strong assumption by allowing users to delegate their tasks to the watchtower [10]. To do so, for each revoked state, some evidence is given to the watchtower. Then, the watchtower as an always-online service provider, monitors the blockchain and acts on behalf of its customers to secure their funds.

Several watchtower schemes have been proposed, some for Bitcoin and some for Turing complete blockchains, each scheme focusing on some particular properties. Monitor [4] is the first watchtower proposal and mainly focuses on channel privacy against watchtower. However, firstly, payment to the watchtower is done upon fraudulent channel closure and secondly dishonest watchtowers are not penalized if do not act upon fraud. These two issues are related to the fairness with respect to the watchtower and its customer, respectively. DCWC and DCWC* [2] provide a solution to incentivize different watchtowers to cooperate with each other. However, for these proposals, payment to the watchtower is still conditional and the watchtower is also unaccountable.

For Cerberus [3], PISA [8] and FPPW [9], watchtower must lock some funds as collateral. This collateral is given to the hiring party given that the watchtower is non-responsive. Furthermore, the watchtower is unconditionally paid per channel update. Thus, these schemes are fair w.r.t both the hiring party and the watchtower. However, this comes at the cost of decrease in currency liquidity or limitations in total capacity of channels that can be covered by a particular watchtower. This is parameterized using a factor called *coverage* [9]. Towards a different direction, [11] proposes a reputation system based on proof of work which forces watchtowers to be responsive without requiring them to lock any funds.

Outpost [6] is a novel scheme which reduces the watchtower's storage requirements per channel from $\mathcal{O}(N)$ to $\mathcal{O}(log(N))$ with N being the maximum number of channel updates. This consequently reduces the operational cost of maintaining watchtowers. Although elegantly designed, Outpost suffers from following shortcomings: 1) The storage requirements of each channel party is still $\mathcal{O}(N)$, 2) The payment channel does not avoid state duplication and hence the number of transactions exponentially increases with the number of applications on top of the channel [1], 3) Outpost works based on "punish-per-output" pattern, meaning that if there are multiple outputs in the published revoked state, the cheated party must claim each output separately. This increases the number of on-chain transactions.

Therefore, the main motivation of this paper is designing a payment channel with watchtower scheme which is storage efficient for both channel parties and the watchtower and avoids state duplication as well as output-based revocation.

2 Preliminaries and Notations

In this section the underlying cryptographic primitives of Garrison as well as notations are introduced. We closely follow the notations stated in [9].

2.1 Preliminaries

Digital Signature A digital signature scheme Π includes three algorithms as following:

– **Key Generation.** $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$ on input 1^κ (κ is the security parameter), outputs the public/private key pair (pk, sk).
– **Signing.** $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$ on inputs the private key sk and a message $m \in \{0,1\}^*$ outputs the signature σ.
– **Verification.** $b \leftarrow \mathsf{Vrfy}_{pk}(m; \sigma)$ takes the public key pk, a message m and a signature σ as input and outputs a bit b.

In this work, we assume that the utilized signature schemes are existentially unforgeable under an adaptive chosen-message attack. It guarantees that it is of negligible probability that an adversary, who has access to a signing oracle, outputs a valid signature on any new message. In this paper, we call such signature schemes secure. ECDSA [5] is a secure signature scheme that is currently being used in Bitcoin. Schnorr [12] is another important secure signature scheme that has been proposed to be introduced in Bitcoin due to its key aggregation and signature aggregation properties.

Hard relation A relation $\mathcal{R}$ with statement/witness pairs $(Y; y)$ is called a hard relation if (i) There exists a polynomial time generating algorithm $(Y; y) \leftarrow \mathsf{GenR}(1^\kappa)$ that on input 1^κ outputs a statement/witness pair $(Y; y) \in \mathcal{R}$; (ii) The relation between Y and y can be verified in polynomial time, and (iii) For any polynomial-time adversary $\mathscr{A}$, the probability that $\mathscr{A}$ on input Y outputs y is negligible. We also let $L_\mathcal{R} := \{Y \mid \exists Y \ s.t. \ (Y, y) \in \mathcal{R}\}$. Statement/witness pairs of $\mathcal{R}$ can be public/private key of a signature scheme generated by Gen algorithm.

Adaptor Signature Adaptor signatures appeared first in [1]. Adaptor signature is used in generalized channels to tie together the authorization of a commit transaction and the leakage of a secret value. In what follows, we recall how an adaptor signature works. Given a hard relation $\mathcal{R}$ and a signature scheme Π, an adaptor signature protocol Ξ includes four algorithms as follows:

– **Pre-Signing.** $\tilde{\sigma} \leftarrow \mathsf{pSign}_{sk}(m, Y)$ is a probabilistic polynomial time (PPT) algorithm that on input a private key sk, message $m \in \{0,1\}^*$ and statement $Y \in L_\mathcal{R}$, outputs a pre-signature $\tilde{\sigma}$.
– **Pre-Verification.** $b \leftarrow \mathsf{pVrfy}_{pk}(m, Y; \tilde{\sigma})$ is a deterministic polynomial time (DPT) algorithm that on input a public key pk, message $m \in \{0,1\}^*$, statement $Y \in L_\mathcal{R}$ and pre-signature $\tilde{\sigma}$, outputs a bit b.
– **Adaptation.** $\sigma \leftarrow \mathsf{Adapt}(\tilde{\sigma}, y)$ is a DPT algorithm that on input a pre-signature $\tilde{\sigma}$ and witness y, outputs a signature σ.
– **Extraction,** $\mathsf{Ext}(\sigma, \tilde{\sigma}, Y)$ is a DPT algorithm that on input a signature σ, pre-signature $\tilde{\sigma}$, and statement $Y \in L_\mathcal{R}$, outputs $\perp$ or a witness y such that $(Y, y) \in \mathcal{R}$.

Correctness of an adaptor signature guarantees that for an honestly generated pre-signature $\tilde{\sigma}$ on the message m w.r.t. a statement $Y \in L_\mathcal{R}$, we have

$\mathsf{pVrfy}_{pk}(m, Y; \tilde{\sigma}) = 1$. Furthermore, when $\tilde{\sigma}$ is adapted to the signature σ, we have $\mathsf{Vrfy}_{pk}(m; \sigma) = 1$ and $\mathsf{Ext}(\sigma, \tilde{\sigma}, Y)$ outputs y such that $(Y, y) \in \mathcal{R}$.

An adaptor signature scheme is secure if it is existentially unforgeable under chosen message attack (aEUF–CMA security), pre-signature adaptable and witness extractable. The aEUF–CMA security guarantees that it is of negligible probability that any PPT adversary who has access to signing and pre-signing oracles outputs a valid signature for any arbitrary new message m even given a valid pre-signature and its corresponding Y on m. Pre-signature adaptablity guarantees that every pre-signature (possibly generated maliciously) w.r.t. Y can adapt to a valid signature using the witness y with $(Y, y) \in \mathcal{R}$. Witness extractablity guarantees that it is of negligible probability that any PPT adversary who has access to signing and pre-signing oracles outputs a valid signature and a statement Y for any new message m such that the valid signature does not reveal a witness for Y even given a valid pre-signature on m w.r.t. Y. The ECDSA-based and Schnorr-based adaptor signature schemes were constructed and analyzed in [1].

2.2 Notations

Throughout this work, we define different attribute tuples. Let T be a tuple of multiple attributes and one of its attributes is denoted by attr. To refer to this attribute, we use $T.\mathsf{attr}$.

Our focus in this work is on Bitcoin or any other blockchains with UTXO model. In this model, units of value which we call coins are held in *outputs*. Formally, an output θ is a tuple of two attributes, $\theta = (\mathsf{cash}, \varphi)$, where $\theta.\mathsf{cash}$ denotes the amount of coins held in this output and $\theta.\varphi$ denotes the condition that needs to be fulfilled to spend the output θ. The condition $\theta.\varphi$ is encoded using any script supported by the underlying blockchain. If the condition $\theta.\varphi$ contains a user P's public key, we say that P controls or owns the output θ because satisfying the condition requires a valid signature corresponding with that public key. Satisfying a condition might require authorizations by multiple parties. Such a condition contains public keys of all the involved parties separated by $\wedge$ operation(s). A condition might also have several subconditions, one of which must be satisfied to spend the output. Different subconditions of an output are separated by $\vee$ operation(s). The OP_RETURN output is a special output which does not hold any coins and is used to add some arbitrary data to the blockchain. Such an output is denoted by $\theta = (0, data)$ where $data$ is its arbitrary data.

A transaction changes ownership of coins, meaning that it takes a list of existing outputs and transfers their coins to a list of new outputs. To distinct between these two lists, we refer to the list of existing outputs as *inputs*. A transaction Tx is formally defined as the tuple $(\mathsf{txid}, \mathsf{Input}, \mathsf{Output}, \mathsf{Witness})$. The identifier $Tx.\mathsf{txid} \in \{0,1\}^*$ is computed as $Tx.\mathsf{txid} := H([Tx])$, where $[Tx]$ is called the *body* of the transaction defined as $[Tx] := (Tx.\mathsf{Input}, Tx.\mathsf{Output})$ and H is a hash function which is modeled as a random oracle. The attribute $Tx.\mathsf{Input}$ is a list of identifiers for all inputs of Tx. The attribute $Tx.\mathsf{Output} = (\theta_1, \ldots, \theta_n)$

is a list of new outputs. The attribute $Tx.\mathsf{Witness} = (\mathsf{W}_1, \ldots, \mathsf{W}_m)$ is a list of tuples where its i^{th} tuple authorizes spending the output that is taken as the i^{th} input of Tx. The tuple $\mathsf{W}_i = (\eta, \zeta)$ of the witness $Tx.\mathsf{Witness}$ contains two attributes where $\mathsf{W}_i.\zeta$ denotes the data, e.g. the signature(s), that is (are) required to meet the $\mathsf{W}_i.\eta^{\text{th}}$ subcondition of the output that is taken as the i^{th} input of Tx. The signature (pre-signature) of party P for $Tx.\mathsf{Witness}.\mathsf{W}_i.\zeta$ is denoted by $\sigma_{Tx}^{P,i}$ ($\tilde{\sigma}_{Tx}^{P,i}$), where i can be removed for transactions with single input. Table 1 summarizes the stated notations.

Table 1. Notations

Notation	Description
$T.\mathsf{attr}$	Attribute attr of the tuple T
$\theta = (\mathsf{cash}, \varphi)$	Output with value cash and script condition φ
$\theta = (0, data)$	OP_RETURN output with $data$ as its data
Tx	Transaction $Tx = (\mathsf{txid}, \mathsf{Input}, \mathsf{nLT}, \mathsf{Output}, \mathsf{Witness})$
$[Tx]$	Tuple $(Tx.\mathsf{Input}, Tx.\mathsf{Output})$
$Tx.\mathsf{txid}$	Identifier of the transaction Tx
$Tx.\mathsf{Input}$	List of identifiers for all inputs of Tx
$Tx.\mathsf{Output}$	List of new outputs $(\theta_1, \ldots, \theta_n)$ for Tx
$Tx.\mathsf{Witness}$	List of witnesses $(W_1, \ldots, W_m)$ for Tx where W_i corresponds with i^{th} input of Tx
$W = (\eta, \zeta)$	Witness that fulfills η^{th} subcondition of an output using data ζ
$\sigma_{Tx}^{P,i}$	Signature of party P on Tx for $Tx.\mathsf{Witness}.\mathsf{W}_i.\zeta$
$\tilde{\sigma}_{Tx}^{P,i}$	Pre-signature of party P on Tx for $Tx.\mathsf{Witness}.\mathsf{W}_i.\zeta$

We additionally use charts to illustrate the connections between different transactions. Transactions that are already published on-chain are illustrated by doubled edge rectangles. Transactions that are ready to be published are illustrated by single edge rectangles. Dotted edge rectangles show transactions that still lack the required witness for at least one input and hence are unprepared to be propagated in the blockchain network. Directional arrows from i^{th} output of transaction Tx to j^{th} input of transaction Tx' shows that the transaction Tx' takes i^{th} output of the transaction Tx as its j^{th} input. As an example, Fig. 1 illustrates that Tx_i and Tx_j are published and unpublished, respectively. The transaction Tx_k is still unprepared to be published on the ledger. According to the figure, we have: $Tx_i.\mathsf{Output} := \{(x, \varphi_1 \vee \varphi_2)\}$, $Tx_j.\mathsf{Input} = Tx_i.\mathsf{txid}\|1$ (the string obtained from concatenation of $Tx_i.\mathsf{txid}$ and the number 1, representing the 1st output of Tx_i), $Tx_j.\mathsf{Output} = (\theta_1, \theta_2)$, and $Tx_j.\mathsf{Witness} = \{(1, \zeta)\}$ (showing that $Tx_j.\mathsf{Witness}$ satisfies the first subcondition of $Tx_i.\mathsf{Output}$, φ_1, using the data ζ). The output θ_2 is an OP_RETURN output.

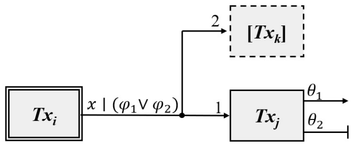


Fig. 1. A sample transaction flow.

3 Garrison Overview

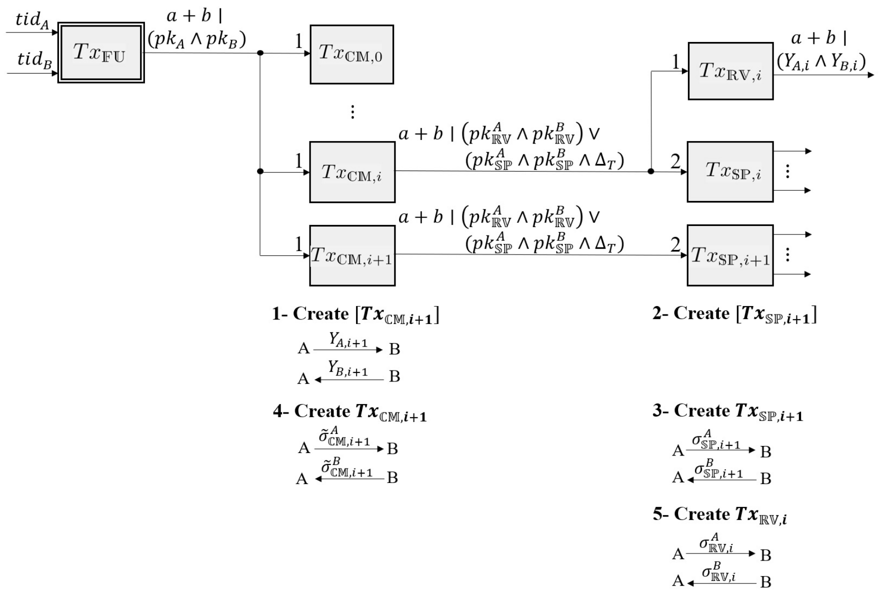
In this section we start with a simple payment channel introduced in [9]. Although this scheme provide state duplication avoidance, its parties' and watchtower's storage requirements linearly increase with each channel update. Then, we modify this simple scheme step by step to mitigate its limitations.

3.1 A Simple Payment Channel

Fig. 2 depicts the simple payment channel, introduced in [9]. This simple channel is created once channel parties publish a funding transaction on the blockchain. By doing so, they fund a 2-of-2 multisignature output on the ledger. The i^{th} channel state includes a commit transaction $Tx_{\mathbb{CM},i}$ as well as a split transaction $Tx_{\mathbb{SP},i}$. The commit transaction sends the channel funds to a new joint account which is shared between the channel parties. Output of the commit transaction has two subconditions where its second subcondition is relatively timelocked by T rounds and can be satisfied after T rounds by the corresponding split transaction and the first subcondition which is not timelocked, as we will explain later, is used for revocation purposes. Split transaction distributes the channel funds among the channel parties and hence represents the channel state.

The transaction $Tx_{\mathbb{CM},i}$ requires signatures of both parties A and B to be published. To generate $\sigma^{B}_{\mathbb{CM},i}$, party A generates a statement/witness pair $(Y_{A,i}, y_{A,i})$ and sends the statement $Y_{A,i}$ to B. Then, party B uses the pre-signing algorithm pSign of the adaptor signature and A's statement $Y_{A,i}$ to generate a pre-signature $\tilde{\sigma}^{B}_{\mathbb{CM},i}$ on $[Tx_{\mathbb{CM},i}]$ and sends the result to A. Thus, whenever it is necessary, A is able to use the adaptation algorithm adapt of the adaptor signature to transform the pre-signature to the signature $\sigma^{B}_{\mathbb{CM},i}$ and publish $Tx_{\mathbb{CM},i}$ on-chain. This also enables B to use the extraction algorithm $\mathsf{Extract}$, the published signature and its corresponding pre-signature to extract the witness value $y_{A,i}$. The witness value, as will be seen, allows the honest party to punish his dishonest channel party by claiming all the channel funds.

As one may submit an intermediate state (which is already replaced by a later state) to the blockchain, the channel parties will need to detect and punish such misbehaviours. To achieve this goal, upon channel update from state i to $i+1$, a revocation transaction is created by parties. This revocation transaction corresponds to the state i of the channel. Unlike the split transaction, the revocation transaction can immediately spend output of the corresponding commit transaction $Tx_{\mathbb{CM},i}$ using its first subcondition which does not contain any timelock. Thus, if the revoked commit transaction $Tx_{\mathbb{CM},i}$ is published by a channel party, let's say A, party B can immediately publish the revocation transaction $Tx_{\mathbb{RV},i}$. Moreover, since commit transactions are signed using the adaptor signature, once $Tx_{\mathbb{CM},i}$ is published by A, his witness $y_{A,i}$ is revealed to B. Thus, only B who knows both $y_{A,i}$ and $y_{B,i}$ can claim the output of the revocation transaction and hence he will be actually the owner of all the channel funds. Broadcast of the latest commit transaction does not pose any risk to its broadcaster because parties have not signed the revocation transaction for the latest state yet.



tid_A
tid_B
Tx_{FU}
$a+b \mid (pk_A \wedge pk_B)$
1
$Tx_{\text{CM},0}$
$Tx_{\text{CM},i}$
$Tx_{\text{CM},i+1}$
$a+b \mid (pk^A_{\mathbb{RV}} \wedge pk^B_{\mathbb{RV}}) \vee (pk^A_{\mathbb{SP}} \wedge pk^B_{\mathbb{SP}} \wedge \Delta_T)$
2
$Tx_{\mathbb{RV},i}$
$a+b \mid (Y_{A,i} \wedge Y_{B,i})$
$Tx_{\mathbb{SP},i}$
$Tx_{\mathbb{SP},i+1}$
1- Create $[Tx_{\mathbb{CM},i+1}]$
$A \xrightarrow{Y_{A,i+1}} B$
$A \xleftarrow{Y_{B,i+1}} B$
2- Create $[Tx_{\mathbb{SP},i+1}]$
3- Create $Tx_{\mathbb{SP},i+1}$
$A \xrightarrow{\sigma^A_{\mathbb{SP},i+1}} B$
$A \xleftarrow{\sigma^B_{\mathbb{SP},i+1}} B$
4- Create $Tx_{\mathbb{CM},i+1}$
$A \xrightarrow{\tilde{\sigma}^A_{\mathbb{CM},i+1}} B$
$A \xleftarrow{\tilde{\sigma}^B_{\mathbb{CM},i+1}} B$
5- Create $Tx_{\mathbb{RV},i}$
$A \xrightarrow{\sigma^A_{\mathbb{RV},i}} B$
$A \xleftarrow{\sigma^B_{\mathbb{RV},i}} B$

Fig. 2. A Simple Payment Channel

3.2 Reducing the Storage Requirements of the Watchtower

For the simple scheme, introduced in the previous section, all revocation transactions must be stored by channel parties or their watchtowers to be published upon fraud. In this section, based on the same idea as Outpost [6], we reduce the storage requirements of the watchtower. To achieve this goal, the main idea is storing the revocation transaction $Tx_{\mathbb{RV},i}$ inside the commit transaction $Tx_{\mathbb{CM},i}$. Then, once $Tx_{\mathbb{CM},i}$ is published, the watchtower extracts $Tx_{\mathbb{RV},i}$ and records it on the blockchain. However, we have $Tx_{\mathbb{RV},i}.\mathsf{Input} = Tx_{\mathbb{CM},i}.\mathsf{txid}\|1$. Thus, if $Tx_{\mathbb{RV},i}$ is created, signed and finally stored inside $Tx_{\mathbb{CM},i}$, then $[Tx_{\mathbb{CM},i}]$ and hence $Tx_{\mathbb{CM},i}.\mathsf{txid}$ and $Tx_{\mathbb{RV},i}$ change. Thus, there is a self-loop situation [6].

To solve this issue, we add an auxiliary output with the value of ϵ to commit transactions where ϵ is the minimum value supported by the Bitcoin blockchain. We also add an auxiliary transaction between each commit transaction and its corresponding split transaction. This new transaction $Tx_{\mathbb{AU},i}$ spends the auxiliary output of the commit transaction. The signatures of party A and party B on $[Tx_{\mathbb{RV},i}]$ are stored in an OP_RETURN output of the auxiliary transaction $Tx_{\mathbb{AU},i}$. The split transaction $Tx_{\mathbb{SP},i}$ spends the main output of $Tx_{\mathbb{CM},i}$ as well as the main output of the auxiliary transaction $Tx_{\mathbb{AU},i}$. Based on this design, parties can be sure that once the revoked commit transaction $Tx_{\mathbb{CM},i}$ is published on the blockchain, its split transaction $Tx_{\mathbb{SP},i}$ cannot be published unless $Tx_{\mathbb{AU},i}$ is also published on the blockchain. Furthermore, due to the timelock in the main output of $Tx_{\mathbb{AU},i}$, once this transaction is published on-chain, $Tx_{\mathbb{SP},i}$ cannot be published within $T-1$ rounds. However, the honest party or his watchtower can extract the signatures on $[Tx_{\mathbb{RV},i}]$ from $Tx_{\mathbb{AU},i}$ and publish $Tx_{\mathbb{RV},i}$ immediately. Fig. 3 depicts the transactions flows.

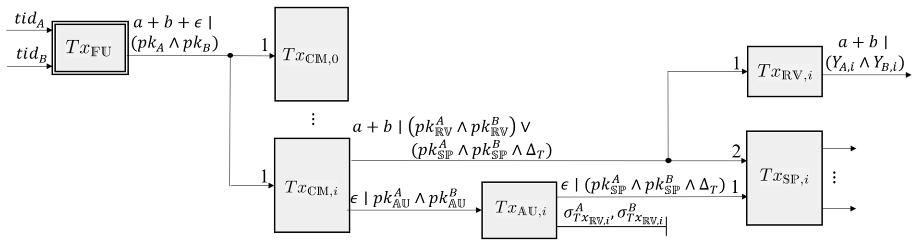

tid_A
tid_B
Tx_{FU}
$a+b+\epsilon \mid (pk_A \wedge pk_B)$
1
$Tx_{\mathbb{CM},0}$
⋮
1
$Tx_{\mathbb{CM},i}$
$a+b \mid (pk^A_{\mathbb{RV}} \wedge pk^B_{\mathbb{RV}}) \vee (pk^A_{\mathbb{SP}} \wedge pk^B_{\mathbb{SP}} \wedge \Delta_T)$
$\epsilon \mid pk^A_{\mathbb{AU}} \wedge pk^B_{\mathbb{AU}}$
$Tx_{\mathbb{AU},i}$
$\epsilon \mid (pk^A_{\mathbb{SP}} \wedge pk^B_{\mathbb{SP}} \wedge \Delta_T)$ 1
$\sigma^A_{Tx_{\mathbb{RV},i}}, \sigma^B_{Tx_{\mathbb{RV},i}}$
1
$Tx_{\mathbb{RV},i}$
$a+b \mid (Y_{A,i} \wedge Y_{B,i})$
2
$Tx_{\mathbb{SP},i}$
⋮

Fig. 3. Reducing the Storage Requirements of the Watchtower

However, this scheme does not work as it has two following issues:

- If the watchtower is supposed to create the revocation transaction and publish it on the blockchain, he must also know the value of $Y_{A,i}$ and $Y_{B,i}$.
- Typically, revocation transaction of state i must be created once parties update the channel state from state i to $i+1$. However, in the proposed scheme signatures on revocation transaction of state i is stored in auxiliary transaction of state i. It means that if an honest party records the latest commit and auxiliary transaction on the blockchain, her counter-party might publish the revocation transaction and take all the channel funds.

To solve the first mentioned issue, $Y_{A,i}$ and $Y_{B,i}$ are stored in an OP_RETURN output in the commit transaction $Tx_{\mathbb{CM},i}$. To solve the second mentioned issue, we add two statements from the hard relation $\mathcal{R}$, $R_{A,i}$ and $R_{B,i}$, to the first subcondition of the main output of $Tx_{\mathbb{CM},i}$, where $R_{A,i}$ ($R_{B,i}$) is generated by A (B) for the state i. Then, once the latest commit and auxiliary transactions are published by A, party B cannot record the revocation transaction as he does not know his counter-party's witness $r_{A,i}$. The witnesses $r_{A,i}$ and $r_{B,i}$ are exchanged between the parties and are given to the watchtower once parties have created $Tx_{\mathbb{CM},i+1}$, $Tx_{\mathbb{AU},i+1}$ and $Tx_{\mathbb{SP},i+1}$. Thus, $Tx_{\mathbb{FU}}$.txid, public keys $pk^A_{\mathbb{RV}}$, $pk^B_{\mathbb{RV}}$, $pk^A_{\mathbb{SP}}$ and $pk^B_{\mathbb{SP}}$ as well as r values of both parties are all data needed by the watchtower to watch the channel for both parties. In Section 5 we will explain that if parties generate their r values in a Merkle tree and use them from the deepest point of the tree to the top, then the storage requirements of the watchtower would be $\mathcal{O}(log(N))$. Fig. 4 depicts the mentioned modifications.

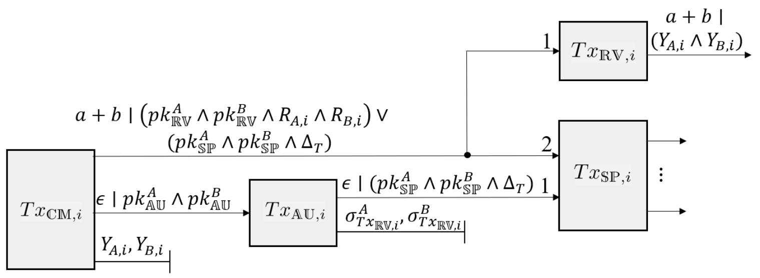
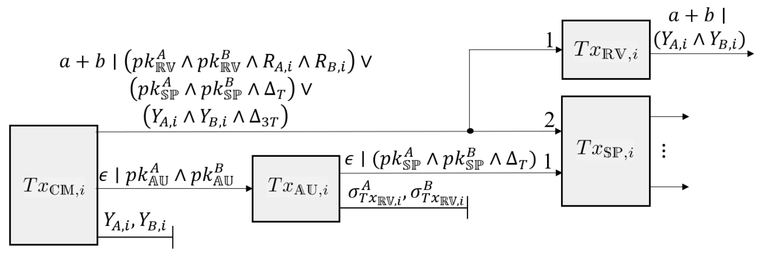

$Tx_{\mathbb{CM},i}$
$a+b \mid (pk^A_{\mathbb{RV}} \wedge pk^B_{\mathbb{RV}} \wedge R_{A,i} \wedge R_{B,i}) \vee (pk^A_{\mathbb{SP}} \wedge pk^B_{\mathbb{SP}} \wedge \Delta_T)$
$\epsilon \mid pk^A_{\mathbb{AU}} \wedge pk^B_{\mathbb{AU}}$
$Y_{A,i}, Y_{B,i}$
$Tx_{\mathbb{AU},i}$
$\epsilon \mid (pk^A_{\mathbb{SP}} \wedge pk^B_{\mathbb{SP}} \wedge \Delta_T)$
$\sigma^A_{Tx_{\mathbb{RV},i}}, \sigma^B_{Tx_{\mathbb{RV},i}}$
1
2
1
$Tx_{\mathbb{RV},i}$
$a+b \mid (Y_{A,i} \wedge Y_{B,i})$
$Tx_{\mathbb{SP},i}$
⋮

Fig. 4. Reducing the Storage Requirements of the Watchtower

3.3 Reducing the Storage Requirements of channel parties

Although in the introduced scheme, the storage of the watchtower is $\mathcal{O}(log(N))$, channel parties still have to store all the signatures of their counter-parties on the revocation transactions. Otherwise, the dishonest channel party publishes a revoked commit transaction $Tx_{\mathbb{CM},i}$ without publishing its auxiliary transaction $Tx_{\mathbb{AU},i}$. Then, the channel funds could be locked forever. This raises a hostage situation. The scheme Outpost suffers from this problem which is why for this scheme, storage requirement of channel parties is $\mathcal{O}(N)$. To solve this problem, one subcondition, $Y_{A,i} \wedge Y_{B,i} \wedge \Delta_{3T}$, is added to the main output of the commit transaction $Tx_{\mathbb{CM},i}$. This subcondition allows the honest channel party to claim all the channel funds in such hostage situations. In other words, if party A publishes the revoked commit transaction $Tx_{\mathbb{CM},i}$, he has $3T$ rounds time to publish $Tx_{\mathbb{AU},i}$ and $Tx_{\mathbb{SP},i}$ before B claiming the channel funds with meeting the subcondition $Y_{A,i} \wedge Y_{B,i} \wedge \Delta_{3T}$. If during this interval, $Tx_{\mathbb{AU},i}$ is published, party B instantly establishes and publishes $Tx_{\mathbb{RV},i}$ and claims its output. To do so, each party must have r values of both parties stored. Since these keys are generated in a Merkle tree, the storage requirements of each channel party for storing these values would be $\mathcal{O}(log(N))$.

Once party A publishes $Tx_{\mathbb{CM},i}$, party B must be able to extract the value of $y_{A,i}$. To do so, he must know the corresponding pre-signature $\tilde{\sigma}^B_{Tx_{\mathbb{CM},i}}$. The naive method is storing all these pre-signatures. However, this method requires storage of $\mathcal{O}(N)$. To avoid this, parties must be able to regenerate the corresponding pre-signature once a commit transaction is published. To achieve this goal, random values which are required to generate pre-signatures must be generated in a Merkle tree and are used from the top to the deepest point in the tree. In this way, once the commit transaction $Tx_{\mathbb{CM},i}$ is published by party A, party B can regenerate the required random values and recompute the corresponding pre-signature $\tilde{\sigma}^{B}_{Tx_{\mathbb{CM},i}}$ and extract the value of $y_{A,i}$. Thus, the storage requirements would be still $\mathcal{O}(log(N))$. More details will be provided in Section 5. Fig. 5 depicts the mentioned modifications.



$a+b \mid (pk^A_{\mathbb{RV}} \wedge pk^B_{\mathbb{RV}} \wedge R_{A,i} \wedge R_{B,i}) \vee$
$(pk^A_{\mathbb{SP}} \wedge pk^B_{\mathbb{SP}} \wedge \Delta_T) \vee$
$(Y_{A,i} \wedge Y_{B,i} \wedge \Delta_{3T})$
$Tx_{\mathbb{CM},i}$
$\epsilon \mid pk^A_{\mathbb{AU}} \wedge pk^B_{\mathbb{AU}}$
$Y_{A,i}, Y_{B,i}$
$Tx_{\mathbb{AU},i}$
$\epsilon \mid (pk^A_{\mathbb{SP}} \wedge pk^B_{\mathbb{SP}} \wedge \Delta_T)$
$\sigma^A_{Tx_{\mathbb{RV},i}}, \sigma^B_{Tx_{\mathbb{RV},i}}$
1
2
1
$Tx_{\mathbb{RV},i}$
$a+b \mid$
$(Y_{A,i} \wedge Y_{B,i})$
$Tx_{\mathbb{SP},i}$
⋮

Fig. 5. Reducing the Storage Requirements of Channel Parties

4 Garrison Channel

We introduce different transactions of a Garrison channel in section 4.1. Then, in section 4.2, we explain its protocol.

4.1 Garrison Transactions

In this section, different transactions of a Daric channel are introduced.

Funding transaction A Daric channel is created once channel parties A and B, by recording the funding transaction of the channel on the ledger, fund it with the initial balance of a and b coins, respectively. The output of funding transaction is a 2-of-2 multisignature address shared between A and B. Assuming that A (B) funds the channel using x^{th} (y^{th}) output of a transaction with transaction identifier of $txid_A$ ($txid_B$), the funding transaction, denoted by $Tx_{\mathbb{FU}}$, is as follows[1]:

$$Tx_{\mathbb{FU}}.\mathsf{Input} := (txid_A \| x, txid_B \| y),$$
$$Tx_{\mathbb{FU}}.\mathsf{Output} := \{(a+b, pk_A \wedge pk_B)\},$$
$$Tx_{\mathbb{FU}}.\mathsf{Witness} := ((1, \sigma^{A,1}_{Tx_{\mathbb{FU}}}), (1, \sigma^{B,2}_{Tx_{\mathbb{FU}}})).$$

[1] We assume that funding sources of $Tx_{\mathbb{FU}}$ are two typical UTXOs owned by A and B.

Commit transaction The commit transaction for state i is denoted by $Tx_{\mathbb{CM},i}$ and is as follows:

$$\begin{aligned}
Tx_{\mathbb{CM},i}.\mathsf{Input} &:= Tx_{\mathbb{FU}}.\mathsf{txid}\|1,\\
Tx_{\mathbb{CM},i}.\mathsf{Output} &:= ((a+b, \varphi_1 \vee \varphi_2 \vee \varphi_3),\\
&\quad\ (\epsilon, pk^A_{\mathbb{AU}} \wedge pk^B_{\mathbb{AU}}),\\
&\quad\ (0, (Y_{A,i}, Y_{B,i})))\\
Tx_{\mathbb{CM},i}.\mathsf{Witness} &:= \{(1, \{\sigma^A_{Tx_{\mathbb{CM},i}}, \sigma^B_{Tx_{\mathbb{CM},i}}\})\}
\end{aligned}$$

with $\varphi_1 := (pk^A_{\mathbb{RV}} \wedge pk^B_{\mathbb{RV}} \wedge R_{A,i} \wedge R_{B,i})$, $\varphi_2 := (pk^A_{\mathbb{SP}} \wedge pk^B_{\mathbb{SP}} \wedge \Delta_T)$, and $\varphi_3 := (Y_{A,i} \wedge Y_{B,i} \wedge \Delta_{3T})$ where $Y_{A,i}$ and $R_{A,i}$ ($Y_{B,i}$ and $R_{B,i}$) are statements of a hard relation $\mathcal{R}$ generated by A (B) for the i^{th} state. The first and second output of the transaction are the main and auxiliary outputs. Normally, if $Tx_{\mathbb{CM},i}$ is the last commit transaction and is published on-chain, first its auxiliary output and then its main output are spent by the auxiliary and split transactions, respectively. The third output of $Tx_{\mathbb{CM},i}$ is an OP_RETURN output containing values of $Y_{A,i}$ and $Y_{B,i}$. Parties A and B use their counter-parties' statements $Y_{B,i}$ and $Y_{A,i}$ and the underlying adaptor signature to generate a pre-signature on the commit transaction for their counter-parties. Thus, once A publishes the commit transaction $Tx_{\mathbb{CM},i}$, he also reveals his witness $y_{B,i}$.

Remark 1. Each Bitcoin transaction can have at most one OP_RETURN output with the size constraint of 80 bytes. To store $Y_{A,i}$ and $Y_{B,i}$ inside an OP_RETURN output, their compressed version, each with the length of 33 bytes, can be stored.

Revocation transaction The revocation transaction for state i is denoted by $Tx_{\mathbb{RV},i}$ and is as follows:

$$\begin{aligned}
Tx_{\mathbb{RV},i}.\mathsf{Input} &:= Tx_{\mathbb{CM}}.\mathsf{txid}\|1,\\
Tx_{\mathbb{AU},i}.\mathsf{Output} &:= \{(a+b, Y_{A,i} \wedge Y_{B,i})\},\\
Tx_{\mathbb{AU},i}.\mathsf{Witness} &:= \{(1, \{\sigma^A_{Tx_{\mathbb{RV},i}}, \sigma^B_{Tx_{\mathbb{RV},i}}, r_{A,i}, r_{B,i}\})\}
\end{aligned}$$

The $Tx_{\mathbb{RV},i}$ spends the main output of $Tx_{\mathbb{CM},i}$ using its non-timelocked subcondition $pk^A_{\mathbb{RV}} \wedge pk^B_{\mathbb{RV}} \wedge R_{A,i} \wedge R_{B,i}$ and sends all the channel funds to an output with the condition $Y_{A,i} \wedge Y_{B,i}$. When a dishonest party, let's say A, publishes the revoked $Tx_{\mathbb{CM},i}$, he must publish $Tx_{\mathbb{AU},i}$ and then wait for T rounds before being able to publish $Tx_{\mathbb{SP},i}$. However, given that the state i is revoked, B knows the value of $r_{A,i}$ and hence can create the revocation transaction $Tx_{\mathbb{RV},i}$ and instantly publish it on the blockchain. The output of $Tx_{\mathbb{RV},i}$ can only be claimed by B because no one else knows the witness $y_{B,i}$.

Auxiliary transaction Auxiliary transaction for state i is as follows:

$$\begin{aligned}
Tx_{\mathbb{AU},i}.\mathsf{Input} &:= Tx_{\mathbb{CM}}.\mathsf{txid}\|2,\\
Tx_{\mathbb{AU},i}.\mathsf{Output} &:= ((\epsilon, pk^A_{\mathbb{SP}} \wedge pk^B_{\mathbb{SP}} \wedge \varDelta_T),\\
&\quad\ (0, (\sigma^A_{Tx_{\mathbb{RV},i}}, \sigma^B_{Tx_{\mathbb{RV},i}})))\\
Tx_{\mathbb{CM},i}.\mathsf{Witness} &:= \{(1, \{\sigma^A_{Tx_{\mathbb{CM},i}}, \sigma^B_{Tx_{\mathbb{CM},i}}\})\}
\end{aligned}$$

This transaction spends the auxiliary output of the commit transaction and its output is spent by the split transaction. In other words, split transaction cannot be published unless auxiliary transaction is on the blockchain. The second output of $Tx_{\mathbb{AU},i}$ is an OP_RETURN output containing signatures of both parties on the corresponding revocation transaction.

Remark 2. Each encoded Bitcoin signature can be up to 73 bytes long. Thus, due to the size constraint of the OP_RETURN output, two separate signatures do not fit into the auxiliary transaction. To solve this issue, parties A and B can aggregate their public keys $pk^A_{\mathbb{RV}}$ and $pk^B_{\mathbb{RV}}$ to form an aggregated public key $pk_{\mathbb{RV}}$ [7] and change φ_1 in $Tx_{\mathbb{CM},i}$ to $(pk_{\mathbb{RV}} \wedge R_{A,i} \wedge R_{B,i})$. Then, rather than two separate signatures on the revocation transaction, they generate a multisignature (with up to 73 byte size) and store it inside the OP_RETURN output of the auxiliary transaction [7].

Split transaction $Tx_{\mathbb{SP},i}$ actually represents the i^{th} channel state and is as follows:

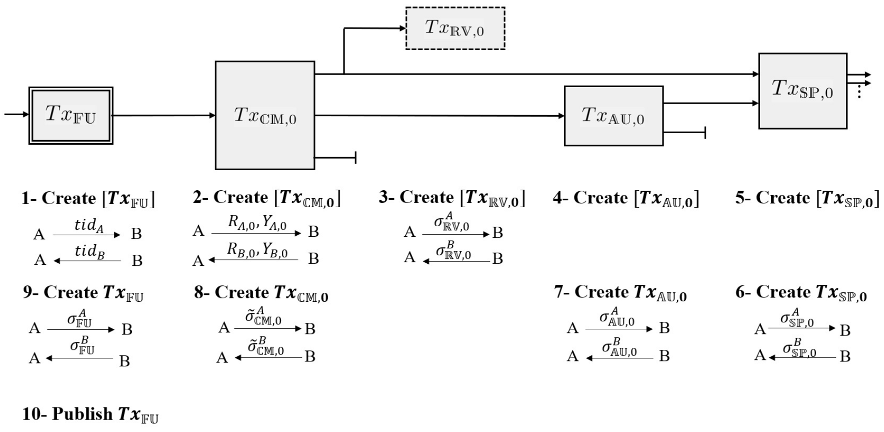
$$\begin{aligned}
Tx_{\mathbb{SP},i}.\mathsf{Input} &:= (Tx_{\mathbb{CM},i}.\mathsf{txid}\|1, Tx_{\mathbb{AU},i}.\mathsf{txid}\|1),\\
Tx_{\mathbb{SP},i}.\mathsf{Output} &:= (\theta_1, \theta_2, \cdots),\\
Tx_{\mathbb{SP},i}.\mathsf{Witness} &:= ((2, \{\sigma^A_{Tx_{\mathbb{SP},i}}, \sigma^B_{Tx_{\mathbb{SP},i}}\}), (1, \{\sigma^A_{Tx_{\mathbb{SP},i}}, \sigma^B_{Tx_{\mathbb{SP},i}}\}))
\end{aligned}$$

The split transaction spends the main output of the commit transaction as well as the first output of the auxiliary transaction.

4.2 Garrison Protocol

The lifetime of a Garrison channel can be divided into 4 phases including "create", "update", "close", and "Punish". These phases are explained, hereinafter.

Create The channel creation phase completes once the funding transaction $Tx_{\mathbb{FU}}$, the commit transactions $Tx_{\mathbb{CM},0}$, the split transaction $Tx_{\mathbb{SP},0}$, the auxiliary transaction $Tx_{\mathbb{AU},0}$ and body of the revocation transaction $[Tx_{\mathbb{RV},0}]$ are created, and $Tx_{\mathbb{FU}}$ is published on the blockchain. In this phase, parties do not have access to $Tx_{\mathbb{RV},0}$ as they have not exchanged $r_{A,i}$ and $r_{B,i}$ yet. At the end of the channel creation phase, the channel would be at state 0. Since output of the funding transaction can only be spent if both parties agree, one party might become unresponsive to raise a hostage situation. To avoid this, parties must sign the commit, revocation, auxiliary and split transactions before signing and publishing the funding transaction. Fig. 6 summarizes the channel creation phase.



$Tx_{\mathbb{RV},0}$
$Tx_{\mathbb{FU}}$
$Tx_{\mathbb{CM},0}$
$Tx_{\mathbb{AU},0}$
$Tx_{\mathbb{SP},0}$
1- Create $[Tx_{\mathbb{FU}}]$
A $\xrightarrow{tid_A}$ B
A $\xleftarrow{tid_B}$ B
2- Create $[Tx_{\mathbb{CM},0}]$
A $\xrightarrow{R_{A,0}, Y_{A,0}}$ B
A $\xleftarrow{R_{B,0}, Y_{B,0}}$ B
3- Create $[Tx_{\mathbb{RV},0}]$
A $\xrightarrow{\sigma^A_{\mathbb{RV},0}}$ B
A $\xleftarrow{\sigma^B_{\mathbb{RV},0}}$ B
4- Create $[Tx_{\mathbb{AU},0}]$
5- Create $[Tx_{\mathbb{SP},0}]$
9- Create $Tx_{\mathbb{FU}}$
A $\xrightarrow{\sigma^A_{\mathbb{FU}}}$ B
A $\xleftarrow{\sigma^B_{\mathbb{FU}}}$ B
8- Create $Tx_{\mathbb{CM},0}$
A $\xrightarrow{\tilde{\sigma}^A_{\mathbb{CM},0}}$ B
A $\xleftarrow{\tilde{\sigma}^B_{\mathbb{CM},0}}$ B
7- Create $Tx_{\mathbb{AU},0}$
A $\xrightarrow{\sigma^A_{\mathbb{AU},0}}$ B
A $\xleftarrow{\sigma^B_{\mathbb{AU},0}}$ B
6- Create $Tx_{\mathbb{SP},0}$
A $\xrightarrow{\sigma^A_{\mathbb{SP},0}}$ B
A $\xleftarrow{\sigma^B_{\mathbb{SP},0}}$ B
10- Publish $Tx_{\mathbb{FU}}$

Fig. 6. Summary of Daric channel creation phase.

Update Let the channel be in state $i \geq 0$ and channel parties decide to update it to state $i+1$. The update process is performed in two sub-phases. In the first sub-phase, channel parties create $Tx_{\mathbb{CM},i+1}$, $Tx_{\mathbb{SP},i+1}$, $Tx_{\mathbb{AU},i+1}$, and $[Tx_{\mathbb{RV},i+1}]$ for the new state. In the second sub-phase, channel parties revoke the state i by exchanging $r_{A,i}$ and $r_{B,i}$. Fig. 7 summarizes the channel update phase.

Close Assume that the channel parties A and B have updated their channel n times and then A and/or B decide to close it. They can close the channel cooperatively. To do so, A and B create a new transaction, called modified split transaction $Tx_{\overline{\mathbb{SP}}}$, and publish it on the blockchain. For $Tx_{\overline{\mathbb{SP}}}$ we have:

$$Tx_{\overline{\mathbb{SP}}}.\mathsf{Input} := Tx_{\mathbb{FU}}.\mathsf{txid}\|1,$$
$$Tx_{\overline{\mathbb{SP}}}.\mathsf{Output} := Tx_{\mathbb{SP},n}.\mathsf{Output},$$
$$Tx_{\overline{\mathbb{SP}}}.\mathsf{Witness} := \{(1, \{\sigma^A_{Tx_{\overline{\mathbb{SP}}}}, \sigma^B_{Tx_{\overline{\mathbb{SP}}}}\})\}.$$

If one of the channel parties, e.g. party B, becomes unresponsive, its counterparty A can still non-collaboratively close the channel. To do so, he publishes $Tx_{\mathbb{CM},n}$ and $Tx_{\mathbb{AU},n}$ on the ledger. Then, he waits for T rounds, and finally publishes $Tx_{\mathbb{SP},n}$.

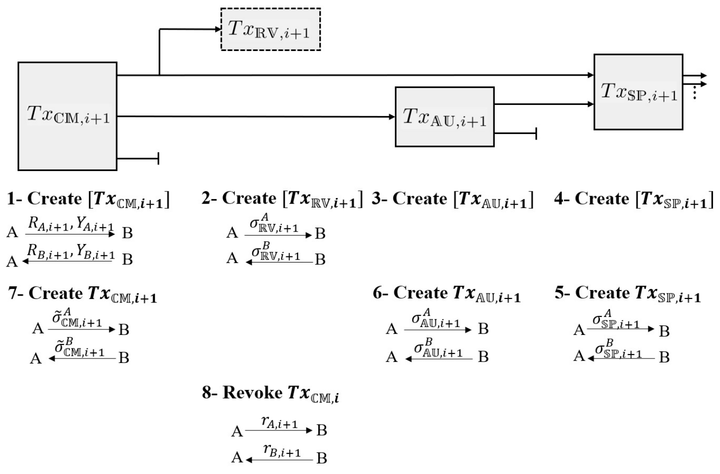

$Tx_{\mathbb{RV},i+1}$
$Tx_{\mathbb{CM},i+1}$
$Tx_{\mathbb{AU},i+1}$
$Tx_{\mathbb{SP},i+1}$
1- Create [$Tx_{\mathbb{CM},i+1}$]
$A \xrightarrow{R_{A,i+1}, Y_{A,i+1}} B$
$A \xleftarrow{R_{B,i+1}, Y_{B,i+1}} B$
2- Create [$Tx_{\mathbb{RV},i+1}$]
$A \xrightarrow{\sigma^A_{\mathbb{RV},i+1}} B$
$A \xleftarrow{\sigma^B_{\mathbb{RV},i+1}} B$
3- Create [$Tx_{\mathbb{AU},i+1}$]
4- Create [$Tx_{\mathbb{SP},i+1}$]
7- Create $Tx_{\mathbb{CM},i+1}$
$A \xrightarrow{\tilde{\sigma}^A_{\mathbb{CM},i+1}} B$
$A \xleftarrow{\tilde{\sigma}^B_{\mathbb{CM},i+1}} B$
6- Create $Tx_{\mathbb{AU},i+1}$
$A \xrightarrow{\sigma^A_{\mathbb{AU},i+1}} B$
$A \xleftarrow{\sigma^B_{\mathbb{AU},i+1}} B$
5- Create $Tx_{\mathbb{SP},i+1}$
$A \xrightarrow{\sigma^A_{\mathbb{SP},i+1}} B$
$A \xleftarrow{\sigma^B_{\mathbb{SP},i+1}} B$
8- Revoke $Tx_{\mathbb{CM},i}$
$A \xrightarrow{r_{A,i+1}} B$
$A \xleftarrow{r_{B,i+1}} B$

Fig. 7. Summary of Daric channel update phase from state i to $i + 1$.

Punish Let the channel be in state n. If a channel party, e.g. party A, publishes $Tx_{\mathbb{CM},i}$ and then $Tx_{\mathbb{AU},i}$ with $i < n$ on the blockchain, party B or his watchtower can create the transaction $Tx_{\mathbb{RV},i}$ and publish it within T rounds. If only $Tx_{\mathbb{CM},i}$ is published, party B claims its first output by meeting its third subcondition $Y_{A,i} \wedge Y_{A,i} \wedge \Delta_{3T}$.

5 Optimization

Assuming that party B publishes the revoked commit transaction $Tx_{\mathbb{CM},i}$ on the ledger, A requires:

- $r_{A,i}$ and $r_{B,i}$ to publish $Tx_{\mathbb{RV},i}$,
- $\tilde{\sigma}^A_{Tx_{\mathbb{CM},i}}$ to extract $y_{B,i}$, and
- $y_{A,i}$ to be used along with $y_{B,i}$ to claim the output of $Tx_{\mathbb{RV},i}$ or to meet the third subcondition of the main output of $Tx_{\mathbb{CM},i}$.

However, if these values for each state are stored independently, the required storage would be $\mathcal{O}(N)$. We optimize this storage and reduce the storage to $\mathcal{O}(log(N))$.

We start with r values. The security requirement for r values is that B must not be able to compute $r_{A,j}$ given that he knows $r_{A,i}$ with $i < j$. Otherwise, when A submits the latest commit transaction $Tx_{\mathbb{CM},j}$, party B using $r_{A,i}$ computes $r_{A,j}$ and hence publishes the revocation transaction $Tx_{\mathbb{RV},i}$ and claims its output. If r values are randomly generated, the mentioned security requirement is met but party A's storage requirement would be $\mathcal{O}(N)$. To reduce the storage and meeting the stated security requirement, parties generate their r values in a Merkle tree and give the corresponding R values to their counter-parties from the deepest point in the tree to the top.

A similar approach can be adopted for y values. However, the security requirement for y values is that B must not be able to compute $y_{A,i}$ given that he knows $y_{A,j}$ with $j > i$. Otherwise, once A submits the latest commit transaction $Tx_{\mathbb{CM},j}$, B computes $y_{A,j}$ and hence derives $y_{A,i}$ with $i < j$ and then try to publish $Tx_{\mathbb{CM},i}$ before $Tx_{\mathbb{CM},j}$ being published on the ledger. Then, B might be able to claim all the channel funds by meeting the third sub-condition of the main output of $Tx_{\mathbb{CM},i}$ or by publishing the revocation transaction $Tx_{\mathbb{RV},i}$ and claiming its output. If y values are randomly generated, the mentioned security requirement is met but party A's storage requirement would be $\mathcal{O}(N)$. To reduce the storage and meeting the stated security requirement, A generates his y values in a Merkle tree and gives the corresponding Y values to his counter-party from the top to the deepest point in the tree.

We assume that once a revoked commit transaction $Tx_{\mathbb{CM},i}$ is published, the honest party A computes the value of the witness $y_{B,i}$. Otherwise, he cannot punish his dishonest counter-party. To extract $y_{B,i}$, both signature $\sigma^A_{Tx_{\mathbb{CM},i}}$ and pre-signature $\tilde{\sigma}^A_{Tx_{\mathbb{CM},i}}$ are required. Since the published transaction $Tx_{\mathbb{CM},i}$ contains $\sigma^A_{Tx_{\mathbb{CM},i}}$, if A has stored either all the pre-signatures or all the random values which are required to generate the pre-signatures, then he can extract $y_{B,i}$. However, it would require storage requirement of $\mathcal{O}(N)$. To reduce the storage, A generates the random values which are required to generate the pre-signatures in a Merkle tree and use those random values from the top to the deepest point in the tree.

References

1. Aumayr, L., Ersoy, O., Erwig, A., Faust, S., Hostakova, K., Maffei, M., Moreno-Sanchez, P., Riahi, S.: Generalized bitcoin-compatible channels. IACR Cryptol. ePrint Arch. **2020**, 476 (2020)
2. Avarikioti, G., Laufenberg, F., Sliwinski, J., Wang, Y., Wattenhofer, R.: Towards secure and efficient payment channels. arXiv preprint arXiv:1811.12740 (2018)
3. Avarikioti, G., Litos, O.S.T., Wattenhofer, R.: Cerberus channels: Incentivizing watchtowers for bitcoin. Financial Cryptography and Data Security (FC) (2020)
4. Dryja, T., Milano, S.B.: Unlinkable outsourced channel monitoring. Talk transcript) https://diyhpl. us/wiki/transcripts/scalingbitcoin/milan/unlinkable-outsourced-channel-monitoring (2016)
5. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ecdsa). International journal of information security **1**(1), 36–63 (2001)
6. Khabbazian, M., Nadahalli, T., Wattenhofer, R.: Outpost: A responsive lightweight watchtower. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies. pp. 31–40 (2019)
7. Lindell, Y.: Fast secure two-party ecdsa signing. In: Annual International Cryptology Conference. pp. 613–644. Springer (2017)
8. McCorry, P., Bakshi, S., Bentov, I., Meiklejohn, S., Miller, A.: Pisa: Arbitration outsourcing for state channels. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies. pp. 16–30 (2019)
9. Mirzaei, A., Sakzad, A., Yu, J., Steinfeld, R.: Fppw: A fair and privacy preserving watchtower for bitcoin. Cryptology ePrint Archive, Report 2021/117 (2021), https://eprint.iacr.org/2021/117

10. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2016)
11. Rahimpour, S., Khabbazian, M.: Hashcashed reputation with application in designing watchtowers. In: 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). pp. 1–9. IEEE (2021)
12. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of cryptology **4**(3), 161–174 (1991)