Maravedí: A Secure and Practical Protocol to Trade Risk for Instantaneous Finality

Mario Larangeira [1,2] and Maxim Jourenko[1]

[1] Department of Mathematical and Computing Science,
School of Computing,
Tokyo Institute of Technology.
Tokyo-to Meguro-ku Ookayama 2-12-1 W8-55, Japan.
{mario@c, jourenko.m.ab@m}.titech.ac.jp
[2] Input Output Global, Singapore.
mario.larangeira@iohk.io
http://iohk.io

Abstract. The efficiency of blockchain systems is often compared to popular credit card networks with respect to the transactions per second rate. This seems to be an unfair comparison since these networks do not complete a transaction from beginning to end. Rather they buy the risk and settle it much later. Typically transactions have only two players, the *payer* and the *payee*, and the settlement of this transaction requires time since it depends on basic properties of the consensus protocol. In practice, the payee, very often, needs to wait for confirmation in order to ship the traded goods. Alternatively, the payee, or merchant, can ship it in faith that the transaction will be confirmed. Our contribution, the Maravedí Protocol, introduces a third player to minimize the risk of the payee to be left without the payment even without the consensus layer confirmation. The main idea is that the third player can work similarly to a *credit card* company. That is, it buys the risk from the merchant, by a small discount, and allows the third player to pay it instantaneously via a payment-channel like protocol. In parallel, the third player receives the regular payment transaction from the payer that can be settled on the chain, thus, after waiting the consensus/blockchain required time. Moreover, the on-chain transaction pays the full amount, allowing the third player to cash in the discount. Hence, on the side of the merchant, our protocol puts forth *instantaneous finality* in a novel way to the best of our knowledge.

1 Introduction

The most widely known application for blockchain based systems is cryptocurrency. One of the main bottlenecks for mass adoption boils down to the number of transactions per second (TPS) rate such systems can process. New transactions are confirmed by the system as new blocks are added to the blockchain, and the same chain is agreed among all the honest nodes of the network via a Nakamoto style consensus protocol. The design and security of the consensus protocol plays a major role in the transaction validation process because it dictates the rate of block creation either in Proof of Stake (PoS) or Proof of Work (PoW) systems. Different protocols implement different refresh rates. Therefore, in order to confirm a transaction, *i.e.*, beyond the non-negligible probability of disappearing from the blockchain, it is necessary to wait a few blocks. At the end of this confirmation period, the systems offer *finality*, *i.e.*, informally, a period of time that a user needs to wait in order to be sure the transaction cannot be reversed. The time requirement for such desirable property varies depending on the system, for example, Bitcoin [20] needs on average every 60 minutes (or 6 blocks), Ethereum [27] requires 2,5 min (or 10 blocks), Cardano/Ouroboros theoretically requires one time-slot which is 20 seconds (however for *true immutability of the chain* according to the newest setting can be 36 hours) [4,8]. In general, pure blockchain based consensus protocols are slower than BFT systems, *e.g.*, PBFT [3]. Therefore, in order to circumvent this crucial limitation in distributed systems, hybrid protocols, *i.e.*, combination of blockchain and BFT approaches, were suggested, *e.g.*, Algorand [9] and Thunder [21]. However the proposed protocols still require some waiting for the confirmation time in order to achieve finality.

Concretely, the confirmation time for the transferred funds can be particularly risky to settle a transaction. Namely, after the Customer performs a payment, *i.e.*, the payer, how to assure, quickly, to the Merchant, *i.e.*, the payee, that it is safe to deliver the purchased goods. In practice, within a distributed environment with confirmation time, the Merchant can deliver the goods in the optimistic case, *i.e.*, on faith that the transaction will be finalized, after the issue of the payment. Technically, in this scenario, the Merchant is under the risk that the transaction will not be finalized given the waiting time of the consensus protocol. In other words, the Merchant is left without any guarantees of receiving the payment. Researchers and practitioners have allocated effort, which has not yet produced a clear definitive solution to the described limitation. In particular, businesses that rely on cryptocurrency are keen to know whether transactions can be securely confirmed much faster.

An earlier attempt to improve the TPS ratio is based on protocols that do not perform most of its transactions on the chain, *i.e.*, an off-chain protocol. The most widely used is the Lightning Protocol [24] which establishes a channel between two parties and limits the interaction with the blockchain. While the work in [24] does offer a way to issue instantaneous, and verifiable, transactions between the two parties, the initial establishment of the channel requires the interaction with the blockchain, and, therefore, it is necessary to wait for the confirmation time. Thus it is not suitable to the case of several parties, as every new payment should trigger the creation of a new channel.

An Unfair Comparison. It is well known that centralized payment methods like debit cards, Paypal or Payment Hubs [11] are orders of magnitude faster than blockchain based systems, and they do offer instantaneous (with the disclaimer we do not consider card information checking, receipt issuing, etc, which in fact make it not strictly "instantaneous"). However, when discussing the transaction throughput of cryptocurrencies, a very common comparison often appears between the decentralized ledger technology and credit card networks. Whereas the former most known example, *i.e.*, Bitcoin, carries 7 TPS [25], the latter handles tens of thousands per second [25]. This alarming difference can be explained by the centralized nature of the network of the credit card company. Since decentralized ledger technology is mostly *decentralized*, it imposes significant overhead in the processing speed. To the best of our knowledge, another often overlooked difference is that credit card companies do not *only* intermediate the transaction between Customer and Merchant, they, in fact, buy the risk, *i.e.*, the credit risk, from the Merchant by a profit. In the light of this description, this *credit card model* does not seem to have a counterpart in the cryptocurrency realm. More concretely, in the physical world, a Customer would ask for the credit card company, which we denote Risk Buyer, to pay the Merchant, while in a future date the Customer pays the Risk Buyer back.

In this setting, the Merchant is assured, given the prior deal with the Risk Buyer, that it will receive the funds even if the Customer does not fulfill its part of the deal. In the real world, the Risk Buyer would require extra information from the Customer to build confidence and, therefore, *buy the risk*. Concrete examples of extra information are typically background check, spending history, reputation, etc. Moreover Risk Buyer would also require some discount from the Merchant. By paying the Merchant in advance, the Risk Buyer receives a profit in order to engage in the deal in the first place. The crucial term is "in advance". That is, how quick, and guaranteed, this payment should be performed in order to satisfy the *finality* property in the eyes of the Merchant.

Instantaneous Payment over Blockchain. Not all types of payments require confirmation time in the blockchain realm. In fact, several protocols put forth instantaneous confirmation payments via payment/state channels [6,13], *i.e.*, depend virtually on the network speed, and the already mentioned [24]. Typically, these protocols need an initial phase which indeed requires confirmation time. Subsequent payments, under certain values, are *instantaneous* between payer and payee, albeit they are not reflected in the consensus protocol immediately. The result of these payments are reflected in the underlying ledger when the channel is closed which requires, again, the confirmation time in order to persist the final state of the channel.

Our approach takes this natural property of payment channels, *i.e.*, instantaneous payments, and combines it with blockchain aided transaction, in order to provide instantaneous settlement, *i.e.*, *finality*, to the receiver of the funds, *i.e.*, the Merchant, in an ad-hoc transaction, *i.e.*, without prior interaction to the Costumer. Our approach is in sharp contrast to the existing protocols as we put forth a novel approach: a hybrid design for protocols, and a relaxed notion of finality, *i.e.*, only applies for the Merchant.

Related Work. As already outlined, early attempts to increase the TPS rate of blockchain based systems rely on the establishment of *payment channels.* Such protocols, in general, work by locking funds from two participants, *i.e.*, the capacity of the channel, by standard transactions registered in the ledger, after the confirmation of such a transactions, they interact directly, *i.e.*, without the ledger. Such an approach, effectively allows these participants to carry out numerous direct transactions without time limitations from the consensus protocol. Such transactions are settled instantaneously, although they are not registered in the blockchain/ledger until the channel is closed. It is important to notice that, although the transaction is not present in the ledger, by the time of its settlement, the participants cannot revert it due to mechanisms of the typical payment channel protocol, *e.g.*, [24]. Such a class of protocols, named layer-2 (layer-1 being the consensus layer), has already a large body of work [12].

The protocol in [24] had received a formal treatment by Kiayias *et al.* [16], which also showed evidence that common ledger functionalities (in the UC sense) introduced in the literature do not seem to be realizable (for instantaneous finality) under realistic network assumptions. In particular, they remark that [6,7,17] presents a model that settles every submitted transaction immediately which does not seem to be realistic for existing network models. A significant reminder of the need for protocols that offer quick finality which motivates our novel approach and suggested protocol. A later work by Dinsdale-Young*et al.* [5] tackles the slow consensus drawback by introducing a concrete finality layer. The work relies on the formal properties of blockchain, *i.e.*, common-prefix, chain-growth and chain-quality. Despite its ingenuity, it cannot offer quick finality.

On a similar topic, Miller *et al.* [18] introduced Sprites, a variant of payment-channels, which reduces the worst-case "collateral cost" that each hop in the payment network may incur. In a nutshell, this work leverages on the use of smart contracts, hence their use is restricted to "state channels", to reduce the attack surface and speed up the closing of the channel. More recently, Jourenko *et al.* [14] improved on the technique to close multi-hop payment channels, but without the use of smart-contracts, making it compatible with systems without smart contract capabilities, such as the Bitcoin. In comparison, although our work has three participants, we do not consider a multi-hop payment channel. In fact, our setting is simpler, given that only the Merchant and the Risk Buyer maintain the channel, making it an "almost" regular channel in the sense of [24]. The only difference is the change we introduce to the construction of the channel, which we describe next. This change virtually makes no significant difference in the efficiency of concrete implementation of [24], but allows us to introduce the third player as our model requires.

Our Contribution. This work introduces the study of finality via risk trade. To the best of our knowledge, it is the first time this approach is suggested. We discuss the desiderata for such an approach. Furthermore, we present the Maravedí Protocol which takes this fundamentally different approach to implement *finality.* Namely, it puts forth *instantaneous finality* to the receiver of the funds (which in our jargon is the "Merchant"). Our design sidesteps the drawback of distributed systems, *i.e.*, the confirmation time of the blockchain systems, by allowing a "pre-processing" phase which is mainly the establishment of a pair-wise payment channel. The channel bridges the Merchant and the participant who pays in advance, therefore buying the risk, (which in our model is the "Risk Buyer"). Hence the payment to the Merchant is timely performed via the already established payment channel, taking advantage of the existing property of the cited protocol [16,24], *i.e.*, instantaneous payment. Note that the channel creation requires the confirmation time from the ledger. However, made in advance, the required time does not interfere in the instantaneous interaction between the Risk Buyer and the Merchant.

Our design allows any Customer to approach the Merchant in an ad hoc manner, *i.e.*, without any previous interaction, a desirable property. The Customer issues a regular transaction to the Merchant, *i.e.*, "regular" means it requires time to settle in the ledger, however it is assigned to the Risk Buyer. The Merchant forwards the transaction to the Risk Buyer conditioned with a payment under its jointly created payment channel. A technical issue we solve in our construction is conditionally fixing the regular payment and the payment channel transferring of funds, without requiring trust between the two participants, *i.e.*, Risk Buyer and Merchant. We adapted the Hash Time Lock Contract (HTLC) technique used in the Lightning Network [24,16] by allowing the Risk Buyer to keep the value x while delivering $y = H(x)$ to the Merchant (which forwards it to the Costumer), for a hash function H. The transaction issued by the Customer is conditioned to the disclosing of x, which happens when the final payment is done via the payment channel or the ledger.

More concretely, in our construction the HTLC technique is leveraged to provide *atomicity* between the transaction to be published in the ledger and the one performed in the payment-channel. Anytime the Risk Buyer publishes the received blockchain transaction, it also discloses x which allows the Merchant to also cash in the funds in the channel. We remark that the instantaneous finality provided by our protocol relies on the combination of the ledger transaction and the payment-channel transferring of funds. Despite the funds being locked in the channel, the Merchant is assured to receive it given that the "pre-processing" phase within the channel was established. The guarantee provided by the channel construction is more desirable than solely relying on the optimistic heuristic that nodes will reach an eventual consensus in the blockchain based system. Note, for example, that a naive construction would be requiring Risk Buyer to pay the Merchant right away, via the payment channel, and just wait for the transaction (from the Customer) to be confirmed. However, in this case, there is no guarantee that the Merchant would cooperate by, for example, publishing the transaction or sending it to the Risk Buyer. it may be the case, that the Risk Buyer, relying on some reputation registry, may no longer want to do business with the Merchant, however, our construction does not need to rely on such strategy. It enforces payment by design.

In summary our contributions are

- We start by a discussion of the desiderata for risk trade in order to obtain finality for the point of view of the Merchant. A notion we introduce in this work in the best of our knowledge;
- We propose the Maravedí Protocol which concretely illustrates this novel approach to put forth instantaneous finality for the Merchant;
- Finally, we describe the main property of our protocol, namely, the Merchant and the Risk Buyer redeem the respective payment securely.

We remark our protocol is quite practical since we adapted an already existing construction, the Lighting Protocol [16,24]. We also show chase that our approach does not require any sort of collateral, smart-contract capability or checkpoints, *e.g.*, [15]. Furthermore, we believe that the adaptation we propose for the HTLC can be easily implemented taking the original source [24] code as a blueprint for the construction.

Roadmap. One of our main contributions is presented in Section 3, *i.e.*, the desiderata of a three-participant trade risk. Section 4 outlines the main phases of our construction, and provides an intuition for the approach, while Section 5.2 describes the concrete construction. We provide the security analysis in Section 6 along with a discussion with respect to the early introduced desiderata. The last section, *i.e.*, Section 7, presents our final comments.

2 Preliminaries

Our protocol relies on the payment-channel protocol for the risk trade for the payment to the Risk Buyer. It is convenient to review the UTXO Model and the Lighting Network design. In addition, for completeness, it is necessary to recall the digital signature primitives used in our construction. However we first start by the primitive which allows us to derive new keys. From now, let λ be the security parameter, while $negl(\lambda)$ is the standard negligible function and $x \overset{r}{\leftarrow} \mathcal{X}$ is the sampling algorithm for x from a uniform distribution $\mathcal{X}$.

2.1 Pseudorandom Function

Each payment performed within the channel, a new set of keys are generated from the original set of master keys. From now, we present a formal definition for the Pseudorandom Function (PRF) and its security notion.

Definition 1 (PRF). *Let $\lambda \in \mathbb{N}$, an index, which defines the family, $s \in \{0,1\}^*$,$p : \mathbb{N} \to \mathbb{N}$ and f_s be a family of functions from $\{0,1\}^{p(|s|)}$ to $\{0,1\}^{p(|s|)}$. Furthermore, let Func_λ be the uniform distribution over the set of all $\{0,1\}^\lambda \to \{0,1\}^\lambda$ functions. We say that f_s is a pseudorandom function family if:*

- *$\forall s \in \{0,1\}^*, \forall x \in \{0,1\}^{p(|s|)}$, $\exists$ PPT algorithm, denoted* $\mathrm{PRF}(x, s)$*, that computes $f_s(x)$ for the input x*

– $\forall\lambda \in \mathbb{N}$, $\forall$ *PPT* $\mathcal{A}$,

$$\left| \Pr_{\substack{s \xleftarrow{r} \{0,1\}^\lambda \\ \mathcal{A}\text{'s coins}}} [\mathcal{A}^{f_s(\cdot)}(1^\lambda) = 1] - \Pr_{\substack{\mathrm{PRF} \xleftarrow{r} \mathsf{Func}_\lambda \\ \mathcal{A}\text{'s coins}}} [\mathcal{A}^{\mathrm{PRF}(\cdot)}(1^\lambda) = 1] \right| = negl(\lambda),$$

where $\mathcal{A}$ is given oracle access to $f_s(\cdot)$ and PRF *in each of the probability expressions above respectively.*

2.2 Digital Signature Schemes

Typically, the standard digital signature scheme is the main cryptographic primitive for transaction frameworks. Furthermore, payment channel [24,16] relies on two extra signature schemes, namely, Identity Based Signature and Combined Signature. It is convenient to review the three schemes.

Definition 2 (The Digital Signature Scheme [10]). *It is the triple of algorithms $\mathcal{S} = \langle \textsc{Gen}, \textsc{Ver}\mathrm{DS}, \textsc{Sign}\mathrm{DS} \rangle$, such that*

– $\textsc{Gen}(1^\lambda) \to$*: The generation procedure, given the security parameter λ, and outputs the key pair, i.e., verification and secret keys, respectively* vk *and* sk*;*
– $\textsc{Sign}\mathrm{DS}(\mathsf{sk}, m) \to \sigma$*: The sign procedure takes as input the secret key* sk *and the message m to output the signature σ;*
– $\textsc{Ver}\mathrm{DS}(\mathsf{vk}, m, \sigma) \to \{0,1\}$*: The verification procedure takes the verification key* vk*, the message* m *and the signature σ to output* 1 *if the signature is valid, or* 0 *otherwise.*

Moreover, the scheme is said to be resistant to Existential Unforgeable under Adaptive Chosen Message Attacks (EUF-CMA) if, with respect to the security parameter λ, for any PPT algorithm $\mathcal{A}_{forger}$, which can query the signature oracle $\textsc{Sign}\mathrm{DS}(\mathsf{sk}, \cdot)$ *for signatures on a polynomial number of messages m_i, it holds* $\Pr[(\mathsf{vk}, \mathsf{sk}) \leftarrow \textsc{Gen}(1^\lambda) : (m, \sigma) \leftarrow \mathcal{A}_{forger}^{\textsc{Sign}\mathrm{DS}(\mathsf{sk},\cdot)} \wedge m \neq m_i] < negl(\lambda)$*, where all the probabilities are computed over the random coins of the adversary and the signature algorithms. Furthermore,* $\Pr[(\mathsf{vk}, \mathsf{sk}) \leftarrow \textsc{Gen}(1^\lambda) : \textsc{Ver}\mathrm{DS}(\textsc{Sign}\mathrm{DS}(\mathsf{sk}, m), \mathsf{vk}, m) = 1] = 1$ *for every m.*

Definition 3 (Identity Based Signature (IBS) [26,22]). *The IBS scheme is a 5-algorithm tuple* $\mathrm{IBS} = \langle \textsc{Gen}, \textsc{KeyDer}, \textsc{PubKeyDer}, \textsc{Sign}\mathrm{IBS}, \textsc{Verify}\mathrm{IBS} \rangle$*, which is an augmented version used in [16] of the originals proposed in [10]. Each algorithm is used as follows:*

– $\textsc{Gen}(1^\lambda) \to (\mathsf{mvk}, \mathsf{msk})$*: The setup algorithm takes the security parameter and outputs the master key pair;*
– $\textsc{KeyDer}(\mathsf{mvk}, \mathsf{msk}, \ell) \to (\mathsf{vk}_\ell, \mathsf{sk}_\ell)$*: The regular signing key pair is derived with respect to the label ℓ;*
– $\textsc{PubKeyDer}(\mathsf{mvk}_\ell) \to \mathsf{vk}_\ell$*: It is possible derive only the verification key* $\textsc{PubKeyDer}$ *with only the label ℓ and the master verification key* mvk*;*
– $\textsc{Sign}\mathrm{IBS}(\mathsf{m}, \mathsf{sk}_\ell) \to \sigma$*: This is the regular signing algorithm. That is, it requires only the message* m *and the secret key* sk*;*
– $\textsc{Verify}\mathrm{IBS}(\sigma, \mathsf{m}, \mathsf{vk}_\ell) \to \{0,1\}$*: The verification* [3] *is performed, as usual, using the* vk, m *and σ.*

[3] Note that here, despite of being an Identity Based Signature, it requires vk_ℓ as input as it is defined in [16][Section 5].

Given the space constraints and the loose connection with our proposed construction, we skip a full description of the security properties of the scheme. We refer the reader [16][Section 5] for a full description of them.

Definition 4 (Combined Signature (CS) [16]). *The CS scheme is a 7-algorithm tuple* $\text{CS} = \langle \textsc{MasterKeyGen}, \textsc{KeyShareGen}, \textsc{CombinePubKey}, \textsc{CombineKey}, \textsc{TestKey}, \textsc{SignCS}, \textsc{VerifyCS} \rangle$ *which was introduced informally in [24] and formally analized [16]. Each algorithm is used as follows:*

- $\textsc{MasterKeyGen}(1^\lambda) \rightarrow (\mathsf{mvk}, \mathsf{msk})$*: The key generation algorithm takes the security parameter and outputs the master key pair;*
- $\textsc{KeyShareGen}(1^\lambda) \rightarrow (\mathsf{vk}, \mathsf{sk})$*: The key generation algorithm takes the security parameter and outputs a regular key pair;*
- $\textsc{CombinePubKey}(\mathsf{mvk}, \mathsf{vk}) \rightarrow \mathsf{cvk}$*: The combination algorithm combines both master and regular verification keys into a new verification key;*
- $\textsc{CombineKey}(\mathsf{mvk}, \mathsf{msk}, \mathsf{vk}, \mathsf{sk}) \rightarrow (\mathsf{cvk}, \mathsf{csk})$*: The combination algorithm receives both master and regular key pairs into a new combined key pair;*
- $\textsc{TestKey}(\mathsf{vk}, \mathsf{sk}) \rightarrow \{0, 1\}$*: The test algorithm verifies if both keys are a regular key pair;*
- $\textsc{SignCS}(\mathsf{m}, \mathsf{csk}) \rightarrow \sigma$*: It works as a regular signature generation algorithm which receives as input the message and the combined verification key;*
- $\textsc{VerifyCS}(\sigma, \mathsf{m}, \mathsf{cvk}) \rightarrow \{0, 1\}$*: The algorithm works as a regular verification signature algorithm which receives signature, message and the combined verification key.*

For similar reasons we refrain to describe the security properties of Definition 3, we also skip the the properties of Definition 4. Again, we refer the interested reader to [16][Section 9] for a full description of them. Later in Section 2.4, instances of these signature schemes are used in the payment-channel state and payment. Each participant of our protocol keeps one instance of the regular digital signature and three of the IBS (for payment, delay payment and HTLC) and one of the CS (for revocation of the old payment channel state).

2.3 The UTXO Model

Transactions in the Unspent Transactions Output (UTXO) Model, popularized by the Bitcoin Whitepaper [20], are composed by two parts: the set of *Inputs* $\mathcal{I}$, and the (ordered) list of *outputs* $\mathcal{O}$. A single output $o \in \mathcal{O}$ contains (1) the value of the output and (2) the condition to redeem that value, *e.g.*, $[k : \mathsf{v}, \mathsf{vk}]$ for which k is the index number of the output the value v, and the witness for public key vk, meaning it requires the signature σ corresponding to vk. The condition could also include time constraints with respect the height of the latest block in the blockchain or the preimage of a hash value as in the HTLC. Furthermore, an input references existing output by pointing its index number in a transaction, that is $\mathtt{Tx}.\mathcal{O}[1]$ refers to the first output of transaction $\mathtt{Tx}$. This mechanism of numbering the outputs is useful because $\mathcal{I}$ contains references of individual outputs in previous (settled) transactions. Settled transactions can be straightforwardly found in the distributed ledger, along with the witness (typically, a signature issued by the correct secret key) that allows a party to redeem the value of the original input.

Transaction Notation Assume a participant A has the pair of public and secret key $(\mathsf{vk}, \mathsf{sk})$, if the pair controls a set of outputs $(o_1, o_2, o_3, \dots)$, respectively in $\mathtt{Tx}_1$, $\mathtt{Tx}_2$ and $\mathtt{Tx}_3$, then it is said that the state of A contains the mentioned outputs, *i.e.*, $\Sigma_A = (o_1, o_2, o_3, \dots)$. Thus, for example, a transaction $\mathtt{Tx}_{transfer}$ redeeming o_2, therefore transferring $\mathsf{v}/2$ to participant B and $\mathsf{v}/2$ to C is given by $\mathtt{Tx}_{transfer} \leftarrow \mathtt{Tx}\{\mathcal{I} : (\mathtt{Tx}_2.o_2); \mathcal{O} : [1 : \mathsf{v}/2, \mathsf{vk}_B], [2 : \mathsf{v}/2, \mathsf{vk}_C]\}$ and it requires, in order to be redeemed, the signature σ_A, as it contains $\mathtt{Tx}_2.o_2$ in its input set $\mathcal{I}$. Figure 1 illustrates this example.

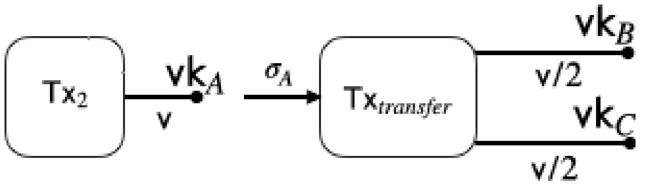


Fig. 1. In order to publish $\mathtt{Tx}_{transfer}$ and therefore redeem the output from $\mathtt{Tx}_2$ and transfer values to B and C, A needs to provide a signature σ_A. In the multisig setting, it may also be possible that two signatures are necessary to redeem the output. Lightning Networks (later) relies on such a setting.

2.4 The Lightning Network

The Lightning Network (LN) [24] takes full advantage of the above model to move transactions outside of the ledger into the so-called Layer-2. Moreover LN was thoroughly described in [16], which shed light on some of the inner workings of the deployed protocol. In particular, it investigated the interplay between the different signature schemes and the HTLC hash value when funds are exchanged between more than two participants, *i.e.*, a payment network. Briefly, in order to illustrate it, consider the existing channel between two players A and B. Without loss of generality, assume that only A has transferred some funds c_A to the channel AB. Assume also that another channel, say BC, exists between B and C, and similarly only B has funded it with c_B.

The main goal of the construction is to offer a way to transfer values from A to C without the creation of the channel AC. For that purpose, C picks a random value x and sends its hash value $y = H(x)$ directly to A. The protocol initiates by A triggering B in order to send the amount v to C, and the overall dynamics is inverted. That is, B, after being triggered by A, pays C the same amount to be transferred, *i.e.*, v, and receives x in exchange. At this point, B is in deficit, because it did not received v yet. Then B exchanges x for v with A which can verify the equality $y = H(x)$.

Remark 1. In comparison to [24,16] our design for the channel has also two players: the Merchant and the Risk Buyer (as in [24,16]), however its use of the HTLC differs. The latter picks the HTLC pre-image x, keeps it secret while disclosing $y = H(x)$ to the Merchant. This design contrasts with [24,16] because in our case the provider of y, *i.e.*, Risk Buyer, is also the one receiving the payment. Later, in our construction, it is shown that the HTLC value x is used to both the delayed release of the payment, when Risk Buyer discloses x to the Merchant, and receiving the funds via the blockchain when publishing the transaction in the ledger.

Initial Setting. The above outlined protocol is a simplification, given that the actual implementation of a payment channel protocol is involved and requires a careful change of states between all the participants. In fact, the creation procedure of each pairwise channel, *e.g.*, between A and B, requires each participant to issue (1) a single *Funding* Transaction $\mathtt{Tx}_F$ and (2) a pair of *Commit* Transactions $\mathtt{Tx}_{commit}$; one denoted *local* and the other *remote*. They differ for the signature used, A's or B's, to redeem the funding transactions.

Each participant keeps its key pairs: the regular signature key pair $(\mathsf{vk}, \mathsf{sk})$, the IBS key pairs $(\mathsf{vk}_{dpay}, \mathsf{sk}_{dpay})$, $(\mathsf{vk}_{pay}, \mathsf{sk}_{pay})$, and $(\mathsf{vk}_{htlc}, \mathsf{sk}_{htlc})$ for the HTLC, and the CS $(\mathsf{vk}_{rev}, \mathsf{sk}_{rev})$ as outlined in Section 2.2. It also keeps track of the counterpart's IBS verification keys vk_{dpay}, vk_{pay} and vk_{htlc}. Series of payments are performed as series of Commit Transaction exchanges between the two participants (with the first Commit Transaction pair being $n = 1$ for a n-long sequence). On every new payment, a new state of the channel is created. It is performed by generating a pair of new transactions, and their signatures, along with freshly derived key pairs are generated, and old committed transactions are revoked (by sharing the revocation keys with the counterpart). For a full specification of the mechanism we refer the reader to [16]. For now, it is enough to look into the two types of transaction pairs. They are

- Funding: The transaction $\mathtt{Tx}_F$ contains an output $o = [1 : c_A, \mathsf{vk}_A]$, therefore $\mathtt{Tx}_F.\mathcal{O}[1] = (c_A, \mathsf{vk}_A \wedge \mathsf{vk}_B)$, and it will be available on the ledger after the next transactions are signed;
- Commit: The first Commit Transaction redeems their outputs (if published), requiring only the signatures on their inputs as witness. However both transactions are kept locally when the channel is active. Concretely, A keeps the following transactions (analogously for B),

$$\mathtt{Tx}^A_{local} \leftarrow \mathtt{Tx}\{\mathcal{I} : (\mathtt{Tx}_{F_A}.\mathcal{O}[1]); \mathcal{O} : [1 : c_A, \mathsf{vk}^B_{rev,n=1} \vee (\mathsf{vk}^A_{dpay,n=1}, \varDelta_B)], \\ [2 : c_B, \mathsf{vk}^B_{pay,n=1}]\};\ \textit{and}$$

$$\mathtt{Tx}^A_{remote} \leftarrow \mathtt{Tx}\{\mathcal{I} : (\mathtt{Tx}_{F_B}.\mathcal{O}[1]); \mathcal{O} : [1 : c_A, \mathsf{vk}^A_{pay,n=1}], \\ [2 : c_B, \mathsf{vk}^A_{rev,n=1} \vee (\mathsf{vk}^B_{dpay,n=1}, \varDelta_A)]\},$$

along with signatures σ_A and σ_B with respect to vk_A and vk_B respectively. Furthermore they also depend on the IBS keys, where $\mathsf{vk}^B_{rev,n=1}$, $\mathsf{vk}^B_{pay,n=1}$, $\mathsf{vk}^B_{dpay,n=1}$ are the counterpart's verification keys, and $\mathsf{vk}^A_{pay,n=1}$, $\mathsf{vk}^A_{rev,n=1}$, $\mathsf{vk}^A_{dpay,n=1}$ are the participant's own verification keys, along with the agreed delay $\varDelta_A$ and $\varDelta_B$ which are arbitrarily chosen by each participant and exchanged prior to the creation of the Funding transactions.

Payment and State Revocation. Payments are performed by altering the balance, *i.e.*, c_A and c_B, via the exchange of new pairs of transactions from the initial state, *i.e.*, $\texttt{Tx}_{remote,1}$ and $\texttt{Tx}_{local,1}$. The initial state $n = 1$ progresses to the new state $n = 2$, *i.e.*, $\texttt{Tx}_{remote,2}$ and $\texttt{Tx}_{local,2}$ via exchange of signatures and new derived keys. Crucially, older states need to be "revoked" which is performed by the computation of the secret revocation key $\mathsf{sh}_{rev,n=1}$ of the old state. The main idea is that a participant is not financially incentivized to publish a transaction from an old state, because the counterpart can use its newly computed revocation key to redeem the funds. Briefly, if a participant publishes an old state, the counterpart can take all the funds of the channel. It is important to remark that the secret revocation key is only computed when the state is agreed to be "old", which prevents any party to misuse it.

During the HTLC, there is an intermediate state create by a pair of transaction $tx_{A,htlc}$ and $tx_{B,htlc}$, under locally derived HTLC keys $(\mathsf{vh}_{htlc}, \mathsf{vt}_{htlc})$, which redeems the outputs of the state transactions $\texttt{Tx}_{remote,n}$ and $\texttt{Tx}_{local,n}$ for the state n. A thorough discussion of revocation and the HTLC mechanism can be found in [16].

Time Analysis. The $n-$hop payment channel/network requires a careful setting between the (relative) time delay parameters $\Delta_1, \ldots, \Delta_{n-1}$, for the dispute of the payments, within each pairwise channel, in addition to the confirmation time of the consensus protocol, *i.e.*, $\Delta_{confirmation}$. This parameter is a theoretic value only, however, the security of deployed systems relies on timing values to be larger than $\Delta_{confirmation}$ such as the dispute time delay parameter Δ_{HTLC} when performing the HTLC procedure as well as Δ_A and Δ_B. While $\Delta_{confirmation}$ is unknown, values for Δ_{HTLC}, Δ_A and Δ_B are set by the implementation [19,23] in an attempt to capture the real delay of the protocol. The parameter Δ_{HTLC} can vary from 14 blocks [23] to a value between 14 and 144 blocks [19] whereas parameters Δ_A and Δ_B are set between 6 and 17 blocks[4]. In Maravedí, we remark that the dispute time delays are arbitrarily picked by each participant of the channel, say Δ_A and Δ_B, as it is defined in [16], and there is also the dispute time delay parameter Δ_{HTLC} when performing the HTLC procedure. In our construction we require that Δ_A, Δ_B, Δ_{HTLC} are greater than $\Delta_{confirmation}$, which, likewise [19,23], can be set by the concrete implementation. In Section 5.2, we only rely on Δ_A and Δ_B, for the Merchant and Risk Buyer, respectively, and Δ_{HTLC} for the intermediate state given it is a pairwise payment channel.

3 Desiderata

We overview the main requirements/characteristics, *i.e.*, desiderata for a protocol to trade risk, for each of the three classes of participants, namely *Customer*, *Merchant* and *Risk-Buyer*. We denote them respectively by C, M and R.

Let us start with the requirements/characteristics from C's point of view:

[4] https://github.com/lightningnetwork/lnd/blob/master/lnrpc/lightning.proto.

- **Instantaneous Finality:** The transaction performed between C and M should be instantaneous. That is, no confirmation time is required. Similarly to a physical transaction with physical goods;
- **Ad Hoc Availability:** The participant C should not be required to know M in advance. In the sense that any sort of interaction or preparation must not be required between M and C;
- **Background Availability:** The participant C may know R, and may interact with it via a direct communication channel prior to be in direct contact with M (it is the same for R later);
- **Trustless Trade:** There is no trust assumed between C and M.

From now we list the requirements/characteristics for M:

- **Instantaneous Settlement:** The merchant M should be in contact with R which allow instantaneous payment which can be created prior to any transaction between C and M;
- **Direct Interaction:** All transactions are exchanged via the direct (not necessarily secure) communication channel between C and M, and M and R;
- **Risk Trade Security:** For each accepted transaction, with an arbitrary value, M receives from R the same value minus a discount, via the already established payment channel (analogous [5] for R);
- **Trustless Risk Buyer:** There is no trust assumed between M and R (the same in the point of view of R).

Lastly we list the requirements/characteristics for R:

- **Allowed Pre-processing:** Prior to any performed transaction between C and M, a "pre-processing" phase is *allowed* (not necessarily related to a payment channel) to implement a instantaneous payment method between M and R with enough payment capacity;
- **Background Availability:** The participant R can potentially keep a list of (ledger) addresses for multiple customers C_i (for accessing the risk of buying C's risk, for example, and it is identical to C's case);
- **Risk Trade Security:** For every transaction accepted by M, R should receive the equal value (after the confirmation time) by publishing a regular transaction (*e.g.*, issued by C) in the ledger (analogous [6] for M).
- **Trustless Merchant:** There is no trust assumed between R and M (the same in the point of view of M).

We remark that C and R do not necessarily interact directly, unless they prior to the execution of the protocol. For example, in order to share "background" as in the *Background Availability*. Therefore, we let it out of the desiderata. However, in the general case they do not necessarily trust each other.

For clarification, our protocol handles two types of transactions which we denote by *payment channel transaction* and *regular transaction*. Whereas the

[5] That is, the procedure of R and M are comparable in certain respects. In other words, they are not quite the same, because R does not receive anything. It, instead, pays the same value minus a discount to M, being its counterpart in the (single-hop) payment channel.

[6] In the sense of what already explained earlier.

former is the transaction between M and R over the payment channel, the later is the one that relies on the blockchain confirmation time, and issued by C to be redeemed by R. Technically, there are only small differences between the two types which become clear later, however we highlight this terminology to avoid ambiguities and improve readability.

4 Outline of the Construction

The main phases of our protocol are *Protocol Set up*, *Issuing Transaction Proposal*, *Risk Buying*, and *Settlement* performed by M, C and R all with access to their respective states Σ_i for $i \in \{\mathsf{M}, \mathsf{C}, \mathsf{R}\}$ via the ledger. Furthermore, we assume they have a pairwise direct communication channel. Their interaction happens through direct exchange of transactions: (1) $\mathtt{Tx}_{\mathsf{C}}$ a regular transaction to be redeemed by R via ledger, and (2) the payment channel transaction $\mathtt{Tx}_{\mathsf{R}}$ to be issued by R and assigned to M. Both transactions require the reveal of the HTLC preimage x picked by R. In addition to the value x, the payment channel transaction $\mathtt{Tx}_{\mathsf{R}}$ requires signatures from R and M as it is standard in [24].

Our model starts with the instantiation of the payment channel in the beginning of a trading day as a “pre-processing” phase.

Protocol Set Up. Initially, each participant generates locally the verification and secret key pair $(\mathsf{vk}_i, \mathsf{sk}_i)$ for $i \in \{\mathsf{M}, \mathsf{C}, \mathsf{R}\}$. On the beginning of each trading day, R initiates a payment channel with M. Note that before accepting any transactions (from C), M and R are required to wait until the confirmation of the channel initialization as given by the consensus layer. This waiting time assures both players that the channel can be safely used.

Transaction Proposal. After the earlier described initial setting, assume a merchant M receives a transaction proposal $\mathsf{Tp} = (\mathsf{v}, \mathsf{vk}_{\mathsf{C}})$ from C, for a value v. The Merchant M directly contacts R through a separated (and independent) communication channel, and offers Tp for trade.

Risk Buying. In the case R accepts Tp, then it hands back $(\mathsf{d}, \mathsf{vk}_{\mathsf{R}}, y)$ to M where y is the image for the HTLC, d is the discount of the value v, vk_{R} is its own public key where the payment should be directed, *i.e.*, regular transaction. At this point, M (1) signals C that the deal can continue by handing y to C, and (2) requests R the payment, which will be done via the payment channel. It follows that C generates $(\mathtt{Tx}_{\mathsf{C}}, \sigma_{\mathsf{C}})$, where σ_{C} is the signature of $\mathtt{Tx}_{\mathsf{C}}$ with respect to vk_{C}, and returns $(\mathtt{Tx}_{\mathsf{C}}, \sigma_{\mathsf{C}})$ to M, while R and M perform the payment in the channel, *i.e.*, progress the channel from, say, state n to $n + 1$. At this point the intermediate state, for the HTLC, is created and both M and R revoke the previous ones.

Settlement. At this point, M has $(\mathtt{Tx}_{\mathsf{C}}, \sigma_{\mathsf{C}})$ and submits it to R which receives in exchange of the HTLC pre-image x such that $y = h(x)$. Since M and R have both signatures for $\mathtt{Tx}_{\mathsf{R}}$ and the value x for the HTLC hash value, the payment over the channel is performed for the value of $\mathsf{v} - \mathsf{d}$ to M, and R has $(\mathtt{Tx}_{\mathsf{C}}, \sigma_{\mathsf{C}})$ which can be redeemed with its publication in the ledger. At this point, M already assured its payment in the channel, whereas R has to wait the confirmation time to settle its payment via the consensus protocol. As far as M is concerned the procedure is over therefore it can initiate another transaction with another customer any time. The properties of the channel assure M that its payment is guaranteed and available by the time of the closure of the channel. By the end of the trading day, both R and M jointly decide to close the channel, which frees the payments for M. This procedure is a regular one from any payment channel.

Remark 2. The overall actions rely on the procedures of a payment channel protocol like [16]. However our protocol uses the HTLC technique in a different way. That is, by adding the hash value into both $\mathtt{Tx}_{\mathsf{C}}$ and $\mathtt{Tx}_{\mathsf{R}}$, by the time of the settlement of the transaction $\mathtt{Tx}_{\mathsf{R}}$ on the chain the preimage is publicly disclosed, thus M can redeem the value received within the channel.

Remark 3. The early description briefly outlines the actions performed, however it is critical to note that the transaction via a payment channel is done by synchronized changes in the channel status of the participants. In particular, the preimage revealing is not performed by an exchange of $\mathtt{Tx}_{\mathsf{R}}$. In fact, the "transfer of $\mathtt{Tx}_{\mathsf{R}}$" is an interactive process that both players instantiate the new state and revoke the previous one (as it is standard in payment channels).

5 Our Protocol: Maravedí

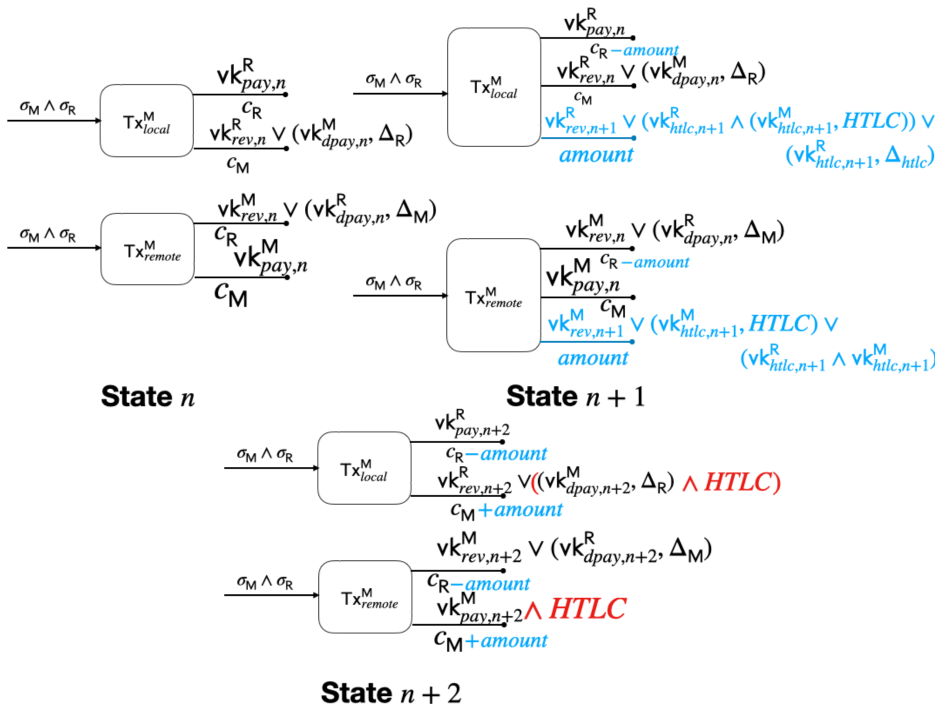
Here we thoroughly describe our construction for the model in Section 4. We start by providing a general intuition of the protocol.

5.1 Intuition

Our protocol relies on the channel between M and R and it is an adaptation from LN as it is described in [16]. Here we skip the full specification of the payment channel, we refer the reader to [16] for a thorough description. For our purposes it suffices to describe the final configuration between the two parties by the end of the setup of the channel. Then we concretely describe our protocol and our adaptation to the use of the HTLC technique.

After exchanging of initial keys, the established requested delay time, *i.e.*, Δ_{R} and Δ_{M}, the participants perform the signing and publication of, respectively, the initial Commit Transactions, *i.e.*, $\mathtt{Tx}_{local}$ and $\mathtt{Tx}_{remote}$, and the Funding Transaction $\mathtt{Tx}_{F_{\mathsf{R}}}$. Each payment received by the Merchant M from the Customer C is processed via the payment channel transaction (instantaneous payment) and a regular transaction (slow payment, performed in the ledger). Therefore the execution of the protocol starts from an initial set up, with $n = 1$ with the initial Commit Transaction. At a later moment, upon receiving a customer transaction, with the last signed Commit Transaction with index n, the protocol progresses to a new (intermediate) state for HTLC, *i.e.*, $n = n + 1$, and then to the final balance with (valid) state $n = n + 2$. Figure 2 illustrates the progress through the three states. Namely, it outlines the inclusion, and the removal, of the outputs into the locally kept pairs of Commit Transactions.

The closing of the channel works like a regular LN channel, *i.e.*, as described in [16]. Our protocol leverages on the original HTLC technique by atomically conditioning the payment channel transaction and the regular transaction, by adding the HTLC value to the latter and the former. With this setting, by the time R cashes in the payment via the ledger, it reveals the HTLC preimage allowing M to also redeem its own payment within the channel, even if R did not reveal the value previously. Note that, as in the regular payment channel, it is required that both participants are online.



$\sigma_M \wedge \sigma_R$
Tx^M_{local}
$vk^R_{pay,n}$
c_R
$vk^R_{rev,n} \vee (vk^M_{dpay,n}, \Delta_R)$
c_M
Tx^M_{remote}
$vk^M_{rev,n} \vee (vk^R_{dpay,n}, \Delta_M)$
$vk^M_{pay,n}$
$c_R - amount$
$vk^R_{rev,n+1} \vee (vk^R_{htlc,n+1} \wedge (vk^M_{htlc,n+1}, HTLC)) \vee (vk^R_{htlc,n+1}, \Delta_{htlc})$
$amount$
$vk^M_{rev,n+1} \vee (vk^M_{htlc,n+1}, HTLC) \vee (vk^R_{htlc,n+1} \wedge vk^M_{htlc,n+1})$
State n
State $n+1$
$vk^R_{pay,n+2}$
$vk^R_{rev,n+2} \vee ((vk^M_{dpay,n+2}, \Delta_R) \wedge HTLC)$
$c_M + amount$
$vk^M_{rev,n+2} \vee (vk^R_{dpay,n+2}, \Delta_M)$
$vk^M_{pay,n+2} \wedge HTLC$
State $n+2$

Fig. 2. This sequence of states illustrates the transferring of *amount*, and it depicts as the pairs of Commit Transactions kept locally by the merchant M. Note that in the intermediate state n+1, in order to redeem the outputs, it is necessary to disclose the preimage of the HTLC value or wait for the delay Δ_{htlc}. Moreover the final state $n+2$ the amount is transferred to the final outputs and conditioned to the revealing of the HTLC value. We highlight the update in the transactions in the intermediate state in blue, and the adaption of our protocol atomically payment between M and R in red.

5.2 Concrete Construction

Maravedí progresses from the initial setting, the "pre-processing" phase, which is the creation of a regular LN channel. Thus it progresses as the phases outlined in Section 4. Hence we start from the initial state and by quickly reviewing the initial key generation. We remark that the initial phase is a regular creation of the LN channel, therefore we only briefly recall it. We refer the reader to a full discussion in [16].

Protocol Setup Up. As a regular LN channel construction, R and M generate private information and exchange cryptography keys. They are

– Randomnesses: seed^i for $i \in \{\mathsf{R}, \mathsf{M}\}$ which is used with the PRF in order to generate new randomness for the keys to be derived on each state of the channel;
– IBS Master keys: These keys, as pointed in [16], are necessary to optimize the exchange of keys. They are $\textsc{Gen}(1^\lambda) \rightarrow (\mathsf{mvk}^i_{dpay}, \mathsf{msk}^i_{dpay}), (\mathsf{mvk}^i_{pay}, \mathsf{msk}^i_{pay})$, $(\mathsf{mvk}^i_{htlc}, \mathsf{msk}^i_{htlc})$, $(\mathsf{vk}^i_{com}, \mathsf{sk}^i_{com})$ for $i \in \{\mathsf{R}, \mathsf{M}\}$, and they are used to derive new keys for the each new state;
– CS Master keys: This key pair is also used to optimize the key exchange. Furthermore it is used to generate the revocation key which allows the participant to revoke an old state of the channel. The key pairs are given by setting $\textsc{MasterKeyGen}(1^\lambda) \rightarrow (\mathsf{mvk}^i_{rev}, \mathsf{msk}^i_{rev})$ for $i \in \{\mathsf{R}, \mathsf{M}\}$.

These initial keys are used to generate and derive new keys for each state. Concretely, R has $\mathsf{vk}_\mathsf{R}, \mathsf{sk}_\mathsf{R}$ to create the channel, and starts by generating, for the first state, $(\mathsf{vk}^\mathsf{R}_{com,1}, \mathsf{sk}^\mathsf{R}_{com,1})$, $\mathsf{vk}^\mathsf{R}_{pay,1}$, $\mathsf{vk}^\mathsf{R}_{rev,1}$, $\mathsf{vk}^\mathsf{R}_{dpay,1}$ $\mathsf{vk}^\mathsf{R}_{htlc,2}$ (and its respective secret keys), and keeps track of $\mathsf{vk}^\mathsf{M}_{rev,1}$, $\mathsf{vk}^\mathsf{M}_{pay,1}$, $\mathsf{vk}^\mathsf{M}_{dpay,1}$, $\mathsf{vk}^\mathsf{M}_{htlc,2}$ which are M public keys. Vice-versa for M which also keeps track of R's verification keys. Note that both parties also have their respective keys regarding the Funding Transaction vk_R and vk_M.

Later, R accesses its state Σ_R in the ledger to issue the Funding Transaction, and exchanges it (after the local Commit Transactions are securely signed and exchanged). In the end of the process, R keeps the following two Commit Transactions

$$\mathtt{Tx}^\mathsf{R}_{local} \leftarrow \mathtt{Tx}\{\mathcal{I} : (\mathtt{Tx}_{F_\mathsf{R}}.\mathcal{O}[1]); \mathcal{O} : [1 : c, \mathsf{vk}^\mathsf{M}_{rev,1} \vee (\mathsf{vk}^\mathsf{R}_{dpay,1}, \Delta_\mathsf{M})], [2 : 0, \mathsf{vk}^\mathsf{M}_{pay,1}]\},$$

$$\mathtt{Tx}^\mathsf{R}_{remote} \leftarrow \mathtt{Tx}\{\mathcal{I} : (\mathtt{Tx}_{F_\mathsf{R}}.\mathcal{O}[1]); \mathcal{O} : [1 : c, \mathsf{vk}^\mathsf{R}_{pay,1}], [2 : 0, \mathsf{vk}^\mathsf{R}_{rev,1} \vee (\mathsf{vk}^\mathsf{M}_{dpay,1}, \Delta_\mathsf{R})]\},$$

while M keeps

$$\mathtt{Tx}^\mathsf{M}_{local} \leftarrow \mathtt{Tx}\{\mathcal{I} : (\mathtt{Tx}_{F_\mathsf{R}}.\mathcal{O}[1]); \mathcal{O} : [1 : c, \mathsf{vk}^\mathsf{R}_{pay,1}], [2 : 0, \mathsf{vk}^\mathsf{R}_{rev,1} \vee (\mathsf{vk}^\mathsf{M}_{dpay,1}, \Delta_\mathsf{R})]\},$$

$$\mathtt{Tx}^\mathsf{M}_{remote} \leftarrow \mathtt{Tx}\{\mathcal{I} : (\mathtt{Tx}_{F_\mathsf{R}}.\mathcal{O}[1]); \mathcal{O} : [1 : c, \mathsf{vk}^\mathsf{M}_{rev,1} \vee (\mathsf{vk}^\mathsf{R}_{dpay,1}, \Delta_\mathsf{M})], [1 : 0, \mathsf{vk}^\mathsf{M}_{pay,1}]\},$$

with both participants having either the signature σ_R or σ_M for each of the locally kept transactions. Note that both transactions rely on the original funds c provided by R in the Funding Transaction $\mathtt{Tx}_{F_\mathsf{R}}$. Although M can also add funds to the channel, as in a regular payment channel, in our construction, it just receives them from R, therefore a single source of funds suffices for us.

Each participant keeps track of the last n committed transactions, starting from $n = 1$, such that $\mathtt{Tx}_{remote} = \mathtt{Tx}_{remote,1}$ and $\mathtt{Tx}_{local} = \mathtt{Tx}_{local,1}$. Moreover, the channel is assigned with a flag `Channel` to monitor the channel with a pair of values $(id, note)$ initially set empty, *i.e.*, $\mathtt{Channel} \leftarrow \perp$, such that $note$ is a string which identifies the state of the channel, and id is an identifier for the payment to be performed.

Transaction Proposal. According to the early described model, C submits a transaction proposal $\mathsf{Tp} = (\mathsf{v}, \mathsf{vk}_\mathsf{C})$, for an amount v and C's verification key, to M which performs the following

– Picks a uniformly random unique identifier *payid*
– Sets $\texttt{Channel}^\mathsf{M} \leftarrow (payid, \cdot)$
– Checks if there is a payment channel with R with capacity greater than v
– Adds to the list `pendingGetPay` the tuple $(\mathsf{vk}_\mathsf{C}, \mathsf{v}, payid, \cdot,$ "Waiting update"$)$
– Submits $(\mathsf{vk}_\mathsf{C}, \mathsf{v}, payid)$ to R via a direct communication channel.

The lists `pendingPay` and `pendingGetPay` are used to keep track of the payments and their current status on either participants.

Risk Buying/Preparation. Given a hash function H, R picks the value of the HTLC preimage, and, jointly with M, prepares its pair of local transactions, *i.e.*, $\mathtt{Tx}_{local}$ and $\mathtt{Tx}_{remote}$. That is, R performs as follows upon receiving $(\mathsf{vk}_\mathsf{C}, \mathsf{v}, payid)$:

– Verifies if $\nexists(\mathsf{vk}_\mathsf{C}, \alpha, \beta, \theta, \gamma, payid, \cdot)$ in `pendingPay` for any α,β,θ and γ. Otherwise, close the channel
– Picks a random value x in the domain of H
– Sets $\texttt{Channel}^\mathsf{R} \leftarrow (payid, \cdot)$
– Sets $(\mathsf{vk}_\mathsf{C}, \mathsf{vk}_\mathsf{M}, H(x), x, \mathsf{v}, payid,$ "Waiting update"$)$ to `pendingPay`
– Picks Δ_{htlc}, for delay period, and sends $(payid, H(x), \Delta_{htlc})$ to M

Upon receiving the message $(payid, y, \Delta_{htlc})$ from R, M prepares its local transactions accordingly, which concretely means to add new outputs into the local kept transactions $\mathtt{Tx}_{local}$ and $\mathtt{Tx}_{remote}$. The new outputs are related to the value of the received hash value y. In other words, M proceeds to perform the following upon receiving $(payid, y, \Delta_{htlc})$:

– If $(\mathsf{vk}_\mathsf{C}, \mathsf{v}, payid, \cdot,$ "Waiting update"$) =$ `pendingGetPay`, then
  • Removes "Waiting update" from the entry, and adds y to the entry, *i.e.*, $(\mathsf{vk}_\mathsf{C}, \mathsf{v}, payid, y, \cdot) \in$ `pendingGetPay`
– Update its locally kept transactions by adding the third output (in blue) with respect to the HTLC state on each transaction:

$$\begin{aligned}\mathtt{Tx}^\mathsf{M}_{local} \leftarrow \mathtt{Tx}\{\mathcal{I} : (\mathtt{Tx}_{F_\mathsf{R}}.\mathcal{O}[1]); \mathcal{O} : [1 : c_\mathsf{R} - (\mathsf{v} - \mathsf{d}), \mathsf{vk}^\mathsf{R}_{pay,n}],\\ [2 : c_\mathsf{M}, \mathsf{vk}^\mathsf{R}_{rev,n} \vee (\mathsf{vk}^\mathsf{M}_{dpay,n}, \Delta_\mathsf{R})],\\ [3 : \mathsf{v} - \mathsf{d}, \mathsf{vk}^\mathsf{R}_{rev,n+1} \vee (\mathsf{vk}^\mathsf{R}_{htlc,n+1} \wedge (\mathsf{vk}^\mathsf{M}_{htlc,n+1}, y)) \vee (\mathsf{vk}^\mathsf{R}_{htlc,n+1}, \Delta_{htlc})]\},\end{aligned}$$

$$\begin{aligned}\mathtt{Tx}^\mathsf{M}_{remote} \leftarrow \mathtt{Tx}\{\mathcal{I} : (\mathtt{Tx}_{F_\mathsf{R}}.\mathcal{O}[1]); \mathcal{O} : [1 : c_\mathsf{R} - (\mathsf{v} - \mathsf{d}), \mathsf{vk}^\mathsf{M}_{rev,n} \vee (\mathsf{vk}^\mathsf{R}_{dpay,n}, \Delta_\mathsf{M})],\\ [2 : c_\mathsf{M}, \mathsf{vk}^\mathsf{M}_{pay,n}],\\ [3 : \mathsf{v} - \mathsf{d}, \mathsf{vk}^\mathsf{M}_{rev,n+1} \vee (\mathsf{vk}^\mathsf{M}_{htlc,n+1}, y) \vee (\mathsf{vk}^\mathsf{M}_{htlc,n+1} \wedge \mathsf{vk}^\mathsf{R}_{htlc,n+1})]\},\end{aligned}$$

where c_R and c_M are the current balances, and the public keys derived from the initial state, in the channel for the $n-$th payment
– Forward $(\mathsf{v}, \mathsf{vk}_\mathsf{R}, y)$ to C, which replies with $(\mathtt{Tx}_\mathsf{C}, \sigma_\mathsf{C})$ such that $\mathtt{Tx}_\mathsf{C} \leftarrow \mathtt{Tx}\{\mathcal{I} : o \in \Sigma_\mathsf{C}; \mathcal{O} : [1 : \mathsf{v}, \mathsf{vk}_\mathsf{R} \wedge y]\}$ which pays directly to R (redeeming C's outputs in Σ_C) conditionally to the disclosure of the preimage of y. Hence M submits $(\textsc{Update}\mathrm{ADDHTLC}, payid)$ message to R, meaning R can update its locally kept transactions.

At this point, only M has prepared its internally kept transactions. Thus it is R's turn to set its transaction variables similarly.

- Verifies if $\exists(\mathsf{vk_C}, \alpha, \beta, \theta, \gamma, payid, note)$ in `pendingPay` for any α,β,θ and γ, and $note =$ "Waiting update". Otherwise, close the channel
- Upon receiving the message (UPDATEADDHTLC,$payid$), R updates (in blue) its local transactions as follows:

$$\begin{aligned}\mathtt{Tx}^{\mathsf{R}}_{local} \leftarrow \mathtt{Tx}\{\mathcal{I} : (\mathtt{Tx}_{F_\mathsf{R}}.\mathcal{O}[1]); \mathcal{O} : [1 : c_\mathsf{R} - (\mathsf{v} - \mathsf{d}), \mathsf{vk}^{\mathsf{M}}_{rev,n} \vee (\mathsf{vk}^{\mathsf{R}}_{dpay,n}, \Delta_\mathsf{M})],\\ [2 : c_\mathsf{M}, \mathsf{vk}^{\mathsf{M}}_{pay,n}],\\ [3 : \mathsf{v} - \mathsf{d}, \mathsf{vk}^{\mathsf{M}}_{rev,n+1} \vee (\mathsf{vk}^{\mathsf{M}}_{htlc,n+1}, y) \vee (\mathsf{vk}^{\mathsf{M}}_{htlc,n+1} \wedge \mathsf{vk}^{\mathsf{R}}_{htlc,n+1})]\};\end{aligned}$$

$$\begin{aligned}\mathtt{Tx}^{\mathsf{R}}_{remote} \leftarrow \mathtt{Tx}\{\mathcal{I} : (\mathtt{Tx}_{F_\mathsf{R}}.\mathcal{O}[1]); \mathcal{O} : [1 : c_\mathsf{R} - (\mathsf{v} - \mathsf{d}), \mathsf{vk}^{\mathsf{R}}_{pay,n}],\\ [2 : c_\mathsf{M}, \mathsf{vk}^{\mathsf{R}}_{rev,n} \vee (\mathsf{vk}^{\mathsf{M}}_{dpay,n}, \Delta_\mathsf{R})],\\ [3 : \mathsf{v} - \mathsf{d}, \mathsf{vk}^{\mathsf{R}}_{rev,n+1} \vee (\mathsf{vk}^{\mathsf{R}}_{htlc,n+1} \wedge (\mathsf{vk}^{\mathsf{M}}_{htlc,n+1}, y)) \vee (\mathsf{vk}^{\mathsf{R}}_{htlc,n+1}, \Delta_{htlc})]\}.\end{aligned}$$

- R moves $(\mathsf{vk_C}, \mathsf{vk_M}, H(x), x, \mathsf{v}, payid, note)$ from `pendingPay` to `paid`, and sets `pendingPay` $= \perp$
- Sets $(payid, x) \rightarrow$ `pendingFulfill`

***Risk Buying/Agreement-*R *Side*.** At this point, M and R proceed to create new HTLC Commitment transactions which relies on the previously updated transactions, respectively redeeming their HTLC outputs. First, R performs the following, assuming, as before, $\mathtt{Tx}^{\mathsf{R}}_{remote,n}$ is the last signed transaction:

- Verifies if $\mathtt{Tx}^{\mathsf{R}}_{remote} \neq \mathtt{Tx}^{\mathsf{R}}_{remote,n}$. Otherwise stop
- Verifies $(payid, note) = \mathtt{Channel}^{\mathsf{R}}$, such that $note \neq$ "waiting for REVOKEANDACK"
- Sets $\mathtt{Tx}^{\mathsf{R}}_{remote,n+1} \leftarrow \mathtt{Tx}^{\mathsf{R}}_{remote}$
- Sets $\sigma_{remote,n+1,\mathsf{R}} \leftarrow \text{SIGNDS}(\mathtt{Tx}^{\mathsf{R}}_{remote,n+1}, \mathsf{sk_R})$
- Issue an unsigned Remote HTLC transaction

$$\begin{aligned}\mathtt{Tx}^{\mathsf{R}}_{remHTLC,n+1} \leftarrow \mathtt{Tx}\{\mathcal{I} : (\mathtt{Tx}^{\mathsf{R}}_{remote,n+1}.\mathcal{O}[3]);\\ \mathcal{O} : [1 : \mathsf{v} - \mathsf{d}, \mathsf{vk}^{\mathsf{R}}_{rev,n+1} \vee (\mathsf{vk}^{\mathsf{M}}_{dpay,n+1}, \Delta_\mathsf{R})]\}\end{aligned}$$

- Set $\sigma_{htlc,n+1,\mathsf{M}} \leftarrow \text{SIGNIBS}(\mathtt{Tx}^{\mathsf{R}}_{remHTLC,n+1}, \mathsf{sk}^{\mathsf{R}}_{htlc,n+1})$
- Set $note \leftarrow$ "waiting for REVOKEANDACK", and $\mathtt{Channel}^{\mathsf{R}} \leftarrow (payid, note)$
- Send M the tuple $(payid, \sigma_{remote,n+1,\mathsf{R}}, \sigma_{htlc,n+1,\mathsf{R}})$

At this point R creates a new status by sending the necessary signatures to M (and revoking the previous state), which verifies its secret key $\mathsf{sh_R}$ for the $(n+1)$-th Commit Transaction, and transaction that redeems the newly added output (which sets the funds in a special state due to the HTLC technique). Therefore M performs the following upon receiving the signatures $(payid, ComSig, HTLCSig)$, respectively for the Commit and HTLC Transactions:

- Verifies $(payid, note) = \mathtt{Channel}^{\mathsf{M}}$, such that $note \neq$ "waiting for RevokeAndAck", otherwise stop
- Let $\mathtt{Tx}^{\mathsf{M}}_{local,n}$ be the last signed transaction, then it verifies if $\mathtt{Tx}^{\mathsf{M}}_{local} \neq \mathtt{Tx}^{\mathsf{M}}_{local,n}$. Otherwise stop
- If $\textsc{VerDS}(ComSig, \mathtt{Tx}^{\mathsf{M}}_{local}, \mathsf{vk}_{\mathsf{R}}) = 0$, then close the channel
- Sets

$$\mathtt{Tx}^{\mathsf{M}}_{localHTLC,n+1} \leftarrow \mathtt{Tx}\{\mathcal{I} : \mathtt{Tx}^{\mathsf{M}}_{local}.\mathcal{O}[3]; \\ \mathcal{O} : [1 : \mathsf{v} - \mathsf{d}, \mathsf{vk}^{\mathsf{R}}_{rev,n+1} \vee (\mathsf{vk}^{\mathsf{M}}_{dpay,n+1}, \Delta_{\mathsf{R}})]\}$$

- If $\textsc{VerifyIBS}(HTLCSig, \mathtt{Tx}^{\mathsf{M}}_{localHTLC,n+1}, \mathsf{vk}^{\mathsf{R}}_{htlc,n+1}) = 0$, then close the channel
- Sets $note \leftarrow$ " irrevocably committed" and adds $(\mathtt{Tx}^{\mathsf{M}}_{local}, note)$ to the list $\mathtt{pendingLocalCom}^{\mathsf{M}}$
- Sets $\mathsf{prand}^{\mathsf{M}}_{n+2} \leftarrow \mathrm{PRF}(\mathsf{seed}^{\mathsf{M}}, n+2)$
- Sets $(\mathsf{vk}^{\mathsf{M}}_{com,n+2}, sk^{\mathsf{M}}_{com,n+2}) \leftarrow \textsc{KeyShareGen}(1^{\lambda}, \mathsf{prand}^{\mathsf{M}}_{n+2})$
- Send $(payid, \mathsf{sk}^{\mathsf{M}}_{com,n}, \mathsf{vk}^{\mathsf{M}}_{com,n+2})$ to R

We remark that disclosing the previous key $\mathsf{sk}^{\mathsf{M}}_{com,n}$ is part of the revocation process to invalidate the previous state of the channel. Such a mechanism allows R to combine the keys (property of the Combined Signature Scheme as outlined in Section 2), and to obtain the secret revocation key $\mathsf{sk}^{\mathsf{R}}_{rev,n}$.

From now, on R's side, it starts the final step of this phase: verifies whether the received keys from M are correct, derive new ones and update the flags. Concretely, upon receiving $(payid, \mathsf{sk}^{\mathsf{M}}_{com,n}, \mathsf{vk}^{\mathsf{M}}_{com,n+2})$, R proceeds as follows:

- Retrieve $note$ from $\mathtt{Channel}^{\mathsf{R}}$, and verify if $note =$ "waiting for RevokeAndAck". Otherwise close the channel
- If $\textsc{TestKey}(\mathsf{vk}^{\mathsf{M}}_{com,n}, \mathsf{sk}^{\mathsf{M}}_{com,n}) = 0$, then close the channel
- Set $note \leftarrow$ "irrevocably committed", and add to the list $\mathtt{pendingRemoteCom}^{\mathsf{R}}$ the tuple $(\mathtt{Tx}^{\mathsf{R}}_{remote}, note)$
- Derive new keys by setting
  - $\mathsf{sk}^{\mathsf{R}}_{rev,n} \leftarrow \textsc{CombineKey}(\mathsf{msk}^{\mathsf{R}}_{rev}, \mathsf{msk}^{\mathsf{R}}_{rev}, \mathsf{vk}^{\mathsf{M}}_{com,n}, \mathsf{sk}^{\mathsf{M}}_{com,n})$
  - $\mathsf{vk}^{\mathsf{R}}_{rev,n+2} \leftarrow \textsc{CombinePubKey}(\mathsf{msk}^{\mathsf{R}}_{rev}, \mathsf{vk}^{\mathsf{M}}_{com,n+2})$
  - $\mathsf{vk}^{\mathsf{M}}_{rev,n+2} \leftarrow \textsc{CombinePubKey}(\mathsf{msk}^{\mathsf{R}}_{rev}, \mathsf{vk}^{\mathsf{R}}_{com,n+2})$
  - $(\mathsf{vk}^{\mathsf{R}}_{dpay,n+2}, \mathsf{sk}^{\mathsf{R}}_{dpay,n+2}) \leftarrow \textsc{KeyDer}(\mathsf{mvk}^{\mathsf{R}}_{dpay}, \mathsf{msk}^{\mathsf{R}}_{dpay}, \mathsf{vk}^{\mathsf{R}}_{com,n+2})$
  - $\mathsf{vk}^{\mathsf{M}}_{dpay,n+2} \leftarrow \textsc{PubKeyDer}(\mathsf{mvk}^{\mathsf{M}}_{dpay}, \mathsf{vk}^{\mathsf{M}}_{com,n+2})$
  - $(\mathsf{vk}^{\mathsf{R}}_{pay,n+2}, \mathsf{sk}^{\mathsf{R}}_{pay,n+2}) \leftarrow \textsc{KeyDer}(\mathsf{mvk}^{\mathsf{R}}_{pay}, \mathsf{msk}^{\mathsf{R}}, \mathsf{vk}^{\mathsf{R}}_{com,n+2})$
  - $\mathsf{vk}^{\mathsf{M}}_{pay,n+2} \leftarrow \textsc{PubKeyDer}(\mathsf{mvk}^{\mathsf{M}}_{pay}, \mathsf{vk}^{\mathsf{M}}_{com,n+2})$
  - $(\mathsf{vk}^{\mathsf{R}}_{htlc,n+2}, \mathsf{sk}^{\mathsf{R}}_{htlc,n+2}) \leftarrow \textsc{KeyDer}(\mathsf{mvk}^{\mathsf{R}}_{htlc}, \mathsf{msk}^{\mathsf{R}}_{htlc}, \mathsf{vk}^{\mathsf{R}}_{com,n+2})$
  - $\mathsf{vk}^{\mathsf{M}}_{htlc,n+2} \leftarrow \textsc{PubKeyDer}(\mathsf{mvk}^{\mathsf{M}}_{htlc}, \mathsf{vk}^{\mathsf{M}}_{com,n+2})$
- Set $note \leftarrow$ "$\cdot$", and update $\mathtt{Channel}^{\mathsf{R}}$

At this point R computed the $\mathsf{sk}^{\mathsf{R}}_{rev,n}$ meaning that the n-th state is "irrevocably revoked" because if the counterpart M tries to publish this state R can use it to redeem all the contents of the channel via the second output of $\mathtt{Tx}^{\mathsf{M}}_{local}$.

Furthermore, an intermediate state is partially created by $\mathtt{Tx}^{\mathsf{R}}_{remote}$ where the funds $(\mathsf{v}-\mathsf{d})$ are carried. In order to fully create the intermediate state, it is necessary that M performs its side of the procedure.

***Risk Buying/Agreement-*M *Side*.** The case is analogous to the R, which means that both states, for M and R, are committed. Briefly M generates the signature $\sigma_{remote,n+1,\mathsf{M}} \leftarrow \textsc{SignDS}(\mathtt{Tx}^{\mathsf{M}}_{remote,n+1}, \mathsf{sk}_{\mathsf{M}})$, for $\mathtt{Tx}^{\mathsf{M}}_{remote,n+1} \leftarrow \mathtt{Tx}^{\mathsf{M}}_{remote}$, and issues a unsigned remote HTLC transaction

$$\mathtt{Tx}^{\mathsf{M}}_{remHTLC,n+1} \leftarrow \mathtt{Tx}\{\mathcal{I} : (\mathtt{Tx}^{\mathsf{M}}_{remote,n}.\mathcal{O}[3]); \\ \mathcal{O} : [1 : \mathsf{v}-\mathsf{d}, \mathsf{vk}^{\mathsf{M}}_{rev,n+1} \vee (\mathsf{vk}^{\mathsf{R}}_{dpay,n+1}, \Delta_{\mathsf{M}})]\},$$

and sets

$$\sigma_{htlc,n+1,\mathsf{M}} \leftarrow \textsc{SignIBS}(\mathtt{Tx}^{\mathsf{M}}_{remHTLC,n+1}, \mathsf{sk}^{\mathsf{M}}_{htlc,n+1}),$$

and submits to R which verifies both $\sigma_{remote,n+1,\mathsf{M}}$ and $\sigma_{htlc,n+1,\mathsf{M}}$ (which R verifies with its locally kept $\mathtt{Tx}^{\mathsf{R}}_{localHTLC,n+1}$). Moreover M derives the new keys, namely, $\mathsf{vk}^{\mathsf{M}}_{rev,n+2}$, $\mathsf{vk}^{\mathsf{R}}_{rev,n+2}$, $(\mathsf{vk}^{\mathsf{M}}_{dpay,n+2}, \mathsf{sk}^{\mathsf{M}}_{dpay,n+2})$, $\mathsf{vk}^{\mathsf{R}}_{dpay,n+2}$, $(\mathsf{vk}^{\mathsf{M}}_{pay,n+2}, \mathsf{sk}^{\mathsf{M}}_{pay,n+2})$, $\mathsf{vk}^{\mathsf{R}}_{pay,n+2}$, $(\mathsf{vk}^{\mathsf{M}}_{htlc,n+2}, \mathsf{sk}^{\mathsf{M}}_{htlc,n+2})$, $\mathsf{vk}^{\mathsf{R}}_{htlc,n+2}$. In particular, M also computes the revocation key $\mathsf{sk}^{\mathsf{M}}_{rev,n}$ (in a similar fashion as R did).

Crucially, the participants keep the states in two lists, namely, for $i \in \{\mathsf{M}, \mathsf{R}\}$, $\mathtt{pendingLocalCom}^i$ and $\mathtt{pendingRemoteCom}^i$, in order to keep track of the committed transactions. This design guarantees that the new state is revoked only when the next one is signed on both sides.

***Settlement*.** Here, the previous state n is successfully revoked on both sides, however the funds are in the intermediated HTLC state $n+1$. For now it remains (1) to move to the final value to state $n+2$ (when M receives the funds); (2) R to reveal the preimage x to M; and (3) M to submit $(\mathtt{Tx}_{buy}, \sigma_{buy})$ to R. This final step allows R to cash the value later via the ledger when it publishes the transaction. However it is first necessary to finalize the payment within the channel, which means to revoke the state $n+1$. Therefore R verifies whether M has not submitted its transaction to the ledger as follows

- Reads the ledger to update the outputs in the state Σ_{R}
- R checks if M has not submitted transaction to the ledger. That is, if the outputs $\mathtt{Tx}^{\mathsf{R}}_{remote}.\mathcal{O} \notin \Sigma_{\mathsf{R}}$, then
  - Send message $(\textsc{Settlement}, payid)$ to M

  Otherwise
  - if $\exists(\alpha, \beta, \gamma, H(x), x, payid, note) = \mathtt{paid}$, then remove the entry
  - Set

$$\mathtt{Tx}^{\mathsf{R}}_{pay} \leftarrow \mathtt{Tx}\{\mathcal{I} : \mathtt{Tx}^{\mathsf{R}}_{remote,com}.\mathcal{O}[3]; \mathcal{O} : [1 : \mathsf{v}-\mathsf{d}, \mathsf{vk}^{\mathsf{R}}_{htlc,n}]\}$$

  - Set $\sigma_{pay} \leftarrow \textsc{SignIBS}(\mathtt{Tx}_{pay}, \mathsf{sk}^{\mathsf{R}}_{htlc,n})$
  - Submit $(\mathtt{Tx}_{pay}, \sigma_{pay})$ to ledger

In case M has not submitted the transaction to the ledger, then it receives $(\textsc{Settlement}, payid)$, and it performs

– Verifies $(payid, note) = \mathtt{Channel}^{\mathsf{M}}$, such that $note \neq$ "waiting for RevokeAndAck", otherwise stop
– M removes the outputs (added in the preparation phase) of its locally kept remote and local transactions, and update the balance. That is, it sets

$$\mathtt{Tx}^{\mathsf{M}}_{local} \leftarrow \mathtt{Tx}\{\mathcal{I} : (\mathtt{Tx}_{F_{\mathsf{R}}}.\mathcal{O}[1]); \mathcal{O} : [1 : c_{\mathsf{R}} - (\mathsf{v} - \mathsf{d}), \mathsf{vk}^{\mathsf{R}}_{pay,n+2}], [2 : c_{\mathsf{M}} + (\mathsf{v} - \mathsf{d}), \mathsf{vk}^{\mathsf{R}}_{rev,n+2} \vee ((\mathsf{vk}^{\mathsf{M}}_{dpay,n+2}, \Delta_{\mathsf{R}}) \wedge y)]\},$$

and

$$\mathtt{Tx}^{\mathsf{M}}_{remote} \leftarrow \mathtt{Tx}\{\mathcal{I} : (\mathtt{Tx}_{F_{\mathsf{R}}}.\mathcal{O}[1]); \mathcal{O} : [1 : c_{\mathsf{R}} - (\mathsf{v} - \mathsf{d}), \mathsf{vk}^{\mathsf{M}}_{rev,n+2} \vee (\mathsf{vk}^{\mathsf{R}}_{dpay,n+2}, \Delta_{\mathsf{M}})], [2 : c_{\mathsf{M}} + (\mathsf{v} - \mathsf{d}), \mathsf{vk}^{\mathsf{M}}_{pay,n+2} \wedge y]\}.$$

– Sets $\sigma_{remote,n+2,\mathsf{M}} \leftarrow \text{SignDS}(\mathtt{Tx}^{\mathsf{M}}_{remote}, \mathsf{sk}_{\mathsf{M}})$
– Sets $note \leftarrow$ "waiting for RevokeAndAck", and $\mathtt{Channel}^{\mathsf{M}} \leftarrow (payid, note)$
– Sends R the tuple $(payid, \sigma_{remote,n+2,\mathsf{M}})$

Note that the preimage of y (in red) is necessary to redeem the output, and it will only be available after R discloses it, upon receiving the transaction issued by C.

From this point, in case R verifies M's signature on its local version of the transaction it can revoke its old state $n+1$. Thus R starts by creating the locally kept Commit Transactions for state $n+2$. Upon receiving $(payid, ComSig)$, R does as follows:
– Analogous to the procedure performed by M, R updates its new state by removing the HTLC outputs and adjusting the balance:

$$\mathtt{Tx}^{\mathsf{R}}_{local} \leftarrow \mathtt{Tx}\{\mathcal{I} : (\mathtt{Tx}_{F_{\mathsf{R}}}.\mathcal{O}[1]); \mathcal{O} : [1 : c_{\mathsf{R}} - (\mathsf{v} - \mathsf{d}), \mathsf{vk}^{\mathsf{M}}_{rev,n+2} \vee (\mathsf{vk}^{\mathsf{R}}_{dpay,n+2}, \Delta_{\mathsf{M}})], [2 : c_{\mathsf{M}} + (\mathsf{v} - \mathsf{d}), \mathsf{vk}^{\mathsf{M}}_{pay,n+2} \wedge y]\}; \textit{and}$$

$$\mathtt{Tx}^{\mathsf{R}}_{remote} \leftarrow \mathtt{Tx}\{\mathcal{I} : (\mathtt{Tx}_{F_{\mathsf{R}}}.\mathcal{O}[1]); \mathcal{O} : [1 : c_{\mathsf{R}} - (\mathsf{v} - \mathsf{d}), \mathsf{vk}^{\mathsf{R}}_{pay,n+2}], [2 : c_{\mathsf{M}} + (\mathsf{v} - \mathsf{d}), \mathsf{vk}^{\mathsf{R}}_{rev,n+2} \vee ((\mathsf{vk}^{\mathsf{M}}_{dpay,n+2}, \Delta_{\mathsf{R}}) \wedge y)]\}.$$

– If $\text{VerDS}(ComSig, \mathtt{Tx}^{\mathsf{R}}_{local}, \mathsf{vk}_{\mathsf{M}}) = 0$, then close the channel
– Verifies $(payid, note) = \mathtt{Channel}$, such that $note \neq$ "waiting for RevokeAndAck"
– Sets $note \leftarrow$ " irrevocably committed" and $\mathtt{pendingLocalCom}^{\mathsf{R}} \leftarrow (\mathtt{Tx}^{\mathsf{R}}_{local}, note)$
– Sets $\mathsf{prand}^{\mathsf{R}}_{n+3} \leftarrow \text{PRF}(\mathsf{seed}^{\mathsf{R}}, n+3)$
– Sets $(\mathsf{vk}^{\mathsf{R}}_{com,n+3}, sk^{\mathsf{R}}_{com,n+3}) \leftarrow \text{KeyShareGen}(1^{\lambda}, \mathsf{prand}^{\mathsf{R}}_{n+3})$
– Sends $(payid, \mathsf{sk}^{\mathsf{R}}_{com,n+1}, \mathsf{vk}^{\mathsf{R}}_{com,n+3})$ to M

As before, the disclosing of $\mathsf{sk}^{\mathsf{R}}_{com,n+1}$ allows M to compute the revocation key and therefore R has revoked the $n+1$ state of the channel on its side. From this point, M can also revoke its state, and derive a whole new set of keys for the next state. The derivation on M's side is similar as before

- $\mathsf{sk}^{\mathsf{M}}_{rev,n+1} \leftarrow \textsc{CombineKey}(\mathsf{msk}^{\mathsf{M}}_{rev},\, \mathsf{msk}^{\mathsf{M}}_{rev},\, \mathsf{vk}^{\mathsf{R}}_{com,n+1},\, \mathsf{sk}^{\mathsf{R}}_{com,n+1})$
- $\mathsf{vk}^{\mathsf{M}}_{rev,n+3} \leftarrow \textsc{CombinePubKey}(\mathsf{msk}^{\mathsf{M}}_{rev}, \mathsf{vk}^{\mathsf{R}}_{com,n+3})$
- $\mathsf{vk}^{\mathsf{R}}_{rev,n+3} \leftarrow \textsc{CombinePubKey}(\mathsf{msk}^{\mathsf{M}}_{rev}, \mathsf{vk}^{\mathsf{M}}_{com,n+3})$
- $(\mathsf{vk}^{\mathsf{M}}_{dpay,n+3}, \mathsf{sk}^{\mathsf{M}}_{dpay,n+3}) \leftarrow \textsc{KeyDer}(\mathsf{mvk}^{\mathsf{M}}_{dpay},\, \mathsf{msk}^{\mathsf{M}}_{dpay},\, \mathsf{vk}^{\mathsf{M}}_{com,n+3})$
- $\mathsf{vk}^{\mathsf{R}}_{dpay,n+3} \leftarrow \textsc{PubKeyDer}(\mathsf{mvk}^{\mathsf{R}}_{dpay},\, \mathsf{vk}^{\mathsf{R}}_{com,n+3})$
- $(\mathsf{vk}^{\mathsf{M}}_{pay,n+3}, \mathsf{sk}^{\mathsf{M}}_{pay,n+3}) \leftarrow \textsc{KeyDer}(\mathsf{mvk}^{\mathsf{M}}_{pay},\, \mathsf{msk}^{\mathsf{M}},\, \mathsf{vk}^{\mathsf{M}}_{com,n+3})$
- $\mathsf{vk}^{\mathsf{R}}_{pay,n+3} \leftarrow \textsc{PubKeyDer}(\mathsf{mvk}^{\mathsf{R}}_{pay}, \mathsf{vk}^{\mathsf{R}}_{com,n+3})$
- $(\mathsf{vk}^{\mathsf{M}}_{htlc,n+3}, \mathsf{sk}^{\mathsf{M}}_{htlc,n+3}) \leftarrow \textsc{KeyDer}(\mathsf{mvk}^{\mathsf{M}}_{htlc},\, \mathsf{msk}^{\mathsf{M}}_{htlc},\, \mathsf{vk}^{\mathsf{M}}_{com,n+3})$
- $\mathsf{vk}^{\mathsf{R}}_{htlc,n+3} \leftarrow \textsc{PubKeyDer}(\mathsf{mvk}^{\mathsf{R}}_{htlc},\, \mathsf{vk}^{\mathsf{R}}_{com,n+3})$
- Set $note \leftarrow$ "·", and update $\mathtt{Channel}^{\mathsf{M}}$

Analogously, M revokes its state channel when verifying the signature of R with its locally kept transaction. Therefore R performs as follows

- Sets $\sigma_{remote,n+2,\mathsf{R}} \leftarrow \textsc{SignDS}(\mathtt{Tx}^{\mathsf{R}}_{remote}, \mathsf{sk}_{\mathsf{R}})$
- Sets $note \leftarrow$ "waiting for $\textsc{RevokeAndAck}$", and $\mathtt{Channel}^{\mathsf{R}} \leftarrow (payid, note)$
- Sends M the tuple $(payid, \sigma_{remote,n+2,\mathsf{R}})$

Upon receiving $(payid, Comsig)$, If $\textsc{VerDS}(ComSig, \mathtt{Tx}^{\mathsf{M}}_{local}, \mathsf{vk}_{\mathsf{R}}) = 0$, then close the channel. Otherwise M does the following:

- Verifies $(payid, note) = \mathtt{Channel}^{\mathsf{M}}$, such that $note \neq$ "waiting for $\textsc{RevokeAndAck}$"
- Sets $note \leftarrow$ "irrevocably committed" and $\mathtt{pendingLocalCom}^{\mathsf{M}} \leftarrow (\mathtt{Tx}^{\mathsf{R}}_{local}, note)$
- Sets $\mathsf{prand}^{\mathsf{M}}_{n+3} \leftarrow \mathrm{PRF}(\mathsf{seed}^{\mathsf{M}}, n+3)$
- Sets $(\mathsf{vk}^{\mathsf{M}}_{com,n+3}, sk^{\mathsf{M}}_{com,n+3}) \leftarrow \textsc{KeyShareGen}(1^{\lambda},\, \mathsf{prand}^{\mathsf{M}}_{n+3})$
- Sends $(payid, \mathsf{sk}^{\mathsf{M}}_{com,n+1}, \mathsf{vk}^{\mathsf{M}}_{com,n+3})$ to R

The secret key $\mathsf{sk}^{\mathsf{M}}_{com,n+1}$ is disclosed by M, triggering the key derivation phase and, therefore, the revocation of the old state. As before, R derives the new set of keys. For completeness, the keys are $\mathsf{vk}^{\mathsf{R}}_{rev,n+3}$, $\mathsf{vk}^{\mathsf{M}}_{rev,n+3}$, $(\mathsf{vk}^{\mathsf{R}}_{dpay,n+3}$, $\mathsf{sk}^{\mathsf{R}}_{dpay,n+3})$, $\mathsf{vk}^{\mathsf{M}}_{dpay,n+3}$, $(\mathsf{vk}^{\mathsf{R}}_{pay,n+3}$, $\mathsf{sk}^{\mathsf{R}}_{pay,n+3})$, $(\mathsf{vk}^{\mathsf{R}}_{htlc,n+3}$, $\mathsf{sk}^{\mathsf{R}}_{htlc,n+3})$, $\mathsf{vk}^{\mathsf{M}}_{pay,n+3}$, $\mathsf{vk}^{\mathsf{M}}_{htlc,n+3}$. In particular, R computes the revocation key $\mathsf{sk}^{\mathsf{R}}_{rev,n+1}$.

At this point, the only valid (not revoked) state is $n+2$, and the balance for R is $c_{\mathsf{R}} - (\mathsf{v} - \mathsf{d})$, whereas for M, it is $c_{\mathsf{M}} + (\mathsf{v} - \mathsf{d})$ (and initial amounts of, respectively, c_{R} and c_{M} before the payment). Note that the last signed transaction requires the preimage of y. Therefore both participants perform as follows

- M submits to R the tuple $(\mathtt{Tx}_{\mathsf{C}}, \sigma_{\mathsf{C}})$ received from C
- R removes $(payid, x)$ from $\mathtt{pendingFulfill}$, sets both $\mathtt{pendingFulfill}$ and $\mathtt{paid}$ to $\perp$, and submits x to M
- M sets $\mathtt{Channel}^{\mathsf{M}}$ and $\mathtt{pendingGetPay}$ to $\perp$
- M delivers the purchased good to C
- R submits $(\mathtt{Tx}_{\mathsf{C}}, \sigma_{\mathsf{C}}, x)$ to ledger, and sets $\mathtt{Channel}^{\mathsf{R}} \leftarrow \perp$

We remark that in order to cash in the whole amount (without the discount d), R publishes $(\mathtt{Tx}_{\mathsf{C}}, \sigma_{\mathsf{C}}, x)$ which allows that, even without communicating directly to M the value x, M to obtain the value from the ledger and therefore can redeem the payment within the channel.

Remark 4. We emphasize that closing the channel, cooperatively or not, in our construction is done with the same procedure of a regular LN channel. Thus, we refer the reader to [16] for a full discussion on the matter.

6 Security Analysis

Here we discuss two complementary security features of our construction: the (1) payment channel based security and its performance, and the (2) risk trade security. While the former is based on the security of LN and its financial punishment mechanism, the latter formally proves the novel property of our construction, *i.e.*, the guaranteed payment between the Merchant M and the Risk Buyer R. This is the main novel security result of this work. We later discuss its resistance against *collusion* attacks. Finally we review our construction in the light of the identified desiderata in Section 3.

We start by discussing the online security of our construction.

6.1 Online Security and Performance

The first key observation, and also a downside of our construction, is that both M and R need to be online during the execution of the protocol. That is a direct feature of the payment channel design we rely on for the instantaneous payment property. However, we remark that solutions like watchtowers [1,2] can be used along with our construction which minimizes the online time.

The online period is necessary because either one of the participants can maliciously claim an old state (possibly with higher balance) of the channel. In such an attack the claimer publishes an old (already revoked) Commit Transaction. A closer look at the locally kept transactions tells us that the outputs of such transactions require a delay period in order to be redeemed (which could be either Δ_{M} or Δ_{R}) as concretely described in Section 5.2. Given the design of the state (in particular the locally kept transactions), another output can be redeemed instead with the revocation public key during this period (which could be either $\mathsf{vk}^{\mathsf{M}}_{rev,n}$ or $\mathsf{vk}^{\mathsf{R}}_{rev,n}$). In conclusion, with such a transaction published, the revocation key can be used by the attacked participant to take the funds from the channel and punish the attacker.

We recall that the progression of the protocol, from state n and $n+1$ as shown in Section 5.2 (state $n+2$ is not revoked), follows the order that (1) first the participant receives from the counterpart the signature to the next state, then (2) it releases the necessary private information for the computation of the revocation secret key (again, it could be either $\mathsf{sk}^{\mathsf{M}}_{rev,n}$ or $\mathsf{sk}^{\mathsf{R}}_{rev,n}$, depending on who is performing the attack). This strict step order guarantees that both participants are always committed to a transaction, which can be used to financially disincentivize the early described old state attack.

Lastly, it is not hard to see, but we remark for completeness, that the direct communication channels between the participants can be plain, *i.e.*, not needed to be secure, since the transactions/messages are signed. In particular, we remark that the transaction which carries the payment to R will be widely known for the verification of its validity in the ledger.

Collusion Cases. We start by pointing out that C has a security advantage as it directly transacts with M (not within a channel). The basic observation is that as far as C is concerned, the transaction is completed when it hands its signed transaction (during the *Risk Buying/Preparation Phase*) to M and it receives the purchased good. In particular, if the transaction is for a physical item, then it makes the transaction completion even more easily verifiable on C's side. To conclude, a collusion between M and R do not have effect, since C can verify its own payment along with the received item.

Similarly with the case when C colludes with either M or R. Here again, once C delivers its signed transaction to M (via the direct communication channel, not the payment channel), the protocol is carried by M and R as it is a single hop channel. Consequently, trust between C and M is not needed since they transact directly. Furthermore, trust between M and R is also not needed because they interact via the payment channel which already assumes mistrusting parties.

Given the security guarantees provided by the payment channel, we can say with confidence that the only attack case is when the adversary, which can be either M or R, after receiving its payment, does not follow the protocol and, thus, does not pay its counterpart. Our novel use of the HTLC technique allows the payment to both parties to be *atomic*, meaning if one receives the payment, the other will necessarily receive it too. We denote this property *Risk Trade Security*, and prove that Maravedí has this property in Section 6.2.

Performance. We argue that the efficiency is equivalent to the [24] as described in [16], since the number of operations is the same. Our construction leverages on a single pairwise channel, *i.e.*, between M and R, as C does not interact in the channel, therefore it is not a multihop channel network. We highlight that the phases, namely, *Risk Buying/Agreement-*R *Side Phase* and *Risk Buying/Agreement-*M *Side Phase*, jointly are equivalent to a single state change of the LN channel (which R and M do sequentially). Similarly to the *Settlement Phase*. That is both sides jointly progress to the next state of the channel, by performing operations locally and sequentially.

– *Risk Buying/Agreement-*R *Side* + *Risk Buying/Agreement-*M *Side*: 4 signature generations, 6 key combinations and 12 key derivations;
– *Settlement Phase*: 4 signature generations, 6 key combinations and 12 key derivations.

These figures are based on the construction in [16]. The only adaption that we introduce are the y values (in red) as in Figure 2. This small change makes us confident that in practice our construction is as efficient as LN implementations.

6.2 Risk Trade Security

Our design prevents that M, or R, receives the payment and prevents its respective counterpart to receive it too. Recall that our protocol deals with two types of transaction (1) regular transaction to be redeemed in the ledger (important to R), and (2) the payment transactions (important to M). The disclosing of the HTLC value x happens only in the final steps during the **Settlement** Phase which enforces the atomicity (in the sense of unlocking funds) of the transactions (1) and (2), despite of being in two different layers.

We concretely prove the early discussion in the following theorem.

Theorem 1. *Given that the protocol in Section 5.2 is jointly executed by* C, M *and* R *such that* C *issues* $(\mathtt{Tx}_{\mathsf{C}}, \sigma_{\mathsf{C}})$*, if* R *redeems* $\mathtt{Tx}_{\mathsf{C}}$*, then* M *redeems the funds received from* R.

Proof. Note that up to the Settlement Phase, M has received $(\mathtt{Tx}_{\mathsf{C}}, \sigma_{\mathsf{C}})$, however it has not handed it to R. Likewise, R has picked the HTLC value y, such that, $y = H(x)$ for the used hash function H. Furthermore, up to this phase the protocol has performed like a payment channel. In particular, it performs until the only valid (not revoked) state is $n+2$.

Now assume, a malicious Risk Buyer R^* which receives $(\mathtt{Tx}_{\mathsf{C}}, \sigma_{\mathsf{C}})$ from M, then aborts the protocol without disclosing x to M. Note that by aborting the protocol, M can also close the channel by signing the locally kept transaction

$$\begin{aligned}\mathtt{Tx}^{\mathsf{R}^*}_{local} \leftarrow \mathtt{Tx}\{&\mathcal{I} : (\mathtt{Tx}_{F_{\mathsf{R}^*}}.\mathcal{O}[1]);\\ &\mathcal{O} : [1 : c_{\mathsf{R}^*} - (\mathsf{v} - \mathsf{d}), \mathsf{vk}^{\mathsf{M}}_{rev,n+2} \vee (\mathsf{vk}^{\mathsf{R}^*}_{dpay,n+2}, \Delta_{\mathsf{M}})],\\ &\qquad\qquad [2 : c_{\mathsf{M}} + (\mathsf{v} - \mathsf{d}), \mathsf{vk}^{\mathsf{M}}_{pay,n+2} \wedge y]\};\end{aligned}$$

and publishing $(\mathtt{Tx}^{\mathsf{R}^*}_{local}, \sigma_{\mathsf{R}}, \sigma_{\mathsf{M}})$ for the priorly received σ_{R} from R (when the state was created). Note $\mathtt{Tx}^{\mathsf{R}^*}_{local}$ redeems the outputs from the Funding Transaction $\mathtt{Tx}_{F_{\mathsf{R}^*}}$ but cannot be further redeemed by M as it did not receive the HTLC preimage x.

Now assume that R^* redeems its received transaction $\mathtt{Tx}_{\mathsf{C}}$, such that $\mathtt{Tx}_{\mathsf{C}} \leftarrow \mathtt{Tx}\{\mathcal{I} : o \in \Sigma_{\mathsf{C}}; \mathcal{O} : [1 : \mathsf{v}, \mathsf{vk}_{\mathsf{R}^*} \wedge y]\}$, meaning it has publicly disclosed σ_{R} and x, such that $y = H(x)$. Consequently, M can now issue a transaction, say $\mathtt{Tx}_{redeem}$, such that $\mathtt{Tx}_{redeem} \leftarrow \mathtt{Tx}\{\mathcal{I} : (\mathtt{Tx}^{\mathsf{R}^*}_{local}.\mathcal{O}[1]); \mathcal{O} : [1 : c_{\mathsf{M}} - (\mathsf{v} - \mathsf{d}), \mathsf{vk}_{\mathsf{M}}]\}$, signs it and publishes the tuple $(\mathtt{Tx}_{redeem}, \sigma_{\mathsf{M}}, x)$ in order to transfer $c_{\mathsf{M}} - (\mathsf{v} - \mathsf{d})$ to its known key vk_{M}, thereby giving the theorem.

Remark 5 (A Note on Collusion). Note that the earlier theorem is enough to show the security of our protocol even regarding collusion of any two of the participants as it was also discussed in Section 6.1. Briefly, since C only issues a regular transaction (for a physical or digital good) as it is not part of the pairwise channel, a meaningful collusion attack would pair C with either (1) M or (2) R. In both cases, it is covered by the security of the channel construction, *i.e.*, it is still a single hop channel but the colluding C is playing along with either of the sides of the channel. In case of (1), R cannot use the received (regular) transaction to transfer the funds, therefore it aborts the protocol without disclosing the HTLC value. Thus preventing any loss of funds since. On the other hand, in case of (2), all the funds to be used for payment would be in control of the colluding parties, *i.e.*, C and R, except for the payments already performed to M, which are protected by the past states of the channel. If R (and C) denies the payment to M in the channel, M and R funds will be locked in the channel and C does not receive the purchased goods (M does not hand it). Furthermore, R cannot receive its funds in the ledger (which would reveal the HTLC value). However, we showcase that R is financially incentivized to not abort the protocol, and, therefore, publish the transaction in the ledger, given the small profit on each transaction it receives.

6.3 Desiderata

We start by observing that our hybrid approach of relying on the LN protocol in order to "buy the risk" already provides us with the *Instantaneous Finality* and *Instantaneous Settlement* properties from Section 3. It also easily allows the requirement of *Direct Interaction*, since transactions are exchanged directly between the participants in such a protocol. Our design also puts forth a *Allowed Pre-processing* phase which in our case is the creation of the channel itself.

The creation of the LN channel does not require trust between the participants, therefore our construction also supports *Trustless Merchant* and *Trustless Risk Buyer*. Furthermore the interaction between the Customer and the Merchant does not require any trust, and can be readily available since it just requires the direct exchange of transactions. Therefore we also have *Ad Hoc Availability.*

In the case of *Trustless Trade*, our protocol does not address the situation of a malicious Merchant which may deny receiving the initial transaction. We highlight that although it may abort the protocol, it is not financially incentivized to do so, since it is expected to receive funds from the Risk Buyer.

It is important to notice that our protocol is generic enough to support *Background Availability* both for the Customer and the Risk Buyer, although we did not explore more this property in this work. One example is that the Customer, prior to the interaction of the Merchant, may have received a list of addresses of the Risk Buyer, therefore it only issues transactions to the addresses in the list, leaving no room for the Merchant to try to deviate the funds.

Finally, Section 6.2 showed a formal proof that our protocol provides *Risk Trade Security* for both the Merchant and the Risk Buyer in the sense that both are guaranteed to not lose funds.

7 Final Remarks

This work presented the Maravedí Protocol which introduces a novel technique to implement *instantaneous finality* for transactions. Instead of previous works, which focus on techniques to provide faster finality of blocks in a consensus protocol, our technique relaxes that criteria by addressing the finality for a single transaction for a single participant, the Merchant. The main intuition is that the receiver of the transaction relies on a third party which covers the risk of the transaction not be fulfilled in the consensus layer.

We investigated the early mentioned risk trade approach and discussed the main ideas for the "risk trade" in a 3-player model of Customer-Merchant-Risk Buyer, outlining its desiderata. We have emphasized that relying on TPS ration to discuss efficiency with decentralized ledger protocols, and in particular to compare it with credit card companies network, may be misleading. The main reason is that blockchain based systems rely on 2-party, *i.e.*, payer and payee, while ours allows the relaxation of the definition of *finality* and the risk trade.

As mentioned earlier, our protocol guarantees that the Merchant will receive the funds while performing a transaction. The guarantee is instantaneous, hence *instantaneous finality*. We devised a hybrid technique which uses a consensus based transaction (slow settlement) and a payment channel transaction (instantaneous settlement). The Maravedí Protocol is based on the Lightning Network, however we proposed an adaption to assure that both the Merchant and the Risk Buyer receive the funds correctly.

Our novel design allows the Risk Buyer to profit on every transaction. It may seem that the game theoretical dynamics of such design may inspire new businesses. It is important to observe that we did not investigate composability properties of our protocol, similarly to what was done in [16] by employing the UC Framework. We leave the study of these properties, both game theoretical dynamics and composability for future work.

8 Acknowledgements

We would like to thank the anonymous reviewers of Africacrypt Conference 2023 for the insightful and detailed comments on the initial submission. Their comments were gladly added to this manuscript.

References

1. Georgia Avarikioti, Orfeas Stefanos Thyfronitis Litos, and Roger Wattenhofer. Cerberus channels: Incentivizing watchtowers for bitcoin. Cryptology ePrint Archive, Report 2019/1092, 2019. `https://eprint.iacr.org/2019/1092`.
2. Zeta Avarikioti, Orfeas Stefanos Thyfronitis Litos, and Roger Wattenhofer. Cerberus channels: Incentivizing watchtowers for bitcoin. In Joseph Bonneau and Nadia Heninger, editors, *FC 2020: 24th International Conference on Financial Cryptography and Data Security*, volume 12059 of *Lecture Notes in Computer Science*, pages 346–366, Kota Kinabalu, Malaysia, February 10–14, 2020. Springer, Heidelberg, Germany.
3. Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance, 1999.
4. Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 66–98, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.
5. Thomas Dinsdale-Young, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. Afgjort: A partially synchronous finality layer for blockchains. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20: 12th International Conference on Security in Communication Networks*, volume 12238 of *Lecture Notes in Computer Science*, pages 24–44, Amalfi, Italy, September 14–16, 2020. Springer, Heidelberg, Germany.
6. Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment hubs over cryptocurrencies. In *2019 IEEE Symposium on Security and Privacy*, pages 106–123, San Francisco, CA, USA, May 19–23, 2019. IEEE Computer Society Press.

7. Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. General state channel networks. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 949–966, Toronto, ON, Canada, October 15–19, 2018. ACM Press.
8. Cardano Stack Exchange. Confusion about the time until true immutability. `https://cardano.stackexchange.com/questions/8943/confusion-about-the-time-until-true-immutability`, 2022. [Online; accessed 12-September-2022].
9. Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 51–68, New York, NY, USA, 2017. ACM.
10. Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A "paradoxical" solution to the signature problem (abstract) (impromptu talk). In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, page 467, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Heidelberg, Germany.
11. Pay Hub. Payments hub. `https://paymentshub.io`. [Online; accessed February-2023].
12. Maxim Jourenko, Kanta Kurazumi, Mario Larangeira, and Keisuke Tanaka. SoK: A taxonomy for layer-2 scalability related protocols for cryptocurrencies. Cryptology ePrint Archive, Report 2019/352, 2019. `https://eprint.iacr.org/2019/352`.
13. Maxim Jourenko, Mario Larangeira, and Keisuke Tanaka. Lightweight virtual payment channels. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, *CANS 20: 19th International Conference on Cryptology and Network Security*, volume 12579 of *Lecture Notes in Computer Science*, pages 365–384, Vienna, Austria, December 14–16, 2020. Springer, Heidelberg, Germany.
14. Maxim Jourenko, Mario Larangeira, and Keisuke Tanaka. Payment trees: Low collateral payments for payment channel networks. In Nikita Borisov and Claudia Díaz, editors, *FC 2021: 25th International Conference on Financial Cryptography and Data Security, Part II*, volume 12675 of *Lecture Notes in Computer Science*, pages 189–208, Virtual Event, March 1–5, 2021. Springer, Heidelberg, Germany.
15. Rami Khalil, Alexei Zamyatin, Guillaume Felley, Pedro Moreno-Sanchez, and Arthur Gervais. Commit-chains: Secure, scalable off-chain payments. Cryptology ePrint Archive, Paper 2018/642, 2018. `https://eprint.iacr.org/2018/642`.
16. Aggelos Kiayias and Orfeas Stefanos Thyfronitis Litos. A composable security treatment of the lightning network. Cryptology ePrint Archive, Report 2019/778, 2019. `https://eprint.iacr.org/2019/778`.
17. Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and privacy with payment-channel networks. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 455–471, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.
18. Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. Sprites and state channels: Payment networks that go faster than lightning. In Ian Goldberg and Tyler Moore, editors, *FC 2019: 23rd International Conference on Financial Cryptography and Data Security*, volume 11598 of *Lecture Notes in Computer Science*, pages 508–526, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019. Springer, Heidelberg, Germany.

19. Ayelet Mizrahi and Aviv Zohar. Congestion attacks in payment channel networks. In Nikita Borisov and Claudia Díaz, editors, *FC 2021: 25th International Conference on Financial Cryptography and Data Security, Part II*, volume 12675 of *Lecture Notes in Computer Science*, pages 170–188, Virtual Event, March 1–5, 2021. Springer, Heidelberg, Germany.
20. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
21. Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 3–33, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.
22. Kenneth G. Paterson and Jacob C. N. Schuldt. Efficient identity-based signatures secure in the standard model. In Lynn Margaret Batten and Reihaneh Safavi-Naini, editors, *ACISP 06: 11th Australasian Conference on Information Security and Privacy*, volume 4058 of *Lecture Notes in Computer Science*, pages 207–222, Melbourne, Australia, July 3–5, 2006. Springer, Heidelberg, Germany.
23. Cristina Pérez-Solà, Alejandro Ranchal Pedrosa, Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, and Joaquín García-Alfaro. LockDown: Balance availability attack against lightning network channels. In Joseph Bonneau and Nadia Heninger, editors, *FC 2020: 24th International Conference on Financial Cryptography and Data Security*, volume 12059 of *Lecture Notes in Computer Science*, pages 245–263, Kota Kinabalu, Malaysia, February 10–14, 2020. Springer, Heidelberg, Germany.
24. Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. *See https://lightning. network/lightning-network-paper. pdf*, 2016.
25. Kai Sedgwick. No, Visa Doesn't Handle 24,000 TPS and Neither Does Your Pet Blockchain. `https://news.bitcoin.com/no-visa-doesnt-handle-24000-tps-and-neither-does-your-pet-blockchain/`, 2018. [Online; accessed 12-September-2022].
26. Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Heidelberg, Germany.
27. Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.