Supplementary material to A^{2}L: Anonymous Atomic Locks for Scalability in Payment Channel Hubs

Abstract—**Payment channel hubs (PCHs) constitute a promising solution to the inherent scalability problems of blockchain technologies, allowing for off-chain payments between sender and receiver through an intermediary, called the tumbler. While state-of-the-art PCHs provide security and privacy guarantees against a malicious tumbler, they do so by relying on the scripting-based functionality available only at few cryptocurrencies, and they thus fall short of fundamental properties such as backwards compatibility and efficiency.**

In this work, we present the first PCH protocol to achieve all aforementioned properties. Our PCH builds upon A^{2}L, a novel cryptographic primitive that realizes a three-party protocol for conditional transactions, where the tumbler pays the receiver only if the latter solves a cryptographic challenge with the help of the sender, which implies the sender has paid the tumbler. We prove the security and privacy guarantees of A^{2}L (which carry over to our PCH construction) in the Universal Composability framework and present a provably secure instantiation based on adaptor signatures and randomizable puzzles. We implemented A^{2}L and compared it to TumbleBit, the state-of-the-art Bitcoin-compatible PCH. Asymptotically, A^{2}L has a communication complexity that is constant, as opposed to linear in the security parameter like in TumbleBit. In practice, A^{2}L requires ~33x less bandwidth than TumleBit, while retaining the computational cost (or providing 2x speedup with a preprocessing technique). This demonstrates that A^{2}L (and thus our PCH construction) is ready to be deployed today.

In theory, we demonstrate for the first time that it is possible to design a secure and privacy-preserving PCH while requiring only digital signatures and timelock functionality from the underlying scripting language. In practice, this result makes our PCH backwards compatible with virtually all cryptocurrencies available today, even those offering a highly restricted form of scripting language such as Ripple or Stellar. The practical appealing of our construction has resulted in a proof-of-concept implementation in the COMIT Network, a blockchain technology focused on cross-currency payments.

I. Introduction

The user base of cryptocurrencies, and more in general blockchain technologies, is rapidly increasing, embracing not only enthusiasts in decentralized payments like in the early days but also banks and leading IT companies (e.g., Facebook and PayPal), which are interested in providing services to connect users and enable secure and efficient payments, and more in general computations, between them. The realization of this vision, however, poses a number of technical challenges, most notably, *unlinkability, atomicity, interoperability, and scalability.*

A. Challenges in Blockchain Technologies: a Path Towards Payment Channel Hubs

Unlinkability. Neither individual users nor companies are willing to disclose the identity of their financial partners to the prying eyes of their competitors. Furthermore, the unlinkability of sender and receiver is an essential requirement even from an economical point of view: the *fungibility* of a currency requires that all coins have the same value. If one can determine by whom a certain coin has been processed, then coins could be valued differently by different users (e.g., coins of unknown provenance could be refused by some users).

The initial perception that Bitcoin provided unlinkability based on the use of public keys as pseudonyms has been largely refuted. Many academic efforts [1], [2] and the blockchain analysis industry [3] have demonstrated that it is possible to link pseudonyms together as well as to link them back to their real-world identities with little effort. Recent empirical analysis point out that deanonymization is an issue across virtually every cryptocurrency, even those designed with privacy-by-default principle such as Monero or Zcash [4]–[6].

In this state of affairs, a market of tumblers (also known as mixers or mixing services) has emerged, acting as opt-in overlays to existing cryptocurrencies that enhance privacy by mixing the coins from a set of senders to a set of receivers so that none can determine which sender paid to which receiver by inspecting the blockchain: for instance, JoinMarket, a mixing service based on the CoinJoin protocol, has been mixing 1M USD in bitcoins per month [7]. More sophisticated cryptographic protocols allow for the unlinkability of sender and receiver even against the participants in the mixing itself: for instance, CashShuffle has been used to mix over 40M USD in Bitcoin Cash coins since it was launched [8].

Atomicity. Mixers are not necessarily honest and, in particular, they might steal the money from honest users [9], [10]. A fundamental security property in the context of tumbler-based payments is thus *atomicity*, that is, either a payment is successful or the money goes back to the sender.

Interoperability. Inasmuch as payer and payee could possess wallets in different cryptocurrencies, payments, and more in general blockchain transactions, should be possible across different blockchains. In fact, exchange services are an essential component of the cryptocurrency ecosystem, with more and more banks (including PayPal) providing such functionality.

Scalability. The increasing adoption of cryptocurrencies has raised scalability issues [11] that go beyond the rapidly growing blockchain size. For instance, the permissionless nature of the consensus algorithm underlying widely deployed cryptocurrencies such as Bitcoin and Ethereum strictly limits their transaction throughput to tens of transactions per second at best [11], which contrasts with the throughput of centralized payment networks such as Visa that supports peaks of up to 47,000 transactions per second [12].

Among the several efforts to mitigate these scalability issues [13]–[15], payment channels have emerged as the most widely deployed solution in practice. The core idea is to let users deposit a certain amount of coins (called *collateral*) in a shared address [1] (called *channel*) controlled by both, storing the corresponding transaction on-chain. From that moment on, these two users can pay each other by simply agreeing on a new distribution of the coins deposited in the channel: the corresponding transactions are stored locally, that is, off-chain. When the two users disagree on the current redistribution or simply terminate their economic relation, they submit an on-chain transaction that sends back the coins to their owners according to the last agreed distribution of coins, thereby closing the channel. Thus, payment channels require only two on-chain transactions (i.e., open and close channel), yet supporting arbitrarily many off-chain payments, which significantly enhances the scalability of the underlying blockchain.

While appealing, this simple construction forces the sender to establish a channel with each possible receiver, which is financially prohibitive, as the sender would have to lock an amount of coins proportional to the number of receivers. Furthermore, the coins locked in a channel cannot be used anywhere else.

Payment channel networks (PCNs) (such as the Lightning Network [16]) offer a partial solution to this problem, enabling multi-hop payments along channel paths: if one sees a PCN as a graph where nodes are users and edges are channels, PCNs enable payments between any two nodes connected by a path. However, a PCN payment requires sequential channel updates from sender to receiver, breaking unlinkability when there is a single intermediary. Moreover, PCNs raise the issue of finding paths in a network and maintaining the network topology.

This has led to the design of tumblers supporting off-chain payments, also called payment channel hubs (PCHs). The basic idea is that each party opens a channel with a *tumbler*, which mediates payments between each pair of sender and receiver, receiving a fee in compensation.

B. Problem Statement and Related Work

Enforcing unlinkability, atomicity, and interoperability in cryptocurrencies in general, and even more so in an off-chain setting, is an open challenge. In particular, some privacy-preserving on-chain mixing protocols like CoinJoin [17] assume trusted users, thereby not providing strong unlinkability guarantees, while others like CoinShuffle [18], [19] or Möbius [20] (among many others) protect even against malicious users, but require custom scripting language from the underlying cryptocurrency (e.g., Bitcoin and Ethereum). Similar reasoning applies to privacy-preserving cryptocurrencies, like Monero or ZCash. In the off-chain setting, existing constructions require either a dedicated scripting language (e.g., Perun [21] and NOCUST [22] rely on Ethereum scripts, Tumblebit [23] on Hashed Timelock Contracts (HTLCs), and Bolt [24] on customized cryptographic primitives) or trusted hardware (e.g., Teechain [25]). Notice that even seemingly mild system assumptions like HTLCs hinder interoperability, as HTLCs are supported only in some cryptocurrencies (e.g., Bitcoin and Ethereum) but not by the so-called scriptless ones (e.g., Ripple and Stellar). Table I summarizes the assumptions and properties of state-of-the-art PCH constructions.

TABLE I: State-of-the-art in mixing services.

		Atomicity	Unlinkability	Interoperability (Required functionality)
On-chain	Trusted Gateways	○	○	●
	CoinJoin	●	○	○(CoinJoin tx)
	Mixing [18]–[20]	●	●	○(Bitcoin or Ethereum)
	Monero, ZCash, ...	●	●	○(Dedicated Currency)
	Tesseract [26]	●	●	○(Trusted Hardware)
Off-chain	BOLT [27]	●	●	○ (Zcash)
	Perun [28]	●	○	○ (Ethereum)
	NOCUST [22]	●	○	○ (Ethereum)
	Teechain [25]	●	●	◐[2] (Trusted Hardware)
	TumbleBit [23]	●	●[1]	◐[3] (HTLC-based currencies)
	A^2L	●	●[1]	● (Digital signature and timelocks)

[1] Payments have fixed amounts; [2] Every user must run a TEE; [3] Not supported by scriptless cryptocurrencies (e.g., Ripple and Stellar).

This state of affairs leads to the following question: *is it possible to design a PCH that provides atomicity, unlinkability, and interoperability (i.e., it is based on few assumptions that are fulfilled by virtually all cryptocurrencies)?*

This question, besides interesting from a theoretical point of view, is also of strong practical relevance. Indeed, such a PCH would enable, for the first time, tumbler-enabled atomic and unlinkable payments across arbitrary cryptocurrencies. In addition, realizing powerful blockchain applications with fewer scripting assumptions is a valuable research direction on its own. Besides the time required to implement a change in the consensus protocol and the low likelihood this is actually accepted, adding functionality to the underlying cryptocurrency increases the trusted computing base (i.e., checking that there are no inconsistencies with other functionalities) which in general exacerbates the problem of verifying scripts (e.g., bugs in Ethereum smart contracts add countless new attack vectors).

C. Our contributions

We present the first PCH construction that requires only digital signatures and timelock functionality from the underlying cryptocurrency. Furthermore, our construction is also the most efficient one among the Bitcoin compatible ones. Specifically,

- We introduce A^2L, a PCH based on a three-party protocol for conditional transactions, where the intermediary pays the receiver only if the latter solves a cryptographic challenge with the help of the sender, which implies that the sender has paid the intermediary. We provide an instantiation based on adaptor signatures, which in turn can be securely instantiated by well-known signature schemes such as Schnorr and ECDSA [29]. By dispensing from custom scripting functionality (e.g., HTLCs), our instantiations offer the highest degree of interoperability among the state-of-the-art PCHs: e.g., Ripple and Stellar support ECDSA and Schnorr but not HTLCs, whereas Mimblewimble [30] supports Schnorr

[1] Technically, a 2-of-2 multisig address, which requires both address owners to agree on the usage of the coins stored therein, which is achieved by signing the corresponding transaction.

but not HTLCs. Moreover, A^2L can be used as a classic on-chain tumbler, leveraging standard techniques to include off-chain operations as on-chain transactions (e.g., as in [23]).

- We model A^2L in the Universal Composability (UC) framework [31], proving the security (more precisely, atomicity and unlinkability) of our construction. UC is a popular proof technique for off-chain protocols (e.g., [21], [32], [33]) as it enables compositional proofs and this is the first formalization of PCHs in UC: this result allows, e.g., to lift the security of a PCH to off-chain applications relying on it as a building block.
- At the core of A^2L lies the novel concept of *randomizable puzzle*. This primitive supports the encoding of a challenge into a puzzle, its rerandomization, and its homomorphic solution (i.e., solving the randomized version of the puzzle reveals the randomized version for the challenge originally encoded in the puzzle). We define security and privacy for randomizable puzzles in the form of cryptographic games. Finally, we give a concrete construction based on an additively homomorphic encryption scheme and formally prove its security and privacy. We find randomizable puzzle as a contribution of interest on its own and leave the design of concrete constructions based on cryptographic building blocks other than additively homomorphic encryption as an interesting future work.
- Our evaluation shows that A^2L requires a running time of ~0.6 seconds. Furthermore, the communication cost is less than 10KB. When compared to TumbleBit, the most interoperable PCH prior to this work, A^2L has a communication complexity that is independent of the security parameter, whereas Tumblebit's one is linear. Our experimental evaluations shows that A^2L requires ~33x less bandwidth, and similar computation costs (or 2x speedup with a preprocessing technique), despite providing additional security guarantees, such as protection against griefing attacks. These results demonstrate that A^2L is the most efficient Bitcoin-compatible PCH. Our construction has already been implemented as a proof-of-concept in the COMIT Network (see Section VIII), an industrial technology for cross-currency payments.

II. Background

Following the notation in [29], a PCH can be represented as a graph, where each vertex represents a party P, and each edge represents a payment channel ς between two parties P_i and P_j, for $P_i, P_j \in \mathbb{P}$, where $\mathbb{P}$ denotes the set of all parties. We define a payment channel ς as an attribute tuple $(\varsigma.\mathsf{id}, \varsigma.\mathsf{users}, \varsigma.\mathsf{cash}, \varsigma.\mathsf{state})$, where $\varsigma.\mathsf{id} \in \{0,1\}^*$ is the channel identifier, $\varsigma.\mathsf{users} \in \mathbb{P}^2$ defines the identities of the channel users, and $\varsigma.\mathsf{cash}\colon \varsigma.\mathsf{users} \to \mathbb{R}^{\geq 0}$ is a mapping from channel users to their respective amount of coins in the channel. Finally, $\varsigma.\mathsf{state} = (\theta_1, \ldots, \theta_n)$ is the current state of the channel, which is composed of a list of deposit distribution updates θ_i.

A. PCH functionality

A PCH is defined with respect to a blockchain $\mathcal{B}$ and it is equipped with three operations: $\mathsf{OpenChannel}$, $\mathsf{CloseChannel}$, and Pay. While $\mathsf{OpenChannel}$ and $\mathsf{CloseChannel}$ are standard payment channel operations, Pay is tailored to PCHs as it involves a sender, a receiver, and a tumbler. We overview these operations here and refer the reader to Appendix C3 for the formalization of a PCH in the Universal Composability framework. In this overview, we denote by $\mathcal{B}[P_i]$ the amount of coins that P_i holds in the blockchain.

$\mathsf{OpenChannel}(P_i, P_j, \beta_i, \beta_j)$: If this operation is authorized by both users P_i and P_j and the condition $\mathcal{B}[P_i] \geq \beta_i \wedge \mathcal{B}[P_j] \geq \beta_j$ holds (i.e., users have enough money on the blockchain), this operation does the following: it (i) creates a payment channel ς with a fresh id $\varsigma.\mathsf{id}$, $\varsigma.\mathsf{users} = (P_i, P_j)$, $\varsigma.\mathsf{cash}(P_i) = \beta_i$, $\varsigma.\mathsf{cash}(P_j) = \beta_j$ and $\varsigma.\mathsf{state} = ()$; (ii) updates the blockchain as $\mathcal{B}[P_i] \mathrel{-}= \beta_i$ and $\mathcal{B}[P_j] \mathrel{-}= \beta_j$; and (iii) adds ς to the graph representing the PCH.

$\mathsf{CloseChannel}(\varsigma, P_i, P_j)$: If this operation is authorized by both users P_i and P_j and $\varsigma.\mathsf{users} = (P_i, P_j)$, this operations does the following: it (i) updates the blockchain as $\mathcal{B}[P_i] \mathrel{+}= \varsigma.\mathsf{cash}(P_i)$ and $\mathcal{B}[P_j] \mathrel{+}= \varsigma.\mathsf{cash}(P_j)$; and (ii) removes ς from the graph representing the PCH.

$\mathsf{Pay}(P_s, P_t, P_r, \beta)$: Let ς be the channel such that $\varsigma.\mathsf{users} = (P_s, P_t)$ and let ς' be the channel such that $\varsigma'.\mathsf{users} = (P_t, P_r)$. If this operation is authorized by all users P_s, P_t, P_r and the condition $\varsigma.\mathsf{cash}(P_s) \geq \beta \wedge \varsigma'.\mathsf{cash}(P_t) \geq \beta$ holds (i.e., the sender and the tumbler have enough money on the respective channels), this operation does the following: it (i) creates a new update $\theta = (\varsigma.\mathsf{cash}(P_s) \mathrel{-}= \beta, \varsigma.\mathsf{cash}(P_t) \mathrel{+}= \beta)$; (ii) creates a new update $\theta' = \varsigma'.\mathsf{cash}(P_t) \mathrel{-}= \beta\ \varsigma'.\mathsf{cash}(P_r) \mathrel{+}= \beta$; and (iii) appends θ to $\varsigma.\mathsf{state}$ and θ' to $\varsigma'.\mathsf{state}$.

We note that in practice for every successful payment P_t receives certain amount of fees, which incentivizes P_t to participate as an intermediary. We omit here the fees for the sake of readability, and discuss them further in Appendix A2.

B. Security and Privacy Goals

We overview the main security and privacy goals for PCHs, referring to Appendix C for the formal security and privacy model.

Authenticity. The PCH should only start a payment procedure if the sender P_s has been successfully registered by the tumbler P_t. We aim to achieve this property to avoid denial of service attacks, as we describe in Section III and Section VI-B.

Atomicity. The PCH should not be exploited to print new money or steal existing money from honest users, even when parties collude. This property thus aims to ensure balance security for the honest parties as in [23].

Unlinkability. The tumbler P_t should not learn information that allows it to associate the sender P_s and the receiver P_r of a payment. We define unlinkability in terms of an *interaction multi-graph* as in [23]. An interaction multi-graph is a mapping of payments from a set of senders to a set of receivers. For each successful payment completed upon a query from the sender P_s^j at epoch e, the graph has an edge, labeled with e, from the sender P_s^j to some receiver P_r^i. An interaction graph is *compatible* if it explains the view of the tumbler, that is, the number of edges labeled with e incident to P_r^i equals the number of coins received by P_r^i. Then, unlinkability requires all compatible interaction graphs to be equally likely. The anonymity set depends thus on the number of compatible interaction graphs.

III. Solution Overview

Inspired from TumbleBit [23], we design A^{2}L in two phases: *puzzle promise* and *puzzle solver*. Intuitively, during these two phases the update on the channel between P_t and P_r (i.e., the tumbler P_t paying coins to the receiver P_r) is established first but its success is conditioned on the successful update of the channel between P_s and P_t (i.e., the sender P_s paying coins to the tumbler P_t). In other words, the tumbler "promises in advance" a payment to the receiver under the condition that the sender successfully pays to the tumbler.

Authenticity. The aforementioned payment paradigm opens the door for a so-called *griefing attack* [34], where the receiver P_r starts many puzzle promise operations, each of which requires that the tumbler P_t locks coins, whereas the corresponding puzzle solver interactions are never carried out. As a consequence, all tumbler's coins are locked and no longer available, which results in a form of denial of service attack. Previous proposals to handle this attack [23] force P_r to pay for a transaction fee on-chain every time it triggers a puzzle promise. This approach, however, does not work in the off-chain setting, which is the focus of this paper. Moreover, the transaction fee that P_r pays is smaller than the amount of coins received in the PCH payment, thereby introducing an amplification factor, which undermines the effectiveness of this mitigation.

Our approach: We observe that in the considered payment paradigm, P_t is at risk. Our approach is to move the risk from P_t to the sender P_s by letting the latter lock coins in advance to prove P_t the willingness to participate in the protocol. This approach lines up the management of the collateral with the incentives of each player. First, the additional collateral (i.e., additional coins locked) is handled by the sender P_s, who is the party that wants to perform the payment in the first place. Second, the tumbler P_t may decide not to carry out the payment, putting however its reputation at stake (and a possible economic benefit in terms of fees as we discuss in Appendix A2).

Mitigating the above mentioned DoS attack requires a careful design to maintain the unlinkability of the payments. For instance, the receiver P_r could indicate to P_t the collateral that the corresponding P_s has locked for the payment to happen. This approach, however, would trivially hinder the unlinkability between P_r and P_s. We thus require a cryptographic mechanism that achieves two goals: (i) P_r can convince P_t that there exists a certain number of coins locked for this interaction without revealing which P_s locked the coins; and (ii) P_t should be able to check that the same collateral is not claimed twice.

We implement this functionality based on a lightweight variant of anonymous credentials, which in turn we base on a (blinded) randomizable signature scheme and non-interactive zero-knowledge proof. Intuitively, P_s locks coins into an address controlled by both P_s and P_t in such a manner that those coins are returned back to P_s after a certain time (i.e., the time to execute the rest of the protocol). Once P_t has verified that, it issues a credential to P_s, who randomizes the credential and forwards it to P_r. Finally, P_r forwards this randomized credential to P_t. At this point P_t can verify that there has been indeed a registration for such request while the randomization intuitively hides the link between P_r and P_s of a given payment. This corresponds to the *registration* phase in Figure 1.

Atomicity. As mentioned earlier, our payment paradigm relies on the fact that the tumbler "promises in advance" a payment to the receiver under the condition that the sender successfully pays to the tumbler. Atomicity thus relies on such conditional payments to ensure that either both payments are performed (i.e., both channels are updated) or none goes through.

Our approach: The technical challenge here resides on how to perform the aforementioned conditional payment. For that, we design *cryptographic puzzles*, a cryptographic scheme that encodes an instance of a cryptographic hard problem (e.g., find a valid pre-image of a given hash value). With that tool in place, our approach for atomicity is to redesign the channel update and tie it together with the puzzle in such a manner that we achieve the following two properties: (i) the channel update is enabled (i.e., added to the channel state) only if a solution to the puzzle is found; and (ii) a valid channel update can be used to extract the solution to the puzzle.

Intuitively, our approach ensures the atomicity of the payment between P_s and P_r as follows. During the puzzle promise phase, P_t creates a fresh cryptographic puzzle Z to which it already knows the solution. Then, P_t updates the channel with P_r conditioned on the puzzle Z. Note that at this point, P_r does not know the solution to Z, and thus, cannot release the coins. Instead, P_r could simply forward this puzzle Z to P_s, triggering thus the puzzle solver phase. In this phase, P_s pays P_t conditioned on P_t solving the puzzle Z. Since P_t has the solution (as P_t was the one that generated the puzzle), P_t can solve the puzzle and update the channel. As mentioned earlier, when P_t updates the channel with P_s, our protocol makes sure that P_s can extract the solution of Z from such a valid channel update. Finally, P_s can forward the solution to P_r who can in turn use it to solve the puzzle at its side and release the coins promised by P_t at the beginning of the promise phase.

Unlinkability. While the previous approach provides atomicity, it does not guarantee the unlinkability of the payments. Note that the same puzzle Z is used by both P_s and P_r, a common identifier that even an honest but curious P_t can use to link who is paying to whom.

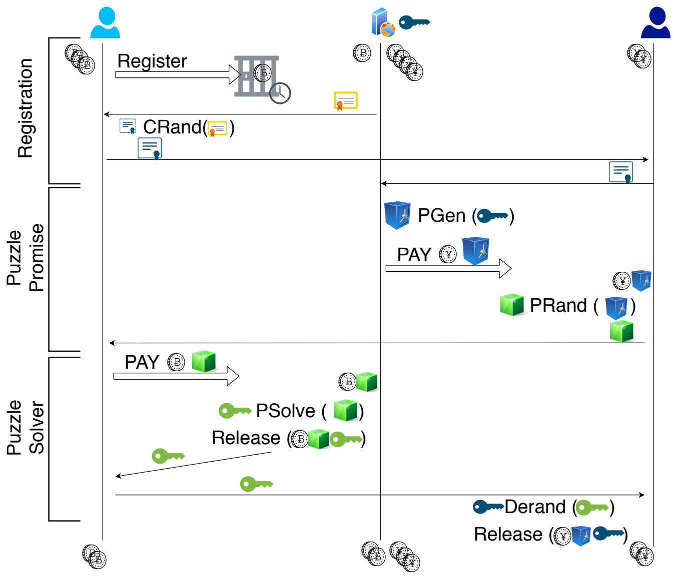

Fig. 1: Overview of our solution. Sender (left user) pays receiver (right user) via tumbler (middle user). The protocol is divided in three phases: (i) registration, (ii) puzzle promise, and (iii) puzzle solver. CRand denotes the randomization of the certificate. PGen, PRand and PSolve denote the generation, randomization and solving of a randomizable cryptographic puzzle. Pay denotes the update of a between two users. Derand denotes the derandomization of the solution for a puzzle and Release denotes the claim of the coins given a puzzle and its corresponding solution.

Our approach: We overcome this challenge by designing *cryptographic randomizable puzzles*, a novel cryptographic scheme that extends the notion of cryptographic puzzle with two intuitive functionalities: (i) given a certain puzzle Z, it is possible to randomize it into a puzzle Z' using a randomness r; (ii) the solution to the puzzle Z' corresponds to the randomized version (using randomness r) of the solution to the puzzle Z.

With cryptographic randomizable puzzles in place, our final solution which achieves authenticity, atomicity, and unlinkability works as depicted in Figure 1. During the puzzle promise phase, P_t generates (using PGen) a puzzle Z (i.e., the blue safe box), which gets randomized by P_r (using PRand) to obtain the randomized puzzle Z' (i.e., the green safe box) The puzzle promise phase ends when P_r sends Z' to P_s. During the puzzle solver phase, P_s pays to P_r attaching Z' as the condition for P_r to accept the payment. After P_t accepts the payment by solving the randomized puzzle Z' (using PSolve), P_s can extract the randomized solution (using Release) and forward it (i.e., the green key) to P_r, who in turn can derandomize this solution (using Derand) to obtain a solution to the original puzzle (i.e., the blue key), and use it solve the puzzle Z.

We devise an instantiation of randomizable puzzle that is based on the discrete logarithm problem and additively homomorphic encryption scheme. Moreover, we redesign the channel update so that it can be made valid only if the solution to the randomizable puzzle is found. For that, we use adaptor signatures, an extension of standard digital signatures that tie together the creation of a digital signature (and thus the authorization of a channel update) and the leakage of a secret value. In a nutshell, one can first generate a pre-signature with respect to the statement (i.e., in our case the randomizable puzzle), which can be converted to a valid signature only by knowing the secret (i.e., in our case the solution to the puzzle). Second, if the pre-signature is converted to a valid signature, one can extract the secret from the pair (pre-signature, valid signature).

We point out that for the sake of readability, throughout this section we have omitted the case where one of the users simply does not respond. For instance, it can be the case that P_t performs the conditional payment to P_r, and afterwards, no longer answers (e.g., it crashes). In order to handle this case, each conditional payment contains an expiration time after which the originator of the conditional payment can claim the coins back unconditionally. For instance, in the previous example, after the expiration time P_t could update the channel into a state where it no longer promises to pay coins to P_r

IV. Preliminaries

We denote by 1^λ, for $\lambda \in \mathbb{N}$, the security parameter. We assume that the security parameter is given as an implicit input to every function, and all our algorithms run in polynomial time in λ. We denote by $x \leftarrow_{\$} \mathcal{X}$ the uniform sampling of the variable x from the set $\mathcal{X}$. We write $x \leftarrow \mathsf{A}(y)$ to denote that a probabilistic polynomial time (PPT) algorithm A on input y, outputs x. We use the same notation also for the assignment of the computational results, for example, $s \leftarrow s_1 + s_2$. If A is a deterministic polynomial time (DPT) algorithm, we use the notation $x := \mathsf{A}(y)$. We use the same notation for expanding the entries of tuples, for example, we write $\sigma := (\sigma_1, \sigma_2)$ for a tuple σ composed of two elements. A function $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}$ is *negligible* in n if for every $k \in \mathbb{N}$, there exists $n_0 \in \mathbb{N}$, such that for every $n \geq n_0$ it holds that $|\mathsf{negl}(n)| \leq 1/n^k$. Throughout the paper we implicitly assume that negligible functions are negligible in the security parameter (i.e., $\mathsf{negl}(\lambda)$).

A. Cryptographic Primitives

Next, we review here the cryptographic primitives used in our protocols.

Commitment scheme. A commitment scheme COM consists of PPT algorithms $\mathsf{COM} = (\mathsf{P_{COM}}, \mathsf{V_{COM}})$, where $\mathsf{P_{COM}}$ is the commitment algorithm, such that $(\mathsf{com}, \mathsf{decom}) \leftarrow \mathsf{P_{COM}}(m)$, and $\mathsf{V_{COM}}$ is the verification algorithm, such that $\{0,1\} := \mathsf{V_{COM}}(\mathsf{com}, \mathsf{decom}, m)$. A COM scheme allows a prover to commit to a message m without revealing it, and convince a verifier, using commitment com and decommitment information decom, that the message m was committed. The security of COM is modeled by the ideal functionality $\mathcal{F}_{\mathsf{COM}}$ [31], as described in Appendix F2. In our protocols we use the Pedersen commitment scheme [35], which is an information-theoretically (unconditionally) hiding and computationally binding commitment scheme.

Non-interactive zero-knowledge. Let R be an NP relation, and let L be a set of positive instances corresponding to the relation R (i.e., $L = \{x \mid \exists w \text{ s.t. } R(x, w) = 1\}$). We say R is a *hard relation* if R is poly-time decidable, there exists a PPT instant sampling function GenR and for all PPT adversaries $\mathcal{A}$, the probability of $\mathcal{A}$ producing the witness w given only the statement x, such that $R(x, w) = 1$, is bounded by a negligible function. This is more formally defined in Appendix D. A non-interactive zero-knowledge proof scheme NIZK [36] consists of PPT algorithms $\mathsf{NIZK} = (\mathsf{P}_{\mathsf{NIZK}}, \mathsf{V}_{\mathsf{NIZK}})$, where $\mathsf{P}_{\mathsf{NIZK}}$ is the prover algorithm, such that $\pi \leftarrow \mathsf{P}_{\mathsf{NIZK}}(x, w)$, and $\mathsf{V}_{\mathsf{NIZK}}$ is the verification algorithm, such that $\{0, 1\} := \mathsf{V}_{\mathsf{NIZK}}(x, \pi)$. A NIZK scheme allows a prover to convince a verifier, using a proof π, about the existence of a witness w for a statement x without revealing any information apart from the fact that it knows the witness w. We model the security of a NIZK scheme using the ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$, defined in Appendix F2.

Homomorphic encryption scheme. A public key encryption scheme Ψ with a message space $\mathcal{M}$ is composed of PPT algorithms $\Psi = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$, such that for every $m \in \mathcal{M}$, it holds that $\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m \mid (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(1^\lambda)] = 1$, for a security parameter 1^λ and a secret/public key pair $(\mathsf{sk}, \mathsf{pk})$. We say that Ψ is additively homomorphic if it supports homomorphic operations over the ciphertexts. More precisely, for every $m_1, m_2 \in \mathcal{M}$ and public key pk, we have that $\mathsf{Enc}(\mathsf{pk}, m_1) \cdot \mathsf{Enc}(\mathsf{pk}, m_2) = \mathsf{Enc}(\mathsf{pk}, m_1 + m_2)$. Furthermore, we assume that the operation $\mathsf{Enc}(\mathsf{pk}, m_1)^{m_2}$ is well-defined, and yields $\mathsf{Enc}(\mathsf{pk}, m_1 \cdot m_2)$. Homomorphic encryption schemes need to satisfy the security notion of indistinguishability under chosen plaintext attack (IND-CPA), which at a high level guarantees that a PPT adversary $\mathcal{A}$ is not able to distinguish the encryption of two messages of its choice. In our construction we use the additively homomorphic encryption scheme by Castagnos-Laguillaumie (CL) [37] (more precisely, HSM-CL described in [38]), where $\mathcal{M} = \mathbb{Z}_q$. The reason for this is that we can instantiate CL to work with any $\mathbb{Z}_q$ for a prime q that is compatible with the underlying signature scheme that we make use of. For more information regarding why we chose the CL encryption scheme we refer the reader to Appendix E. Additionally, we assume existence of a function $\mathsf{RandCtx}$, which given as input a ciphertext c, returns the ciphertext c' randomized through homomorphic multiplication operation and the randomization factor r used in the process. More precisely, given $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$, the randomization process produces $(c', r) \leftarrow \mathsf{RandCtx}(c)$, where r is the randomization factor, and c' is encryption of $m \cdot r$. This operation is supported in CL encryption scheme through homomorphic multiplication with plaintext (in this case the plaintext is the randomization factor r).

Digital signature scheme. A digital signature scheme Σ with a message space $\mathcal{M}$ is composed of PPT algorithms $\Sigma = (\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$, such that for every $m \in \mathcal{M}$, it holds that $\Pr[\mathsf{Vf}(\mathsf{pk}, \mathsf{Sig}(\mathsf{sk}, m), m) = 1 \mid (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(1^\lambda)] = 1$, for a security parameter 1^λ and a secret/public key pair $(\mathsf{sk}, \mathsf{pk})$. The most common security requirement of a signature scheme is existential unforgeability under chosen message attack (EUF-CMA). At a high level, it ensures that a PPT adversary $\mathcal{A}$, that does not know the secret key sk, cannot produce a valid signature σ on a message m even if it sees polynomially many valid signatures on messages of its choice (but different from m).

Adaptor signature scheme. An adaptor signature scheme is defined with respect to a hard relation R and a signature scheme Σ and consists of four algorithms $\Xi_{R,\Sigma} = (\mathsf{PreSig}, \mathsf{PreVf}, \mathsf{Adapt}, \mathsf{Ext})$. For every statement/witness pair $(Y, y) \in R$, secret/public key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \Sigma.\mathsf{KGen}(1^\lambda)$ and a message $m \in \mathcal{M}$, we have that $\hat{\sigma} \leftarrow \mathsf{PreSig}(\mathsf{sk}, m, Y)$ is a *pre-signature* and $\sigma := \mathsf{Adapt}(\hat{\sigma}, y)$ is a valid signature, and (pre-)verification holds under pk for $\hat{\sigma}$ and σ, respectively. Furthermore, it holds that $y := \mathsf{Ext}(\sigma, \hat{\sigma}, Y)$. Adaptor signatures were formally defined in [29], where the authors also defined the security notion called the existential unforgeability under chosen message attack for adaptor signature (aEUF-CMA). Apart from aEUF-CMA security, an adaptor signature should also provide pre-signature correctness, pre-signature adaptability and witness extractability. Roughly speaking, pre-signature correctness ensures that an honestly generated pre-signature $\hat{\sigma}$ w.r.t a statement Y is a valid pre-signature and can be completed into a valid signature σ, from which a witness of Y can be extracted. On the other hand, pre-signature adaptability means that the pre-signature $\hat{\sigma}$ can be adapted into a valid signature σ using the witness y. Lastly, witness extractability guarantees that a valid signature/pre-signature pair $(\sigma, \hat{\sigma})$ can be used to extract the corresponding witness y of Y. We refer the reader to Appendix D for a more formal and detailed treatment of adaptor signatures. Lastly, we point out that the authors of [29] additionally gave provably secure Schnorr- and ECDSA-based instantiations of adaptor signatures.

(Blinded) Randomizable signature scheme. Furthermore, we need a signature scheme which can be randomizable, and that enables the issuance of a signature on a committed value, which can be seen as a type of a blinded signature. More precisely, we call here a signature scheme $\widetilde{\Sigma}$ a blinded randomizable signature scheme, if it provides three additional PPT algorithms, namely, $\mathsf{BlindSig}, \mathsf{UnblindSig}$ and $\mathsf{RandSig}$, in addition to the ones provided by a regular signature scheme Σ. Given a commitment com to a message m, the signer can generate a blinded signature $\sigma^* \leftarrow \mathsf{BlindSig}(\mathsf{sk}, \mathsf{com})$. Then, the party holding the decommitment information decom, can unblind σ^* to produce a valid signature $\sigma := \mathsf{UnblindSig}(\mathsf{decom}, \sigma^*)$. Lastly, given a valid signature σ, one can generate a randomized signature as $\sigma' \leftarrow \mathsf{RandSig}(\sigma)$. A signature scheme that provides these features and which we use in our construction is Pointcheval-Sanders (PS) [39] signature scheme, which works over Type-III bilinear groups.

V. Randomizable Puzzles

In order to ease the exposition here we define a primitive called *randomizable puzzle*, which we later use to construct

A^{2}L, and that also captures the puzzle used in [23].

A. Definitions

Definition 1 (Randomizable Puzzle)**.** *A randomizable puzzle scheme* $\mathsf{RP} = (\mathsf{PSetup}, \mathsf{PGen}, \mathsf{PSolve}, \mathsf{PRand})$ *with a solution space* $\mathcal{S}$ *(and a function* ϕ *acting on* $\mathcal{S}$*) consists of four algorithms defined as:*

$(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{PSetup}(1^\lambda)$**:** *is a* PPT *algorithm that on input a security parameter* 1^λ*, outputs public parameters* pp *and a trapdoor* td.

$Z \leftarrow \mathsf{PGen}(\mathsf{pp}, \zeta)$**:** *is a* PPT *algorithm that on input public parameters* pp *and a puzzle solution* ζ*, outputs a puzzle* Z.

$\zeta := \mathsf{PSolve}(\mathsf{td}, Z)$**:** *is a* DPT *algorithm that on input a trapdoor* td *and puzzle* Z*, outputs a puzzle solution* ζ.

$(Z', r) \leftarrow \mathsf{PRand}(\mathsf{pp}, Z)$**:** *is a* PPT *algorithm that on input public parameters* pp *and a puzzle* Z *(which has a solution* ζ*), outputs a randomization factor* r *and a randomized puzzle* Z' *(which has a solution* $\phi(\zeta, r)$*).*

We assume that the solution space $\mathcal{S}$ has an algebraic structure, such that it is easy to (de-)randomize a solution in a way that it stays within $\mathcal{S}$. More precisely, we assume existence of a deterministic function $\phi(\cdot,\cdot)$, such that for a puzzle Z with the solution ζ and its randomized version Z' with the randomization factor r, we have that $\phi(\zeta, r) \in \mathcal{S}$ is a solution to Z'. For example, in our construction described in Section V-B, the solution space $\mathcal{S}$ is the field $\mathbb{Z}_q$ and ϕ is the multiplication operation, such that $\phi(a, b) := a \cdot b \bmod q$.

Furthermore, note that we do not impose any restrictions on the input puzzle Z to the randomization algorithm PRand. Hence, the input can be a freshly generated puzzle, or a puzzle that was previously randomized. This allows us to capture multiple randomizations of a puzzle.

We require that a randomizable puzzle (RP) satisfies correctness, security and privacy properties. Correctness property ensures that using the trapdoor we can always recover the correct solution to the puzzle (where a randomized puzzle's solution depends on the randomization factor).

Definition 2 (Correctness)**.** *For all* $\lambda \in \mathbb{N}$*,* $n = \mathsf{poly}(\lambda)$*, pair* $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{PSetup}(1^\lambda)$*, for every* $\zeta^{(1)} \in \mathcal{S}$ *and* $1 \le i \le n$*, we have that*

$$\Pr[\mathsf{PSolve}(\mathsf{td}, Z^{(i)}) = \zeta^{(i)}] = 1,$$

where $Z^{(1)} \leftarrow \mathsf{PGen}(\mathsf{pp}, \zeta^{(1)})$*, and for* $2 \le i \le n$*,* $(Z^{(i)}, r^{(i)}) \leftarrow \mathsf{PRand}(\mathsf{pp}, Z^{(i-1)})$ *and* $\zeta^{(i)} := \phi(\zeta^{(i-1)}, r^{(i)})$*.*

We say that a RP scheme is secure if it is infeasible for an adversary that has access only to the puzzle and the public parameters to obtain the underlying solution.

Definition 3 (Security)**.** *A randomizable puzzle scheme* RP *is secure, if there exists a negligible function* $\mathsf{negl}(\cdot)$*, such that*

$$\Pr\left[\zeta \leftarrow \mathcal{A}(\mathsf{pp}, Z) \;\middle|\; \begin{array}{l}(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{PSetup}(1^\lambda)\\ \zeta \leftarrow_{\$} \mathcal{S}, Z \leftarrow \mathsf{PGen}(\mathsf{pp}, \zeta)\end{array}\right] \le \mathsf{negl}(\lambda).$$

Lastly, we define the privacy property using a cryptographic game between the challenger and adversary. The adversary provides two puzzles that are correctly formed, then the challenger randomizes one of the puzzles and returns it to the adversary. The goal of the adversary is to find out which of the two puzzles was randomized. We say that a RP scheme is private if the adversary cannot do any better than guessing even when it has access to the trapdoor. Although, this definition seems rather strong, what it actually ensures is that the privacy is retained as long as the randomization factor used during the randomization procedure stays hidden, which is a natural requirement.

Definition 4 (Privacy)**.** *A randomizable puzzle scheme* RP *is private if for every* PPT *adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:* $\Pr[\mathsf{RPRandSec}_{\mathcal{A},\mathsf{RP}}(\lambda) = 1] \le 1/2 + \mathsf{negl}(\lambda)$*, where the experiment* $\mathsf{RPRandSec}_{\mathcal{A},\mathsf{RP}}$ *is defined as follows:*

$\mathsf{RPRandSec}_{\mathcal{A},\mathsf{RP}}(\lambda)$

1 : $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{PSetup}(1^\lambda)$
2 : $((Z_0, \zeta_0), (Z_1, \zeta_1)) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{td})$
3 : $b \leftarrow_{\$} \{0, 1\}$
4 : $(Z'_0, r_0) \leftarrow \mathsf{PRand}(\mathsf{pp}, Z_0)$
5 : $(Z'_1, r_1) \leftarrow \mathsf{PRand}(\mathsf{pp}, Z_1)$
6 : $b' \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{td}, Z'_b)$
7 : **return** $\mathsf{PSolve}(\mathsf{td}, Z_0) = \zeta_0 \wedge \mathsf{PSolve}(\mathsf{td}, Z_1) = \zeta_1$
8 : $\quad \wedge\, b = b'$

Discussion. We note that our RP primitive also captures the puzzle construction of TumbleBit [23]. More concretely, in case of TumbleBit we have that $\mathcal{S} = \mathbb{Z}_N$, for a composite $N = p \cdot q$ (i.e., a strong RSA integer). A puzzle Z is defined as the RSA encryption of the solution ζ, and randomization of a puzzle is defined by blinding the encrypted solution with a random value using the homomorphic properties of RSA. Both the RP from TumbleBit [23] and ours from Section V-B rely on an encryption scheme with homomorphic properties. This can be seen as a rather generic way to construct a RP scheme. We leave it as an open problem to devise a RP scheme that relies on other cryptographic primitives.

B. Cryptographic Construction

Next, we describe a construction of a RP scheme. Our construction can be generically instantiated over a group $\mathbb{G}$ where the discrete logarithm (DLOG) problem is assumed to be hard, and with an additively homomorphic encryption scheme Ψ. In this work we set the group $\mathbb{G}$ to be an elliptic curve group of order q, and set the encryption scheme Ψ to Castagnos-Laguillaumie (CL) [37] encryption scheme with a message space $\mathcal{M} = \mathbb{Z}_q$ (as described in Section IV-A). Hence, this instantiation has a solution space $\mathcal{S} = \mathbb{Z}_q$. Our construction can be seen in Construction 1. As it can be observed in our construction, the puzzle includes a group element, however, that element is not used at all during the solution of the puzzle (i.e., in PSolve algorithm). We note that this is specific to our scenario, and we need it to link the puzzle to our conditional payment, which uses adaptor signature over the same group $\mathbb{G}$.

Construction 1. *We assume existence of group description parameters* $\mathsf{gp} = (\mathbb{G}, g, q)$.

$\mathsf{PSetup}(1^\lambda)$: *sample a key pair* $(\mathsf{sk}^\Psi, \mathsf{pk}^\Psi) \leftarrow \mathsf{KGen}(1^\lambda)$, *set* $\mathsf{pp} := (\mathsf{gp}, \mathsf{pk}^\Psi)$ *and* $\mathsf{td} := \mathsf{sk}^\Psi$, *and return* $(\mathsf{pp}, \mathsf{td})$.

$\mathsf{PGen}(\mathsf{pp}, \zeta)$: *parse* pp *as* $(\mathsf{gp}, \mathsf{pk}^\Psi)$, *compute* $A = g^\zeta$ *and* $c \leftarrow \mathsf{Enc}(\mathsf{pk}^\Psi, \zeta)$, *and return* $Z := (A, c)$.

$\mathsf{PSolve}(\mathsf{td}, Z)$: *parse* td *as* sk^Ψ *and* Z *as* (A, c), *compute* $\zeta \leftarrow \mathsf{Dec}(\mathsf{sk}^\Psi, c)$, *and return* ζ.

$\mathsf{PRand}(\mathsf{pp}, Z)$: *parse* Z *as* (A, ζ), *sample* $r \leftarrow_{\$} \mathcal{S}$, *compute* $A' = A^r$ *and* $c' = c^r$, *set* $Z' := (A', c')$, *and return* (Z', r).

The security of our construction is established with the following theorem, for which we provide a proof sketch in Appendix F1.

Theorem 1. *Let* $\mathbb{G}$ *be a DLOG-hard group, and* Ψ *be an* IND-CPA *secure encryption scheme, then Construction 1 is a correct, secure and private randomizable puzzle scheme.*

VI. Our Protocols

System assumptions. We assume a constant amount of coins (i.e., amt) for every payment, as otherwise it becomes trivial to link P_s and P_r in a payment. Moreover, as in TumbleBit [23], we assume that the protocols are run in phases and epochs. Each epoch is composed of three phases for us: (i) registration phase (see Section VI-B) (i) puzzle promise phase (escrow phase in TumbleBit), and (iii) puzzle solver phase (payment phase in TumbleBit). In each epoch, instances of our protocols are executed in their corresponding phases (e.g., puzzle promise protocol is executed during the puzzle promise phase), optimizing thereby the anonymity set within an epoch.

Here we further assume that both the sender P_s and the receiver P_r have already carried out the key generation procedure and have set up the payment channels with the tumbler P_t. We finally assume that communication between honest P_s and P_r is unnoticed by P_t, which is a common assumption in other privacy-preserving PCH constructions [23]. We stress that we only need this anonymous communication between the sender and receiver when exchanging the randomizable puzzle and its solution.

A. Anonymous Atomic Locks (A^2L)

In this section, we describe the puzzle promise and puzzle solver phases. These phases are independent on the registration phase, which we describe later in Section VI-B.

Puzzle promise. The puzzle promise protocol (and subsequently, the puzzle solver protocol) relies on an adaptor signature scheme $\Xi_{R,\Sigma}$ for a hard relation R and a signature scheme Σ. This protocol starts with P_r sending to P_t its own valid signature σ'_r on a previously agreed message m', which is the agreed transaction (lines 2-3 in Figure 2).

Next in the protocol, P_t samples a statement/witness pair (A, α), for a statement $A := g^\alpha$ (DLOG), generates the randomizable puzzle $Z := (A := g^\alpha, c_\alpha)$ using the PGen algorithm, and produces a NIZK proof π_α proving that α is a valid solution to puzzle Z (lines 6-7 in Figure 2). The proof π_α in our instantiation of randomizable puzzles (see Section V-B) can be interpreted as $\pi_\alpha \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists \alpha \mid c_\alpha = \mathsf{Enc}(\mathsf{pk}_t^\Psi, \alpha) \wedge A = g^\alpha\}, \alpha)$. Additionally, P_t generates an adaptor signature $\hat{\sigma}'_t$ over the previously agreed message (transaction) m', where the secret adaptor is α, and shares the puzzle Z along with the adaptor signature $\hat{\sigma}'_t$ with P_r (lines 8-9 in Figure 2). At this point P_r cannot claim the coins, because the signature $\hat{\sigma}'_t$ is not valid, however, P_r can pre-verify it by relying on the pre-signature correctness property of the adaptor signature (line 11 in Figure 2).

Once P_r is convinced of the validity of $\hat{\sigma}'_t$, it randomizes the puzzle Z using the PRand algorithm to obtain the puzzle Z', which it shares with P_s (lines 12-13 in Figure 2). This finalizes the puzzle promise protocol, and allows P_s to start the puzzle solver protocol with P_t, as shown in Figure 3.

We note that the blue parts in Figure 2 correspond to the additional operations needed to protect against the griefing attack described in Section VI-B. Roughly speaking, P_r provides P_t with a token (which it previously obtains from P_s), and P_t verifies the validity and freshness of the token before starting the puzzle promise protocol. We refer the reader to Section VI-B for further details.

Puzzle solver. In the puzzle solver protocol, P_s also randomizes the puzzle Z', that it receives from P_r, into Z'' (line 2 in Figure 3) to preserve its own anonymity and thwart attacks involving collusion of P_t and P_r (see Appendix A1). Then, P_s gives an adaptor signature $\hat{\sigma}_s$ to P_t (lines 3-4 in Figure 3), which is adapted with the newly randomized puzzle Z''. Since P_t has the trapdoor td of the randomizable puzzle scheme, it can solve the puzzle Z'' to obtain the doubly randomized version of the value α (i.e., the secret value required by P_r to complete the adaptor signature $\hat{\sigma}'_t$ from puzzle promise). As α'' is randomized, P_t cannot link it to P_r and yet can use it to convert $\hat{\sigma}_s$ into a valid signature σ_s by adapting it with α'' (line 5 in Figure 3).

All that remains for P_t in order to get paid is to compute its own signature σ_t on a previously agreed upon message (transaction) m, and update the channel with the signature pair (σ_s, σ_t) (lines 6-8 in Figure 3). Once P_t provides P_s with this signature and gets paid, P_s can extract α'' using the adaptor signature $\hat{\sigma}_s$ and the valid signature σ_s. Then, P_s gets rid of one layer of the randomization to obtain α', which it shares with P_r (lines 10-12 in Figure 3). Finally, P_r removes its part of the randomness from α', and thereby gets the original value α, which it uses to adapt the "almost valid" signature $\hat{\sigma}'_t$ into a fully valid one σ'_t, as shown inside the Open algorithm in Figure 4.

1) Discussion: Our protocol achieves interoperability with virtually all current cryptocurrencies. The interoperability is achieved due to the minimal cryptographic requirements of our construction from the underlying cryptocurrency. More precisely, we only require a digital signature that can be turned into an adaptor signature, and a timelock mechanism from

Public parameters: group description $(\mathbb{G}, g, q)$, message m'

$\mathsf{PuzzlePromise}_{P_t}((\mathsf{sk}^\Sigma_t, \mathsf{pk}^\Sigma_t), (\mathsf{pp}, \mathsf{td}), \mathsf{pk}^\Sigma_r, \mathsf{pk}^{\tilde{\Sigma}}_t)$		$\mathsf{PuzzlePromise}_{P_r}((\mathsf{sk}^\Sigma_r, \mathsf{pk}^\Sigma_r), \mathsf{pk}^\Sigma_t, \mathsf{pp}, (\mathsf{tid}, \sigma'_{\mathsf{tid}}))$
		$\sigma'_r \leftarrow \mathsf{Sig}(\mathsf{sk}^\Sigma_r, m')$
	$\xleftarrow{(\mathsf{tid}, \sigma'_{\mathsf{tid}}), \sigma'_r}$	
If $\mathsf{tid} \in \mathcal{T} \vee \mathsf{Vf}(\mathsf{pk}^{\tilde{\Sigma}}_t, \mathsf{tid}, \sigma'_{\mathsf{tid}}) \neq 1$ then abort		
Else add tid into $\mathcal{T}$		
$(A, \alpha) \leftarrow \mathsf{GenR}(1^\lambda)$; $Z \leftarrow \mathsf{PGen}(\mathsf{pp}, \alpha)$		
$\pi_\alpha \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists \alpha \mid \mathsf{PSolve}(\mathsf{td}, Z) = \alpha\}, \alpha)$		
$\hat{\sigma}'_t \leftarrow \mathsf{PreSig}(\mathsf{sk}^\Sigma_t, m', A)$		
	$\xrightarrow{Z := (A, c_\alpha), \pi_\alpha, \hat{\sigma}'_t}$	
		If $\mathsf{V}_{\mathsf{NIZK}}(\pi_\alpha, Z) \neq 1$ then abort
		If $\mathsf{PreVf}(\mathsf{pk}^\Sigma_t, m', A, \hat{\sigma}'_t) \neq 1$ then abort
		$(Z', \beta) \leftarrow \mathsf{PRand}(\mathsf{pp}, Z)$
		Send $Z' := (A', c'_\alpha)$ to P_s
		Set $\Pi := (\beta, (\mathsf{pk}^\Sigma_t, \mathsf{pk}^\Sigma_r), m', (\hat{\sigma}'_t, \sigma'_r))$
return $(\mathsf{Adapt}(\hat{\sigma}'_t, \alpha), \sigma_r)$		**return** $(\Pi, (Z, Z'))$

Fig. 2: Puzzle promise protocol of A^{2}L. Blue parts are related to the griefing protection (see Section VI-B).

Public parameters: group description $(\mathbb{G}, g, q)$, message m

$\mathsf{PuzzleSolver}_{P_s}((\mathsf{sk}^\Sigma_s, \mathsf{pk}^\Sigma_s), \mathsf{pp}, Z' := (A', c'_\alpha))$		$\mathsf{PuzzleSolver}_{P_t}((\mathsf{sk}^\Sigma_t, \mathsf{pk}^\Sigma_t), (\mathsf{pp}, \mathsf{td}), \mathsf{pk}^\Sigma_s)$
$(Z'' := (A'', c''_\alpha), \tau) \leftarrow \mathsf{PRand}(\mathsf{pp}, Z')$		
$\hat{\sigma}_s \leftarrow \mathsf{PreSig}(\mathsf{sk}^\Sigma_s, m, A'')$		
	$\xrightarrow{Z'', \hat{\sigma}_s}$	
		$\alpha'' := \mathsf{PSolve}(\mathsf{td}, Z'')$; $\sigma_s := \mathsf{Adapt}(\hat{\sigma}_s, \alpha'')$
		$\sigma_t \leftarrow \mathsf{Sig}(\mathsf{sk}^\Sigma_t, m)$
		If $\mathsf{Vf}(\mathsf{pk}^\Sigma_s, m, \sigma_s) \neq 1$ then abort
		Else publish (σ_s, σ_t)
	$\xleftarrow{\sigma_s}$	
$\alpha'' := \mathsf{Ext}(\sigma_s, \hat{\sigma}_s, A'')$		
If $\alpha'' = \perp$ then abort		
Else $\alpha' \leftarrow \alpha'' \cdot \tau^{-1}$ and send α' to P_r		
return α'		**return** $\top$

Fig. 3: Puzzle solver protocol of A^{2}L.

$\mathsf{Open}(\Pi, \alpha')$

Parse Π as $(\beta, (\mathsf{pk}^\Sigma_t, \mathsf{pk}^\Sigma_r), m', (\hat{\sigma}'_t, \sigma'_r))$
$\alpha \leftarrow \alpha' \cdot \beta^{-1}$
$\sigma'_t := \mathsf{Adapt}(\hat{\sigma}'_t, \alpha)$
return (σ'_t, σ'_r)

$\mathsf{Verify}(\Pi, \sigma)$

Parse Π as $(\beta, (\mathsf{pk}^\Sigma_1, \mathsf{pk}^\Sigma_2), m', (\sigma'_1, \sigma'_2))$
Parse σ as (σ_1, σ_2)
return $\mathsf{Vf}(\mathsf{pk}^\Sigma_1, m', \sigma_1) \wedge \mathsf{Vf}(\mathsf{pk}^\Sigma_2, m', \sigma_2)$

Fig. 4: Open and verify algorithms of A^{2}L.

the underlying cryptocurrency, two functionalities provided by virtually all cryptocurrencies today. As a matter of fact, we can also adapt our approach to cryptocurrencies that totally lack a scripting language support for 2-of-2 multisignatures, such as Ripple, Stellar or Mimblewimble following the threshold version of adaptor signatures [40]. We describe in Appendix G how to use A^{2}L with threshold signatures where the output of our protocols will result in accepting a channel update with a single signature (instead of a 2-of-2 multisignature) verifiable by a single public key.

Furthermore, our protocol opens the door to mediate payments in different cryptocurrencies, by running the puzzle promise and puzzle solver protocols in different cryptocurrencies. For example, this can be achieved by instantiating our construction with adaptor signatures that work over the same group, and using one signature scheme for the puzzle promise phase, and the other one for the puzzle solver phase [40], thereby enabling cross-chain applications like exchanges. Moreover, even when the groups are not the same we can still use this technique, assuming there exists an efficiently computable bijection between the two groups, and utilizing the proof for discrete logarithm equality across groups described in [41]. We discuss further deployment aspects for cross-chain payments in Appendix A.

B. Extension: Registration Protocol

We describe a protocol called *registration* protocol, which is used to defend against the griefing attack as explained in

Section III. Although, our registration protocol is an extension of A^{2}L, it is rather generic, and can be used with other constructions that require protection against similar type of griefing attack (e.g., TumbleBit [23]). The registration protocol is executed between the sender P_s and the tumbler P_t, and assumes that P_s has locked coins with P_t in 2-of-2 escrow output (oid) before the start of the protocol. The registration protocol can be seen in Figure 5.

Our protocol is inspired from anonymous credentials [42], however, contrary to the anonymous credentials where the issued credentials can be used multiple times, we need to ensure that the issued credential (token) is used only once. Furthermore, the party issuing and authenticating the tokens in our case is the same party, i.e., the tumbler P_t, whereas in anonymous credentials the issuance and authentication of the credentials might be done by different parties. Hence, instead of anonymous credentials we have opted for a simpler but more efficient protocol that is backwards compatible with current cryptocurrencies.

Our registration protocol makes use of a (blinded) randomizable signature scheme $\widetilde{\Sigma}$ as described in Section IV-A, which we instantiate with Pointcheval-Sanders (PS) [39], a commitment scheme, for which we use Pedersen commitment [35], and a NIZK proof of knowledge (PoK) for opening of a Pedersen commitment.

At the beginning of the protocol P_s generates a random token identifier tid and a commitment com to tid using Pedersen commitment, along with a NIZK proof π for the opening of the commitment, and sends the pair (π, com) and the escrow output oid to P_t (lines 2-5 in Figure 5). P_t verifies the proof π, and then (blindly) generates a signature σ^* on the token tid using the commitment com, and sends σ^* to P_s (lines 6-8 in Figure 5). Here, it is important that tid is hidden (i.e., inside a commitment), otherwise, P_t can trivially link the sender P_s and the corresponding receiver P_r. The reason for this is that the puzzle promise protocol (see Figure 2) starts with the receiver P_r sharing this tid in the clear with the tumbler P_t as a form of validation (i.e., that there already exists a payment promised to P_t). This is also the reason why we require a signature scheme that allows to (blindly) sign a value hidden inside a commitment (such as Pointcheval-Sanders [39] signature scheme).

Next, P_s unblinds σ^* using the decommitment information decom to obtain a valid signature σ_{tid} on the token tid (line 9 in Figure 5). Lastly, P_s randomizes σ_{tid} to obtain σ'_{tid} and sends the pair $(\mathsf{tid}, \sigma'_{\mathsf{tid}})$ to the receiver P_r (lines 11-12 in Figure 5), which finalizes the registration protocol. We note that PS signature scheme is composed of two group elements, and unblinding operation only affects the second component of the signature, hence, we have that the first components of both σ_{tid} and σ'_{tid} are the same. Therefore, if P_r gives σ_{tid} to P_t at the beginning of the puzzle promise protocol for validation, then P_t can trivially link P_s and P_t. This is the reason why we randomize σ_{tid} and only share the randomized signature σ'_{tid} with P_t. This randomization can be done either by P_s or P_r before the start of the puzzle promise protocol (in Figure 5 it is randomized by P_s as part of the registration protocol).

Although, this concludes the registration procedure, as previously mentioned, the token/signature pair need to be presented to P_t at the start of the puzzle promise protocol (see Figure 2). P_t checks that the token tid has not been previously used, in order to be protected against replay attacks (i.e., P_r tries to claim the same collateral locked by P_s more than once). For this reason P_t has to keep a list $\mathcal{T}$ with all the previously seen token identifiers. We note that since we expect our protocols to run in epochs (see Section VI) we can reduce the storage requirement of P_t by letting it generate a new key pair, publish the new $\mathsf{pk}_t^{\widetilde{\Sigma}}$ so that it is available to others, and then reset the list $\mathcal{T}$ at the beginning of each epoch. Hence, from that point onward all the tokens signed with the secret key of the previous epoch become invalid from the perspective of P_t.

C. Our PCH Instantiation

We realize a PCH by setting channel updates and timelock mechanism for the payment agnostic A^{2}L. In particular:

1) **Collateral setup**: Before the registration phase of A^{2}L starts, P_s updates its channel with P_t to create a escrow for the duration of the rest of the protocol between P_s and P_t, represented by oid. This escrow locks amt coins from the balance of P_s into oid. This oid plays two roles: (i) since P_s does not authorize the spending of oid, P_s ensures that she can recover the amt coins locked there after the timeout expires; and (ii) since P_t does not authorize the spending of oid either, P_t ensures that the amt coins locked there cannot be reused before the timeout, effectively serving as a proof of collateral for the rest of the PCH protocol.

2) **Payment channel update proposals**: Before the puzzle promise phase of A^{2}L starts, P_t updates its channel with P_r to propose a payment where amt coins are transferred from the balance of P_t to the balance of P_r. The authorization of this channel update is then handled by A^{2}L. A similar payment for amt coins is proposed in the channel between P_s and P_t before the puzzle solver phase of A^{2}L is initiated. We note that both payments have associated an expiration time so that if A^{2}L is not successful (e.g., one of the parties does not collaborate), the payments are deemed invalid and the coins return to the original owners.

3) **Payment channel update resolutions**: After both puzzle promise and puzzle solver have finished, the channel updates proposed in the previous step are finalized. There could be two outcomes. On the one hand, if both puzzle promise and puzzle solver are successful, the PCH first updates the channel between P_s and P_t. Afterwards, P_r can finalize the authorization of the update of this channel with P_t and accordingly reflect the payment. On the other hand, if any of puzzle promise or puzzle solver fails, then both payment proposals are expired, leaving balances at both channels as before the start of the execution of the payment.

4) **Collateral release**: At the end of the protocol, independently of the outcome of the previous phases, the coins locked

Public parameters: bilinear groups description $(q, e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, g_T)$

$\mathsf{Registration}_{P_s}(\mathsf{pk}_t^{\widetilde{\Sigma}}, \mathsf{oid})$		$\mathsf{Registration}_{P_t}((\mathsf{sk}_t^{\widetilde{\Sigma}}, \mathsf{pk}_t^{\widetilde{\Sigma}}))$
Sample a token $\mathsf{tid} \leftarrow\$ \mathbb{Z}_q$		
$(\mathsf{com}, \mathsf{decom} := (\mathsf{tid}, r)) \leftarrow \mathsf{P_{COM}}(\mathsf{tid})$		
$\pi \leftarrow \mathsf{P_{NIZK}}(\{\exists \mathsf{decom} \mid \mathsf{V_{COM}}(\mathsf{com}, \mathsf{decom}, \mathsf{tid}) = 1\}, \mathsf{decom})$		
	$\xrightarrow{(\pi, \mathsf{com}), \mathsf{oid}}$	
		If $\mathsf{V_{NIZK}}(\pi, \mathsf{com}) \neq 1$ then abort
		$\sigma^* \leftarrow \mathsf{BlindSig}(\mathsf{sk}_t^{\widetilde{\Sigma}}, \mathsf{com})$
	$\xleftarrow{\sigma^*}$	
$\sigma_{\mathsf{tid}} := \mathsf{UnblindSig}(\mathsf{decom}, \sigma^*)$		
If $\mathsf{Vf}(\mathsf{pk}_t^{\widetilde{\Sigma}}, \mathsf{tid}, \sigma_{\mathsf{tid}}) \neq 1$ then abort		
$\sigma'_{\mathsf{tid}} \leftarrow \mathsf{RandSig}(\sigma_{\mathsf{tid}})$		
Send $(\mathsf{tid}, \sigma'_{\mathsf{tid}})$ to P_r		
return $(\mathsf{tid}, \sigma'_{\mathsf{tid}})$		**return** $\top$

Fig. 5: Registration protocol for griefing protection. Blue part is related to the payment (i.e., non-cryptographic operation).

by P_s in oid at the beginning of the payment are released and set back to P_s.

For a full description of our PCH construction we refer the reader to Appendix B.

VII. Security Analysis

We formalize the security and privacy of A^{2}L and our PCH construction in the universal composability (UC) framework [31]. We rely on the synchronous version of global UC framework (GUC) [43]. We prove the security in the UC framework because unlike the standalone simulation-based or game-based proofs, it allows for a concurrent composition of protocols. This imples that a protocol remains secure when many instances are executed concurrently (with arbitrary other protocols), possibly on related inputs.

Here we describe the high level ideas of our security analysis in the UC framework, and refer the reader to Appendix C for more details about our security model. First, we define an ideal functionality $\mathcal{F}_{\mathsf{A^2L}}$ capturing the ideal behavior of our A^{2}L construction. $\mathcal{F}_{\mathsf{A^2L}}$ specifies the input/output behavior of A^{2}L protocols, and the possible influence of an adversary on the execution. Next, we show that our A^{2}L construction emulates $\mathcal{F}_{\mathsf{A^2L}}$. Roughly speaking, this means that our construction is at least as secure as $\mathcal{F}_{\mathsf{A^2L}}$ itself, and any attack that can be performed on our protocols can be simulated as an attack on $\mathcal{F}_{\mathsf{A^2L}}$.

The description of our ideal functionality $\mathcal{F}_{\mathsf{A^2L}}$ (along with its hybrid ideal functionalities) can be found in Appendix C. In Appendix F2, we formally prove the following theorem.

Theorem 2. *Let* COM *be a secure commitment scheme,* NIZK *be a non-interactive zero-knowledge scheme,* $\Sigma, \widetilde{\Sigma}$ *be* EUF-CMA *secure signature schemes,* R *be a hard relation,* $\Xi_{R,\Sigma}$ *be a secure adaptor signature scheme, and* RP *be a secure and private randomizable puzzle scheme, then the construction in Figures 2 to 5 UC-realizes the ideal functionality* $\mathcal{F}_{\mathsf{A^2L}}$ *in the* $(\mathcal{F}_{\mathsf{GDC}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{anon}})$*-hybrid model.*

Moreover, we define an ideal functionality $\mathcal{F}_{\mathsf{PCH}}$ describing the ideal behavior of our PCH construction. Similar to the proof of emulation of $\mathcal{F}_{\mathsf{A^2L}}$, we prove indistinguishability between the real and ideal world. More precisely, for $\mathcal{F}_{\mathsf{PCH}}$ described in Appendix C, we prove the following theorem in Appendix F3.

Theorem 3. *The protocol in Figure 6, UC-realizes* $\mathcal{F}_{\mathsf{PCH}}$ *in the* $(\mathcal{F}_{\mathsf{GDC}}, \mathcal{F}_{\mathsf{GC}}, \mathcal{F}_{\mathsf{clock}}, \mathcal{F}_{\mathsf{A^2L}})$*-hybrid model.*

A. *Informal Analysis*

Authenticity. Authenticity ensures that only authentic payment requests which are previously backed up by some locked coins are processed by the tumbler P_t during the payment procedure. In our construction this is enforced by P_t giving a blindly signed token to the sender P_s during the registration protocol (see Figure 5), which then during the puzzle promise protocol (see Figure 2) is presented to P_t, by the receiver P_r, where P_t authenticates the validity of the token and starts the payment procedure. The security of this depends on the unforgeability of the underlying (blinded) randomizable signature scheme $\widetilde{\Sigma}$. More precisely, if an adversary can make the tumbler start the payment procedure (i.e., execute the puzzle promise protocol) before previously obtaining a valid token (i.e., via the registration protocol), then we can construct an adversary against the unforgeability of the randomizable signature scheme.

Atomicity. Atomicity guarantees that no malicious party can print new money and no honest party loses money, which ensures balance security for the involved parties. This property is only related to the puzzle promise and puzzle solver protocols, and it relies on the security of the underlying adaptor signature scheme $\Xi_{R,\Sigma}$, (see Appendix D) hardness of the relation R (which is implied by the security of the adaptor signature scheme), and the security of the randomizable puzzle scheme RP.

We observe that the tumbler P_t loses money if it pays to the receiver P_r without previously getting paid by the sender P_s. This can only happen if P_r receives a valid signature signed by P_t before the execution of the puzzle solver protocol. Note that P_t only shares with P_r a pre-signature $\hat{\sigma}'_t$ over the statement A and a randomizable puzzle Z, which also includes the statement A. Hence, the only ways that P_r can have a valid signature signed by P_t before an execution of the puzzle solver protocol are the following: (i) generate a signature on behalf of P_t; (ii) obtain the solution to the randomizable puzzle Z. If the first approach succeeds, then we create an adversary that can use the generated signature in order to win the unforgeability game of the adaptor signature scheme. If the second approach succeeds, then using the puzzle solution we obtain an adversary that wins the security of the randomizable puzzle scheme. However, from our randomizable puzzle construction from Section V-B this implies that we can either break the discrete logarithm (DLOG) problem, which is believed to be a hard problem to solve, or construct an adversary against the indistinguishability of the homomorphic encryption scheme, which implies protection even against partial information leakage about the plaintext.

On the other hand, P_s loses money if at the end of the puzzle solver protocol P_t receives money, but P_r does not get paid. This can only happen if P_t provides a valid signature signed by P_s which either does not reveal the (randomized) solution that P_r needs to get paid or it reveals an invalid solution that is useless to P_r. However, the latter implies that the adversary can win the witness indistinguishability game of the adaptor signature scheme, and the former implies that the adversary can break the pre-signature adaptability property. We refer the reader to Appendix D for the formal definitions of witness indistinguishability and pre-signature adaptability of adaptor signatures.

Unlinkability. Unlinkability is defined in terms of interaction multi-graphs (defined in Section II-B) and must hold against a malicious tumbler P_t which does not collude with other parties. Towards this goal we have to show that all possible interaction multi-graphs compatible with P_t's view are equally likely.

First of all, since we are using payments of a common denomination (of amount amt as described in Section VI), P_t cannot correlate the transaction values to learn any non-trivial information. Next, in Section VI we also assumed that all protocols are coordinated in phases and epochs. All registration, puzzle promise and puzzle solver protocol executions happen during their corresponding registration, puzzle promise and puzzle solver epochs, respectively. This rules out the timing attacks where P_t intentionally delays or speeds up its interactions with another party. Looking at the protocol transcripts, we see that during the registration protocol P_t only signs a commited value, hence, due to the hiding property of the commitment scheme COM, we have that P_t does not learn the signed token, and cannot use the token with the signature it receives at the start of the puzzle promise protocol to link the sender P_s and the receiver P_r. Furthermore, the transcripts of the puzzle promise and puzzle solver protocols are unlinkable due to the privacy property of RP. More precisely, for our construction of a randomizable puzzle from Section V-B, we have that this is information-theoretically unlinkable. This is due to the fact that the randomized puzzles Z' and Z'' are equally likely to be randomizations of any puzzle Z produced by P_t during the puzzle promise phase. Lastly, in Section VI we assumed that P_s and P_r communicate through a secure and anonymous communication channel, so P_t cannot eavesdrop and use the network information to link P_s and P_r.

VIII. Performance Analysis

Implementation details. The implementation is written in C, and it relies on the RELIC library [44] for the cryptographic operations (with GMP [45]) as the underlying arithmetic library), and on the PARI library [46] for the arithmetic operations in class groups. We implemented two instantiations of the adaptor signature scheme $\Xi_{R,\Sigma}$ with the underlying signature scheme being either Schnorr or ECDSA, and the hard relation R being DLOG in both instantiations. Both instantiations are over the the elliptic curve *secp256k1*, which is also used in Bitcoin. The homomorphic encryption scheme Ψ has been instantiated with HSM-CL encryption scheme [37], [38] for 128-bit security level as described in [38, Section 4]. In order to ease the implementation we instantiated the (blinded) randomizable signature scheme $\widetilde{\Sigma}$ using Pointcheval-Sanders (PS) [39] signature scheme over the curve *BN P-256*, which analogous to *secp256k1* uses a 256-bit prime, however, unlike *secp256k1* it is pairing-friendly (i.e., pairing is secure and efficiently computable). Although, this is not the same curve as the underlying cryptocurrency (e.g., Bitcoin), we note that this is not an issue as it is used solely in the registration protocol to sign information that is kept only off-chain. Zero-knowledge proofs (and arguments) of knowledge for discrete logarithm (DLOG), CL discrete logarithm (CLDL) and Diffie-Hellman (DH) tuple have been implemented using Σ-protocols [47] and made non-interactive using the Fiat-Shamir heuristic [48]. Lastly, we have instantiated the commitment scheme COM for the registration protocol (see Figure 5) using the Pedersen commitment scheme [35]. We replaced the key generation procedure by randomly assigning keys to every party. The key generation is a one-time operation at setup (e.g., when opening a payment channel). The source code is available at https://github.com/a2l-trilero/a2l.

Testbed. We used three EC2 instances from Amazon AWS, where the tumbler P_t was a m5a.2xlarge instance (2.50GHz AMD EPYC 7571 processor with 8 cores, 32GB RAM) located in Frankfurt, while the sender P_s and the receiver P_r were m5a.large instances (2.50GHz AMD EPYC 7571 processor with 2 cores, 8GB RAM) located in Oregon and Singapore, respectively. In order to show that network latency is the biggest bottleneck in running times, we also measured performance in a LAN network. The benchmarks for a LAN network were taken on a machine with 2.80GHz Intel Xeon E3-1505M v5 processor with 8 cores, 32GB RAM. All the machines were running Ubuntu 18.04 LTS. We measured the average runtimes over 100 runs each. The results of our performance evaluation are reported in Table II.

Computation time. All our protocols complete in ~3 seconds, where the running time is dominated by network latency. The impact of network latency is obvious when we look at the running time for the LAN setting. We can observe that both Schnorr- and ECDSA-based constructions require about the same computation time, with ECDSA being slightly more expensive due to the inversion operations required when computing the signature, and the additional DH tuple NIZK proof needed during the adaptor signature computation as described in [29]. Next, we compare our constructions with the state-of-the-art payment hub TumbleBit [23]. In order to have more precise results, we performed the comparison in a LAN setting without any network latency. TumbleBit requires ~0.6 second to complete, hence, our Schnorr-based construction is slightly faster, whereas our ECDSA-based construction requires about the same time without any pre-processing. However, if we apply the pre-processing described in the Discussion paragraph below, we obtain about 2x speed-up in comparison to TumbleBit.

Communication overhead. We measured the communication overhead as the amount of information that parties need to exchange during the execution of the protocols. Hence, the bandwidth column in our table corresponds to the combined total amount of messages exchanged for the specific protocol. The ECDSA-based construction has a slightly higher communication overhead in the puzzle promise protocol compared to the Schnorr-based construction as it requires an additional ZK proof during adaptor signature computation as specified in [29]. TumbleBit requires 326KB of bandwidth, thus, our ECDSA- and Schnorr-based constructions incur ~33x less communication overhead.

Discussion. In summary, we highlight four points. First, our construction provides ~33x reduction in the communication complexity while retaining a computation time comparable to TumbleBit (or providing 2x speedup with a preprocessing technique discussed below). Interestingly, the results for TumbleBit [23] do not include any protection against the griefing attack explained in Section VI-B, whereas we have the registration protocol that provides protection for such attacks. Thus, our construction is more efficient even when providing a higher security.

Second, the reduction in communication overhead is not due to a more efficient implementation, but because A^{2}L is *asymptotically* more efficient. In a bit more detail, TumbleBit relies on the cut-and-choose technique, which implies that the security is bounded by $\binom{m+n}{m}$ and the parties need to compute and exchange messages composed of $m+n$ elements, where m and n are the parameters for the cut-and-choose game. For instance, authors of TumbleBit used $m = 15$ and $n = 285$ in puzzle solver and $m = n = 42$ in puzzle promise protocol to achieve 80 bits of security. On the other hand, A^{2}L requires to compute and exchange message composed of constant number of elements.

Third, we point that the main bottleneck with respect to computation and communication in our constructions is CL encryption [37] and CLDL zero-knowledge argument of knowledge (AoK) [49] (denoted as π_α in our construction), which comes from our randomizable puzzle instantiation (see Section V). In our implementation a single CL ciphertext has size of 2.15KB and takes ~140ms to compute and ~80ms to decrypt, while a CLDL proof has size of 2.50KB and takes ~140ms for both proving and verification operations. A possible optimization is for the tumbler to pre-compute many random α values, along with their corresponding ciphertext c_α and proof π_α during its idle time. We call this preprocessing, and it results in nearly 50% saving in the overall computation time (even though it only affects the puzzle promise phase) as shown in Table II.

Lastly, we note that our A^{2}L construction has already attracted the attention of current blockchain deployments, such as the COMIT Network, whose business focusses on cross-currency payments. In particular, they have provided an open-source proof-of-concept implementation in Rust [50].

TABLE II: Performance of A^{2}L. Time is shown in seconds.

		Registration	Promise	Solver	Total
WAN[1]	Schnorr	0.722	1.221	1.071	3.014
	ECDSA	0.726	1.251	1.076	3.053
LAN	Schnorr	0.008	0.464	0.116	0.588
	ECDSA	0.008	0.475	0.118	0.601
LAN (preprocessing)	Schnorr	0.008	0.183	0.118	0.307
	ECDSA	0.008	0.194	0.118	0.320
Bandwidth (KB)	Schnorr	0.30	7.18	2.31	9.79
	ECDSA	0.30	7.31	2.31	9.92

[1]Payment Hub (Oregon-Frankfurt-Singapore)

IX. Conclusion

We presented A^{2}L a novel three-party protocol to synchronize the updates between the payment channels involved in a PCH, and using which we build a secure, privacy-preserving, interoperable, and scalable PCH. Our construction relies on an adaptor signature scheme, which can be instantiations from Schnorr or ECDSA signatures. [29]. We defined and proved security and privacy of A^{2}L and our PCH construction in the UC framework. We further demonstrated that our PCH is the most efficient Bitcoin-compatible PCH, showing that our construction requires ~33x less bandwidth, than the state-of-the-art PCH TumbleBit, while retaining the computational cost (or providing 2x speedup with a preprocessing technique). Moreover, our PCH provides backwards compatibility with virtually all cryptocurrencies today.

References

[1] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A fistful of bitcoins: Characterizing payments among men with no names," *login Usenix Mag.*, vol. 38, no. 6, 2013.

[2] D. Ron and A. Shamir, "Quantitative analysis of the full bitcoin transaction graph," in *FC*, vol. 7859, 2013, pp. 6–24.

[3] C. Inc, "Chainanalysis: Blockchain analysis," https://www.chainalysis.com/.

[4] M. Möser, K. Soska, E. Heilman, K. Lee, H. Heffan, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan, and N. Christin, "An empirical analysis of traceability in the monero blockchain," 2018.

[5] A. Biryukov and S. Tikhomirov, "Deanonymization and linkability of cryptocurrency transactions based on network analysis," in *EuroS&P)*, 2019, pp. 172–184.

[6] G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn, "An empirical analysis of anonymity in zcash," in *USENIX Security*, 2018, p. 463–477.
[7] M. Möser and R. Böhme, "Join me on a market for anonymity," in *WPES'16*.
[8] K. Sedgwick, "4 bitcoin mixers for the privacy conscious," https://news.bitcoin.com/4-bitcoin-mixers-for-the-privacy-conscious/.
[9] L. Valenta and B. Rowan, "Blindcoin: Blinded, Accountable Mixes for Bitcoin," in *Financial Cryptography and Data Security*, 2015, pp. 112–126.
[10] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, and E. W. Felten, "Mixcoin: Anonymity for Bitcoin with Accountable Mixes," in *Financial Cryptography and Data Security*, 2014, pp. 486–504.
[11] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, D. Song, and R. Wattenhofer, "On scaling decentralized blockchains," in *Financial Cryptography and Data Security*, 2016.
[12] M. Trillo, "Stress test prepares visanet for the most wonderful time of the year," http://www.visa.com/blogarchives/us/2013/10/10/stress-test-prepares-visanet-for-the-most-wonderful-time-of-the-year/index.html, 2013, accessed: 2017-08-07.
[13] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *IEEE S&P 2018, Proceedings*, 2018, pp. 583–598.
[14] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," Technical Report, 2016.
[15] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *USENIX*, 2016, pp. 279–296.
[16] "Lightning network specifications," https://github.com/lightningnetwork/lightning-rfc.
[17] G. Maxwell, "Coinjoin: Bitcoin privacy for the real world," ,https://bitcointalk.org/index.php?topic=279249.
[18] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "Coinshuffle: Practical decentralized coin mixing for bitcoin," in *ESORICS*, vol. 8713, 2014, pp. 345–364.
[19] ——, "P2P mixing and unlinkable bitcoin transactions," in *NDSS*, 2017.
[20] S. Meiklejohn and R. Mercer, "Möbius: Trustless tumbling for transaction privacy," *PETS*, vol. 2018, no. 2, pp. 105–121.
[21] S. Dziembowski, S. Faust, and K. Hostakova, "General state channel networks," in *CCS*, 2018.
[22] R. Khalil, A. Gervais, and G. Felley, "Nocust - a securely scalable commit-chain," Cryptology ePrint Archive, Report 2018/642, 2018.
[23] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "TumbleBit: An untrusted bitcoin-compatible anonymous payment hub," in *NDSS*, 2017.
[24] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," in *CCS*, 2017.
[25] J. Lind, O. Naor, I. Eyal, F. Kelbert, P. R. Pietzuch, and E. G. Sirer, "Teechain: Reducing storage costs on the blockchain with offline payment channels," in *ACM SYSTOR*, 2018, p. 125.
[26] I. Bentov, Y. Ji, F. Zhang, Y. Li, X. Zhao, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware," in *ePrint Archive*, 2017, p. 1153. [Online]. Available: http://eprint.iacr.org/2017/1153
[27] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," in *CCS*, 2017, p. 473–489.
[28] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment hubs over cryptocurrencies," Cryptology ePrint Archive, Report 2017/635, 2017.
[29] L. Aumayr, O. Ersoy, A. Erwig, S. Faust, K. Hostakova, M. Maffei, P. Moreno-Sanchez, and S. Riahi, "Generalized bitcoin-compatible channels," Cryptology ePrint Archive, Report 2020/476, 2020.
[30] G. Fuchsbauer, M. Orrù, and Y. Seurin, "Aggregate cash systems: A cryptographic investigation of mimblewimble," Cryptology ePrint Archive, Report 2018/1039, 2018, https://eprint.iacr.org/2018/1039.
[31] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," Cryptology ePrint Archive, Report 2000/067, 2000.
[32] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *CCS*, 2017, p. 455–471.
[33] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," in *NDSS*, 2019.
[34] D. Robinson, "Htlcs considered harmful," stanford Blockchain Conference https://cbr.stanford.edu/sbc19/.
[35] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Advances in Cryptology – CRYPTO' 91*, 1991, p. 129–140.
[36] M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge and its applications," in *STOC*. ACM, 1988, pp. 103–112.
[37] G. Castagnos and F. Laguillaumie, "Linearly homomorphic encryption from ddh," in *CT-RSA*, 2015, pp. 487–505.
[38] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker, "Two-party ecdsa from hash proof systems and efficient instantiations," in *Advances in Cryptology – CRYPTO*, 2019, pp. 191–221.
[39] D. Pointcheval and O. Sanders, "Short randomizable signatures," in *Topics in Cryptology - CT-RSA 2016*, 2016, pp. 111–126.
[40] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," 2019.
[41] S. Noether, "Discrete logarithm equality across groups," 2018, https://www.getmonero.org/resources/research-lab/pubs/MRL-0010.pdf.
[42] J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," in *Advances in Cryptology — EUROCRYPT 2001*, pp. 93–118.
[43] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in *Theory of Cryptography Conference*, 2007, pp. 61–85.
[44] D. F. Aranha and C. P. L. Gouvêa, "RELIC is an Efficient LIbrary for Cryptography," Github Project, 2020.
[45] T. Granlund and the GMP development team, *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6th ed., 2019.
[46] *PARI/GP version* `2.12.0`, The PARI Group, Univ. Bordeaux, 2019.
[47] I. Damgård, "On the σ-protocols," Lecture Notes, University of Aarhus, Department for Computer Science, 2002.
[48] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology — CRYPTO' 86*, 1987, pp. 186–194.
[49] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker, "Bandwidth-efficient threshold ec-dsa," pp. 266–296, 2020.
[50] bonomat (pseudonym), "A2l proof of concept by comit network," https://github.com/comit-network/a2l-poc.
[51] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin." in *IEEE S&P 2014*, 2014, pp. 459–474.
[52] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *CCS*. ACM, 2017, pp. 455–471.
[53] S. Dziembowski, L. Eckey, S. Faust, J. Hesse, and K. Hostáková, "Multi-party virtual state channels," in *Advances in Cryptology – EUROCRYPT 2019*, 2019, pp. 625–656.
[54] J. Katz, U. Maurer, B. Tackmann, and V. Zikas, "Universally composable synchronous computation," in *TCC*, 2013, pp. 477–498.
[55] J. Camenisch and A. Lysyanskaya, "A formal treatment of onion routing," in *Advances in Cryptology – CRYPTO 2005*, pp. 169–187.
[56] Y. Lindell, "Fast secure two-party ecdsa signing," Cryptology ePrint Archive, Report 2017/552, 2017.

APPENDIX

A. Discussion

We discuss here further aspects of our PCH regarding both limitations of unlinkability and practical deployment.

1) Limitations of Unlinkability: In this section, we discuss the unlinkability limitations inherent to the PCH setting, and thus also affecting our construction. We remark that these limitations are inherent to any tumbler protocol, as shown for instance in TumbleBit [23]. Furthermore, even with these limitations, our construction augments the privacy guarantees for the users of a PCH service.

Epoch anonymity. Assume that P_t executes the puzzle promise protocol with k parties during a phase of an epoch. If within the next phase, k payments successfully complete, then the anonymity set is of size k since there exist k compatible interaction graphs, as defined in Section II-B.

It is however not always the case that k is equal to the total number of parties. The exact anonymity level can be established only at the end of the epoch depending on the number of successful puzzle promise and puzzle solver protocols. For instance, anonymity is reduced by 1 if P_t aborts a payment made by a party P_s. The payment between P_s and P_r would be the only one failing, thereby showing that P_r was the expected receiver. It is important to note that P_s does not lose coins as P_t obtains a valid channel update only if it cooperates in solving the puzzle.

Tumbler/receiver collusion. The tumbler P_t and the receiver P_r can collude to learn the identity of the sender P_s. Intuitively, this type of attack is only useful if P_r can be paid by P_s without learning its true identity (e.g., in anonymous donations). We partially address this collusion in our constructions by letting P_s randomize the puzzle it receives from P_r. However, P_r can still send a maliciously constructed puzzle (more precisely, an invalid puzzle or a non-randomized puzzle) to P_s, which can cause an abort or leak information to P_t during the execution of the puzzle solver protocol between P_s and P_t. This in turn can allow P_t to link that P_s was the party that intended to pay P_r. One possible mitigation to this is to force P_r to give a zero-knowledge proof to P_s that the puzzle it sends is a valid randomized puzzle.

Intersection attack. The aforementioned k-anonymity notion is broadly used in mixing protocols with an intermediate P_t. However, P_t can further reduce the anonymity set. At any epoch, P_t can record the set of senders and receivers that participate in the puzzle solver and puzzle promise protocols respectively. Then, P_t can correlate this information across phases and epochs to de-anonymize users (e.g., using frequency analysis).

Ceiling attack. The amount of payments that a certain P_r can receive during a certain epoch is limited by the balance at the channel ς between P_t and P_r. If the channel is exhausted (i.e., $\varsigma.\mathsf{cash}(P_t) = 0$), P_t can deterministically derive the fact that P_r is not a potential receiver within the current epoch.

Attacks with auxiliary information. Our notion of unlinkability does not consider auxiliary information available to P_t. Assume that P_t knows that a certain P_r has an online shop selling a product for a value $2{\cdot}\mathsf{amt}$. Further assume that during an epoch, P_t executes the puzzle promise protocol only once on every channel except with P_r, for which the puzzle promise protocol is executed twice. Similarly, P_t could observe that there exists a single P_s executing twice the puzzle solver protocol, allowing P_t to link the pair P_s, P_r. As indicated in [23], this type of attacks (called Potato attack in [23]) could be mitigated by aggregating payments or adding noise à la differential privacy.

2) Practical Deployment: In this section, we discuss the practical considerations for real-life deployment of our PCH.

Hub vs tumbler functionality. Our PCH, as described in this work, provides a tumbler functionality, that is, allows payments between P_s and P_r while ensuring atomicity and unlinkability. Providing these guarantees comes at the cost of communication and computation overhead when compared to payment hubs that simply forward payments from P_s to P_r through P_t. Yet, our evaluation results show that our construction is the most efficient PCH among those tumbler protocols with emphasis on privacy.

Variable payment amounts and fees. Our PCH sacrifices the support of arbitrary payment amounts in favor of achieving unlinkability. While for readability, we have described our PCH working with a single fixed payment amount amt, this limitation can be somewhat mitigated in practice by having a set of fixed denominations (e.g., $\mathsf{amt}, 10 \cdot \mathsf{amt}, 100 \cdot \mathsf{amt}$, etc.). This thereby provides a tradeoff between more practical functionality at the expense of reducing the anonymity set (and thus unlinkability) to those payments with the same denomination. Similarly, our PCH can be extended to let the tumbler P_t charge a fee for each puzzle promise/solver pair that it processes. In particular, the PCH could be setup such that each P_s pays $\mathsf{amt} + \mathsf{fee}$ while P_t pays only amt to each P_r. As before, the unlinkability property requires that fee is the same for all payments within the anonymity set.

Cross-currency payments. In principle, the cryptographic protocol in A^{2}L (and thus our PCH construction) supports the authorization of transactions across different cryptocurrencies. However, deploying our construction as full-fledge cross-currency PCH requires to consider several practical aspects. In the following, we describe (a possibly incomplete list of) them. First, one would require to fix exchange rate between the cryptocurrencies being exchanged to ensure unlinkability of payments (similar to the aforementioned argument for the fees). In practice, one could fix an exchange rate for a period of time (say one day) and let the PCH use it during that period. Then, the tumbler P_t could account for the fluctuations on the exchange rate during that period by (possibly over approximating) the fee charged to each payment. Second, one would require to fix a timeout for each phase independently of the cryptocurrencies being exchanged (which may have different block creation times) in order to maintain unlinkability.

Communication between P_s and P_r. As discussed in Section VI, we assumed that P_t does not notice the communication between P_s and P_r (e.g., the sending of the puzzle and its randomized solution), as otherwise it trivially breaks unlinkability. We note that this a standard assumption in payment protocols providing privacy guarantees [23], [51]. In practice, P_s and P_t could communicate via an anonymous communication channel (e.g., Tor).

Implementing phases and epochs. We expect our construction to run in phases and epochs as described in Section VI. An epoch constitutes a single run of our complete construction, whereas phases are disjoint timeslots inside an epoch, which correspond to our individual protocol runs (e.g., all instances of the registration protocol run during the registration phase). In practice one can simply set a system specific duration for an epoch (e.g., one day), and then divide the epoch duration into four equal timeslots (e.g., 6 hours per slot), one for each of our four phases: registration phase, puzzle promise phase, puzzle solver phase, and open phase. Making sure that the timeslots within an epoch are equal, and more importantly, disjoint reduces the possible information leakage that can be obtained from the timing attacks.

B. Description of our PCH

In our PCH, we combine A^{2}L with a blockchain $\mathcal{B}$ in order to realize a fully-fledged PCH. We denote the channel between P_s and P_t as ς, and the channel between P_t and P_r as ς'. A payment of amt coins between P_s and P_r through P_t is realized by updating both channels, such that P_t gets amt coins in ς if and only if P_r gets amt coin in ς'. In order to ensure this invariant, our PCH relies on two contracts built upon A^{2}L. More precisely, first P_t and P_r execute the $\mathsf{PuzzlePromise}$ protocol from A^{2}L to get the input required to establish the A^{2}L-Promise$(P_t, P_r, \Pi, \mathsf{amt}, \varsigma', t')$ contract:

1) If P_r produces a valid signature σ, so that $\mathsf{Verify}(\Pi, \sigma) = 1$ before time t' expires, then ς' is updated as $(\varsigma'.\mathsf{cash}(P_t) \mathrel{-}= \mathsf{amt}, \varsigma'.\mathsf{cash}(P_r) \mathrel{+}= \mathsf{amt})$ (i.e., tumbler pays the receiver amt coins).
2) If timeout t' expires, ς' remains unchanged (i.e., tumbler regains control over amt coins).

Here, Π comes from the output of the $\mathsf{PuzzlePromise}$ protocol in A^{2}L, and t is an expiration time (validity period) of the promise, which is properly set to give P_r the time it needs to reveal the final valid signature σ. In case this does not happen, then P_t gets back the money, thereby avoiding an indefinite locking of money in the channel. Notice that we require the blockchain $\mathcal{B}$ to support the Verify algorithm and time management in its scripting language. This is the case in practice as Verify is implemented as the unmodified verification algorithm of the digital signature scheme, and virtually all cryptocurrencies natively implement a timelock mechanism where time is measured as the number of blocks in the blockchain.

Second, P_r sends the randomized puzzle Z' (as output by the $\mathsf{PuzzlePromise}$ protocol) to P_s. Then, P_s and P_t execute the $\mathsf{PuzzleSolver}$ protocol to get the input required to establish the A^{2}L-Solve$(P_s, P_t, Z', \mathsf{amt}, \varsigma, t)$ contract:

1) If before t, P_t sends P_s the solution α' to the cryptographic challenge encoded in Z', ς is updated as $(\varsigma.\mathsf{cash}(P_s) \mathrel{-}= \mathsf{amt}, \varsigma.\mathsf{cash}(P_t) \mathrel{+}= \mathsf{amt})$ (i.e., the sender pays tumbler amt coins).
2) Otherwise, ς remains unchanged (i.e., the sender regains control over amt coins).

Lastly, P_s gets the solution α' to the challenge encoded in the puzzle Z', and sends α' to P_r who can complete the A^{2}L-Promise contract with the signature $\sigma := \mathsf{Open}(\Pi, \alpha')$. Our PCH construction can be seen in Figure 6.

C. Security and Privacy Model

1) Preliminaries: We define our security and privacy model modularly by leveraging the Universal Composability (UC) framework from Canetti [31]. More precisely, we rely on the synchronous version of global UC framework (GUC) [43]. We first describe the ideal functionality $\mathcal{F}_{\mathsf{A^2L}}$ for A^{2}L, which captures the expected behavior as well as the security and privacy properties of the interaction among the sender P_s, receiver P_r and tumbler P_t, for which we provided an implementation in Section VI-A, along with its extension for handling griefing attack in Section VI-B. Then, we describe payment-channel hub (PCH) ideal functionality $\mathcal{F}_{\mathsf{PCH}}$ covering the security and privacy notions for a PCH, and which relies on $\mathcal{F}_{\mathsf{A^2L}}$, and for which we already presented an implementation in Appendix B. The security proofs for A^{2}L and our PCH are given in Appendix F2 and Appendix F3, respectively.

Attacker model. We model the parties as interactive Turing machines (ITMs), which communicate with a trusted functionality $\mathcal{F}$ via secure and authenticated communication channels. We model the adversary $\mathcal{S}$ as a PPT machine that has access to an interface $\mathsf{corrupt}(\cdot)$, which takes as input a party identifier P and provides the attacker with the internal state of P. From that point onward, all subsequent incoming and outgoing communication of P is routed through $\mathcal{S}$. As commonly done in the literature [23], [40], [52], we consider the static corruption model, that is, the adversary commits to the identifiers of the parties it corrupts ahead of time.

Communication model. We consider a synchronous communication network, where communication proceeds in discrete rounds. We follow [53] (which in turn follows [54]), and formalizes the notion of rounds via a global ideal functionality $\mathcal{F}_{\mathsf{clock}}$, which represents the clock. The ideal functionality requires all honest parties to indicate that they are ready to proceed to the next round before the clock is ticked. Similar to [53], we treat the clock functionality as a global ideal functionality defined in the GUC model [43]. This implies that all parties are aware of the given round.

We assume that the parties are connected via authenticated communication channels with guaranteed delivery of exactly one round (as in [29]). The adversary can change the order of messages that were sent in the same round, but it cannot delay or drop messages sent between parties, or it cannot insert a new message. For simplicity, we assume that computation is instantaneous. These assumptions on the communication channels are formalized as an ideal functionality $\mathcal{F}_{\mathsf{GDC}}$, as defined in [53].

Additionally, we use the secure transmission functionality $\mathcal{F}_{\mathsf{smt}}$, as defined in [31], which ensures that the adversary cannot read or change the content of the messages. Lastly, we assume the existence of an anonymous communication channel as defined in [55], which we denote here as $\mathcal{F}_{\mathsf{anon}}$, and which is only needed for communication between the sender P_s and receiver P_r.

Payment channels. We make use of the global ideal functionality $\mathcal{F}_{\mathsf{GC}}$ [29], which defines generalized channels, which can be seen as a generalization of payment channels. The ideal functionality provides all the backbone necessary for handling payment channels, such as the following interfaces: Create and Close are used for opening and closing a payment channel, respectively, and Update is used to update the balances of the

Public parameters: constant amount amt, validity period υ of a promise, current time Δ

$P_s(\varsigma)$		$P_t(\varsigma, \varsigma')$	$P_r(\varsigma')$
Create escrow output oid			
$(\mathsf{tid}, \sigma'_{\mathsf{tid}}) \leftarrow \langle \mathsf{Registration}_{P_s}(\mathsf{oid}), \mathsf{Registration}_{P_t}() \rangle$			
	$\xrightarrow{(\mathsf{tid}, \sigma'_{\mathsf{tid}})}$		
		If $\varsigma'.\mathsf{cash}(P_t) < \mathsf{amt}$ then abort	
		$\langle \mathsf{PuzzlePromise}_{P_t}(), \mathsf{PuzzlePromise}_{P_r}(\mathsf{tid}, \sigma'_{\mathsf{tid}}) \rangle \rightarrow (\Pi, (Z, Z'))$	
		If $\Pi = \perp$ or $Z = \perp$ or $Z' = \perp$ then abort	
		Set $t' := \Delta + \upsilon$	
		$\xleftrightarrow{\mathrm{A^2L{-}Promise}(P_t, P_r, \Pi, \mathsf{amt}, \varsigma', t')}$	
	$\xleftarrow{Z'}$		
If $\varsigma.\mathsf{cash}(P_s) < \mathsf{amt}$ or $t' < \Delta$ then abort			
$\xleftrightarrow{\mathrm{A^2L{-}Solve}(P_s, P_t, Z', \mathsf{amt}, \varsigma, t)}$			
$\alpha' \leftarrow \langle \mathsf{PuzzleSolver}_{P_s}(Z'), \mathsf{PuzzleSolver}_{P_t}() \rangle$			
If $\alpha' = \perp$ then abort			
	$\xrightarrow{\alpha'}$		
			$\sigma := \mathsf{Open}(\Pi, \alpha')$
			Check $\mathsf{Verify}(\Pi, \sigma)$

Fig. 6: Our PCH construction (cryptographic keys removed as inputs to subprotocols for readability).

Ideal Functionality $\mathcal{F}_{\mathsf{A^2L}}$

Registration: On input $(\mathsf{Registration}, P_r)$ from P_s, $\mathcal{F}_{\mathsf{A^2L}}$ proceeds as follows:
- Send $(\mathsf{registration{-}req}, P_s)$ to P_t and $\mathcal{S}$.
- Receive $(\mathsf{register{-}res}, b)$ from P_t.
- If $b = \perp$ then abort.
- Sample $\mathsf{tid} \leftarrow_\$ \{0,1\}^\lambda$ and add tid into $\mathcal{T}$.
- Send $(\mathsf{registered}, \mathsf{tid})$ to P_s, P_r and $\mathcal{S}$.

Puzzle Promise: On input $(\mathsf{PuzzlePromise}, P_s, \mathsf{tid})$ from P_r, $\mathcal{F}_{\mathsf{A^2L}}$ proceeds as follows:
- If $\mathsf{tid} \notin \mathcal{T}$ then abort.
- Else remove tid from $\mathcal{T}$.
- Send $(\mathsf{promise{-}req}, P_r, \mathsf{tid})$ to P_t and $\mathcal{S}$.
- Receive $(\mathsf{promise{-}res}, b)$ from P_t.
- If $b = \perp$ then abort.
- Sample $\mathsf{pid}, \mathsf{pid}' \leftarrow_\$ \{0,1\}^\lambda$.
- Store the tuple $(\mathsf{pid}, \mathsf{pid}', \perp)$ into $\mathcal{P}$.
- Send $(\mathsf{promise}, (\mathsf{pid}, \mathsf{pid}'))$ to P_r, $(\mathsf{promise}, \mathsf{pid})$ to P_t, $(\mathsf{promise}, \mathsf{pid}')$ to P_s, and inform $\mathcal{S}$.

Puzzle Solver: On input $(\mathsf{PuzzleSolver}, P_r, \mathsf{pid}')$ from P_s, $\mathcal{F}_{\mathsf{A^2L}}$ proceeds as follows:
- If $\nexists(\cdot, \mathsf{pid}', \cdot) \in \mathcal{P}$ then abort.
- Send $(\mathsf{solve{-}req}, P_s, \mathsf{pid}')$ to P_t and $\mathcal{S}$.
- Receive $(\mathsf{solve{-}res}, b)$ from P_t.
- If $b = \perp$ then abort.
- Update entry to $(\cdot, \mathsf{pid}', \top)$ in $\mathcal{P}$.
- Send $(\mathsf{solved}, \mathsf{pid}', \top)$ to P_s, P_r and $\mathcal{S}$.

Open: On input $(\mathsf{Open}, \mathsf{pid})$ from P_r, $\mathcal{F}_{\mathsf{A^2L}}$ proceeds as follows:
- If $\nexists(\mathsf{pid}, \cdot, b) \in \mathcal{P}$ or $b = \perp$ then send $(\mathsf{open}, \mathsf{pid}, \perp)$ to P_r and abort. Else send $(\mathsf{open}, \mathsf{pid}, \top)$ to P_r.

Fig. 7: Ideal functionality $\mathcal{F}_{\mathsf{A^2L}}$.

parties involved in the payment channel.

(Global) Universal composability. We briefly overview the notion of secure realization in the UC framework [31], and its variant called global UC (GUC) framework [43]. Intuitively, a protocol realizes an ideal functionality if any distinguisher (the environment) has no way of distinguishing between a real run of the protocol and a simulated interaction with the ideal functionality.

For our $\mathcal{F}_{\mathsf{PCH}}$ ideal functionality we rely on the global channel functionalities $\mathcal{F}_{\mathsf{GC}}$ and global clock functionality $\mathcal{F}_{\mathsf{clock}}$. Hence, we need to define the UC-realization with respect to these global functionalities. More precisely, let π be a protocol with access to the global channel $\mathcal{F}_{\mathsf{GC}}$ and the global clock $\mathcal{F}_{\mathsf{clock}}$. Let $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{E}}$ denote the ensemble of the outputs of the environment $\mathcal{E}$ when interacting with the adversary $\mathcal{A}$ and users running protocol π, we define universal composability as follows. Then, we can define the UC-realization with respect to the global functionalities as:

Definition 5 (Global Universal Composability). *A protocol π UC-realizes an ideal functionality $\mathcal{F}$ with respect to a global channel $\mathcal{F}_{\mathsf{GC}}$ and global clock $\mathcal{F}_{\mathsf{clock}}$ if for any* PPT *adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$, such that for any environment $\mathcal{E}$, the ensembles $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{GC}},\mathcal{F}_{\mathsf{clock}}}_{\pi,\mathcal{A},\mathcal{E}}$ and $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{GC}},\mathcal{F}_{\mathsf{clock}}}_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are computationally indistinguishable.*

2) Anonymous Atomic Lock (A^{2}L): Here, we formalize the notion of anonymous atomic locks (A^{2}L).

Ideal functionality. We illustrate the ideal functionality $\mathcal{F}_{\mathsf{A^2L}}$ for A^{2}L in Figure 7, where it implicitly uses $\mathcal{F}_{\mathsf{GDC}}$, $\mathcal{F}_{\mathsf{smt}}$ and $\mathcal{F}_{\mathsf{anon}}$, thus, $\mathcal{F}_{\mathsf{A^2L}}$ is defined in the $(\mathcal{F}_{\mathsf{GDC}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{anon}})$-hybrid model.

Furthermore, $\mathcal{F}_{\mathsf{A^2L}}$ manages a list $\mathcal{P}$ (initially set to $\mathcal{P} := \emptyset$), to keep track of the cryptographic puzzles. The entries in the list $\mathcal{P}$ have the format $(\mathsf{pid}, \mathsf{pid}', b)$, where pid is the puzzle, pid' is the randomized version of the puzzle and b is a bit specifying whether the puzzle has been solved. Additionally, it managed as list $\mathcal{T}$, which keeps track of the valid (i.e., active and unused) tokens.

$\mathcal{F}_{\mathsf{A^2L}}$ provides three interfaces, which are depicted in Figure 7. The Registration interface allows a party to obtain a token, which is used for authentication purposes. The PuzzlePromise interface given as input a valid token provides a puzzle. The PuzzleSolver interface allows a party to acquire a solution to a puzzle. Lastly, the Open interface allows a party to check the validity of the puzzle solution.

Discussion. We introduced the security and privacy notions of interest for our system in Section II-B. Here, we paraphrase them regarding A^{2}L and explain why $\mathcal{F}_{\mathsf{A^2L}}$ achieves these notions.

Authenticity: Authenticity ensures that puzzle promise can only be executed if a valid token has been acquired and that each token can only be used once. This is enforced by $\mathcal{F}_{\mathsf{A^2L}}$ as it checks the validity of the input token tid to the PuzzlePromise interface, before continuing with its execution. If the token is invalid it aborts the execution, and otherwise, it removes the token from the list $\mathcal{T}$ and continues with the execution of PuzzlePromise.

Atomicity: Loosely speaking, atomicity for A^{2}L means that a puzzle can only be solved, if there has been a corresponding execution of puzzle solver for that puzzle. This is enforced by $\mathcal{F}_{\mathsf{A^2L}}$ because it keeps track of the puzzles in the list $\mathcal{P}$, and checks whether the puzzle given as input to the Open interface corresponds to one of the existing entries in the list $\mathcal{P}$ and it has been already solved. Since the puzzles only get solved inside the PuzzleSolver interface and $\mathcal{F}_{\mathsf{A^2L}}$ is trusted, this ensures that PuzzleSolver must be called before Open in order for it to succeed.

Unlinkability: Intuitively, unlinkability means that the tumbler P_t does not learn information that allows it to associate the sender P_s and the receiver P_r of a payment (i.e, cannot link the calls of PuzzlePromise and PuzzleSolver). This property is enforced by $\mathcal{F}_{\mathsf{A^2L}}$ since for each call to the PuzzlePromise interface, $\mathcal{F}_{\mathsf{A^2L}}$ samples both a puzzle pid and its randomized version pid$'$, and stores them as part of the same entry in $\mathcal{P}$. Then, only the randomized puzzle pid$'$ is given to P_t inside the PuzzleSolver interface.

Additionally, since we assumed the existence of secure and anonymous communication channel for P_s and P_t (see Section VI), which can be realized with $\mathcal{F}_{\mathsf{anon}}$ [55] ideal functionality, P_t cannot use the network information to correlate P_s and P_r. We remark that this assumption is indispensable for unlinkability and is commonly adopted in the PCH-related literature, such as in [**?**], [23].

3) Payment Channel Hub (PCH): Here, we formalize the notion of a PCH by relying on A^{2}L. We use the notation described in Section II for payment channels.

Ideal functionality. $\mathcal{F}_{\mathsf{PCH}}$ ideal functionality makes use of $\mathcal{F}_{\mathsf{GDC}}, \mathcal{F}_{\mathsf{GC}}, \mathcal{F}_{\mathsf{clock}}$, and $\mathcal{F}_{\mathsf{A^2L}}$ ideal functionalities, hence, it is defined in $(\mathcal{F}_{\mathsf{GDC}}, \mathcal{F}_{\mathsf{GC}}, \mathcal{F}_{\mathsf{clock}}, \mathcal{F}_{\mathsf{A^2L}})$-hybrid model. $\mathcal{F}_{\mathsf{PCH}}$ is shown in Figure 8.

$\mathcal{F}_{\mathsf{PCH}}$ manages a list $\mathcal{C}$ (initially set to $\mathcal{C} := \emptyset$), which stores the currently open channels. In our model, we expect that every participating party has a channel with the central designated tumbler P_t, and that every payment transfers a fixed amount amt of coins, which we assume is globally available to all parties. Additionally, we assume that there is a constant validity period v for payments, and we denote the current time by Δ. In order to simplify the model we do not include any transaction fees, but we note that our protocol retains its security and privacy properties even in the presence of constant transaction fees. $\mathcal{F}_{\mathsf{PCH}}$ provides three interfaces, where OpenChannel and CloseChannel operations are the standard channel opening/closing operations [16], [52], which in our case are handled via simulator calls to the $\mathcal{F}_{\mathsf{GC}}$ ideal functionality defined in [29]. Lastly, Pay handles the payment operation from the sender P_s to the receiver P_r via the tumbler P_t by making use of $\mathcal{F}_{\mathsf{A^2L}}$.

Ideal Functionality $\mathcal{F}_{\mathsf{PCH}}$

Open Channel: On input $(\mathsf{OpenChannel}, \varsigma, \mathsf{txid}_P)$ from a party P, $\mathcal{F}_{\mathsf{PCH}}$ proceeds as follows:

- Send $(\mathsf{Create}, \varsigma, \mathsf{txid}_P)$ to $\mathcal{S}$.
- Receive b from $\mathcal{S}$.
- If $b = \bot$, then abort.
- Add ς into $\mathcal{C}$.
- Send $(\mathsf{created}, \varsigma.\mathsf{cid})$ to $\varsigma.\mathsf{users}$.

Close Channel: On input $(\mathsf{CloseChannel}, \varsigma)$ from a party P, $\mathcal{F}_{\mathsf{PCH}}$ proceeds as follows:

- Send $(\mathsf{Close}, \varsigma.\mathsf{cid})$ to $\mathcal{S}$.
- Receive b from $\mathcal{S}$.
- If $b = \bot$, then abort.
- Remove ς from $\mathcal{C}$.
- Send $(\mathsf{closed}, \varsigma.\mathsf{cid})$ to $\varsigma.\mathsf{users}$.

Pay: On input (Pay, P_r) from P_s, $\mathcal{F}_{\mathsf{PCH}}$ proceeds as follows:

- Retrieve ς and ς' from $\mathcal{C}$, where $\varsigma.\mathsf{users} = \{P_s, P_t\}$ and $\varsigma'.\mathsf{users} = \{P_t, P_r\}$.
- If $\varsigma = \bot$ or $\varsigma' = \bot$ then abort.
- Send $(\mathsf{Registration}, P_r)$ to $\mathcal{S}$.
- Receive tid from $\mathcal{S}$.
- If tid $= \bot$ then abort.
- Set $t' = \Delta + 2v$ and propose $\varsigma'.\mathsf{TLP}(\theta' := (\varsigma'.\mathsf{cash}(P_t) \mathrel{-}= \mathsf{amt}, \varsigma'.\mathsf{cash}(P_s) \mathrel{+}= \mathsf{amt}), t')$ to P_t and P_r.
- Send $(\mathsf{PuzzlePromise}, P_s, \mathsf{tid})$ to $\mathcal{S}$.
- Receive $(\mathsf{pid}, \mathsf{pid}')$ from $\mathcal{S}$.
- If $\mathsf{pid}' = \bot$ then abort.
- Set $t = \Delta + v$ and propose $\varsigma.\mathsf{TLP}(\theta := (\varsigma.\mathsf{cash}(P_s) \mathrel{-}= \mathsf{amt}, \varsigma.\mathsf{cash}(P_t) \mathrel{+}= \mathsf{amt}), t)$ to P_s and P_t.
- Send $(\mathsf{PuzzleSolver}, P_r, \mathsf{pid}')$ to $\mathcal{S}$.
- Receive b from $\mathcal{S}$.
- If $b = \bot$ then abort.
- Send $(\mathsf{Open}, \mathsf{pid})$ to $\mathcal{S}$.
- Receive b from $\mathcal{S}$.
- If $b = \bot$ or $t < \Delta$ then send $\bot$ to P_s.
- Send the update $(\mathsf{Update}, \varsigma.\mathsf{cid}, \theta := (\varsigma.\mathsf{cash}(P_s) \mathrel{-}= \mathsf{amt}, \varsigma.\mathsf{cash}(P_t) \mathrel{+}= \mathsf{amt}))$ to $\mathcal{S}$.
- Send the update $(\mathsf{Update}, \varsigma'.\mathsf{cid}, \theta' := (\varsigma'.\mathsf{cash}(P_t) \mathrel{-}= \mathsf{amt}, \varsigma'.\mathsf{cash}(P_r) \mathrel{+}= \mathsf{amt}))$ to $\mathcal{S}$.

Fig. 8: Ideal functionality $\mathcal{F}_{\mathsf{PCH}}$.

Discussion. We discuss here how the ideal functionality captures the security and privacy notions of interest for payment hubs as defined in Section II-B.

Authenticity: This property ensures that only authenticated payments should go through. Since $\mathcal{F}_{\mathsf{PCH}}$ is defined in hybrid model with $\mathcal{F}_{\mathsf{A^2L}}$, it automatically inherits the authenticity guarantees of $\mathcal{F}_{\mathsf{A^2L}}$.

Atomicity: The system should not be exploited to print new money or steal existing money, even when parties collude. $\mathcal{F}_{\mathsf{PCH}}$ achieves atomicity as the only place where the balances are updated is at the end of the Pay interface, where the ideal functionality makes sure that all the previous operations related to A^{2}L have been successfully finished. This implies that $\mathcal{F}_{\mathsf{PCH}}$ inherits the atomicity guarantees of $\mathcal{F}_{\mathsf{A^2L}}$.

Unlinkability: The intermediary should not learn information that allows it to associate the sender and the receiver of a payment. In Appendix C2 it was argued that $\mathcal{F}_{\mathsf{A^2L}}$ provides such an unlinkability guarantee. Since, $\mathcal{F}_{\mathsf{PCH}}$ is defined in hybrid model with $\mathcal{F}_{\mathsf{A^2L}}$, it inherits the unlinkability guarantees on $\mathcal{F}_{\mathsf{A^2L}}$. However, $\mathcal{F}_{\mathsf{PCH}}$ also handles the payments, hence, we need to ensure that the actual payments do not leak any information that can be used to link the sender/receiver pair. Though, we note that $\mathcal{F}_{\mathsf{PCH}}$ uses constant amount amt for all payments and fixed time intervals, therefore, the amounts and timing information do not help in differing the payments.

D. Adaptor Signatures

Here we give a more detailed and formal description of an adaptor signature and its properties. The definitions and security experiments are taken from [29] with minor changes to fit our notation. Adaptor signatures have been introduced by the cryptocurrency community to tie together the authorization of a transaction with leakage of a secret value. Due to its utility, adaptor signatures have been used in previous works for various applications like atomic swaps or payment channel networks [40]. An adaptor signature scheme is essentially a two-step signing algorithm bound to a secret: first a partial signature is generated such that it can be completed only by a party that knows a certain secret, where the completion of the signature reveals the underlying secret.

More precisely, we define an adaptor signature scheme with respect to a standard signature scheme Σ and a hard relation R. Before moving on with the formal definition of an adaptor signature, we first recall the definition of a hard relation.

Definition 6 (Hard Relation)**.** *Let R be a relation with statement/witness pairs (Y, y). Let us denote L_R the associated language defined as $L_R := \{Y \mid \exists y \text{ s.t. } (Y, y) \in R\}$. We say that R is a* hard relation *if the following holds:*

- *There exists a* PPT *sampling algorithm* $\mathsf{GenR}(1^\lambda)$ *that on input the security parameter λ outputs a statement/witness pair $(Y, y) \in R$.*
- *The relation is poly-time decidable.*
- *For all ppt adversaries $\mathcal{A}$ there exists a negligible function* negl*, such that:*

$$\Pr\left[(Y, y^*) \in R \;\middle|\; \begin{array}{c}(Y, y) \leftarrow \mathsf{GenR}(1^\lambda),\\ y^* \leftarrow \mathcal{A}(Y)\end{array}\right] \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the randomness of GenR *and $\mathcal{A}$.*

In an adaptor signature scheme, for any statement $Y \in L_R$, a signer holding a secret key is able to produce a *pre-signature* w.r.t. Y on any message m. Such pre-signature can be *adapted* into a full valid signature on m if and only if the adaptor knows a witness for Y. Moreover, if such a valid signature is produced, it must be possible to extract the witness for Y given the pre-signature and the adapted signature. This is formalized as follows, where we take the message space $\mathcal{M}$ to be $\{0, 1\}^*$.

Definition 7 (Adaptor Signature Scheme)**.** *An adaptor signature scheme w.r.t. a hard relation R and a signature scheme $\Sigma = (\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$ consists of four algorithms $\Xi_{R,\Sigma} = (\mathsf{PreSig}, \mathsf{Adapt}, \mathsf{PreVf}, \mathsf{Ext})$ defined as:*

$\mathsf{PreSig}(\mathsf{sk}, m, Y)$**:** *is a* PPT *algorithm that on input a secret key* sk*, message $m \in \{0, 1\}^*$ and statement $Y \in L_R$, outputs a pre-signature $\hat{\sigma}$.*

$\mathsf{PreVf}(\mathsf{pk}, m, Y, \hat{\sigma})$**:** *is a* DPT *algorithm that on input a public key* pk*, message $m \in \{0, 1\}^*$, statement $Y \in L_R$ and pre-signature $\hat{\sigma}$, outputs a bit b.*

$\mathsf{Adapt}(\hat{\sigma}, y)$**:** *is a* DPT *algorithm that on input a pre-signature $\hat{\sigma}$ and witness y, outputs a signature σ.*

$\mathsf{Ext}(\sigma, \hat{\sigma}, Y)$**:** *is a* DPT *algorithm that on input a signature σ, pre-signature $\hat{\sigma}$ and statement $Y \in L_R$, outputs a witness y such that $(Y, y) \in R$, or $\perp$.*

In addition to the standard signature correctness, an adaptor signature scheme has to satisfy *pre-signature correctness*. Informally, an honestly generated pre-signature w.r.t. a statement $Y \in L_R$ is a valid pre-signature and can be adapted into a valid signature from which a witness for Y can be extracted.

Definition 8 (Pre-signature Correctness)**.** *An adaptor signature scheme $\Xi_{R,\Sigma}$ satisfies pre-signature correctness if for every $\lambda \in \mathbb{N}$, every message $m \in \{0, 1\}^*$ and every statement/witness pair $(Y, y) \in R$, the following holds:*

$$\Pr\left[\begin{array}{c}\mathsf{PreVf}(\mathsf{pk}, m, Y, \hat{\sigma}) = 1\\ \wedge\\ \mathsf{Vf}(\mathsf{pk}, m, \sigma) = 1\\ \wedge\\ (Y, y') \in R\end{array} \;\middle|\; \begin{array}{l}(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(1^\lambda)\\ \hat{\sigma} \leftarrow \mathsf{PreSig}(\mathsf{sk}, m, Y)\\ \sigma := \mathsf{Adapt}(\hat{\sigma}, y)\\ y' := \mathsf{Ext}(\sigma, \hat{\sigma}, Y)\end{array}\right] = 1.$$

Next, we define the security properties of an adaptor signature scheme. We start with the notion of unforgeability, which is similar to existential unforgeability under chosen message attacks (EUF-CMA) but additionally requires that producing a forgery σ for some message m is hard even given a pre-signature on m w.r.t. a random statement $Y \in L_R$. We note that allowing the adversary to learn a pre-signature on the forgery message m is crucial as for our applications unforgeability needs to hold even in case the adversary learns a pre-signature for m without knowing a witness for Y. We now formally define the existential unforgeability under chosen message attack for adaptor signature (aEUF-CMA).

Definition 9 (aEUF-CMA Security)**.** *An adaptor signature scheme $\Xi_{R,\Sigma}$ is* aEUF-CMA *secure if for every* PPT *adversary $\mathcal{A}$ there exists a negligible function* negl *such that:*

$\Pr[\mathsf{aSigForge}_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$*, where the experiment* $\mathsf{aSigForge}_{\mathcal{A},\Xi_{R,\Sigma}}$ *is defined as follows:*

$\mathsf{aSigForge}_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda)$	$\mathcal{O}_{\mathrm{S}}(m)$
$1: \mathcal{Q} := \emptyset$	$1: \sigma \leftarrow \mathsf{Sig}(\mathsf{sk}, m)$
$2: (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(1^\lambda)$	$2: \mathcal{Q} := \mathcal{Q} \cup \{m\}$
$3: m \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{S}}(\cdot),\mathcal{O}_{\mathrm{pS}}(\cdot,\cdot)}(\mathsf{pk})$	$3:$ **return** σ
$4: (Y, y) \leftarrow \mathsf{GenR}(1^\lambda)$	$\mathcal{O}_{\mathrm{pS}}(m, Y)$
$5: \hat{\sigma} \leftarrow \mathsf{PreSig}(\mathsf{sk}, m, Y)$	$1: \hat{\sigma} \leftarrow \mathsf{PreSig}(\mathsf{sk}, m, Y)$
$6: \sigma \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{S}}(\cdot),\mathcal{O}_{\mathrm{pS}}(\cdot,\cdot)}(\hat{\sigma}, Y)$	$2: \mathcal{Q} := \mathcal{Q} \cup \{m\}$
$7:$ **return** $(m \notin \mathcal{Q} \wedge \mathsf{Vf}(\mathsf{pk}, m, \sigma))$	$3:$ **return** $\hat{\sigma}$

An additional property that we require from adaptor signatures is *pre-signature adaptability*, which states that any valid pre-signature w.r.t. Y (possibly produced by a malicious signer) can be adapted into a valid signature using the witness y with $(Y, y) \in R$. We note that this property is stronger than the pre-signature correctness property from Definition 8, since we require that even maliciously produced pre-signatures can always be completed into valid signatures. The following definition formalizes this property.

Definition 10 (Pre-signature Adaptability)**.** *An adaptor signature scheme* $\Xi_{R,\Sigma}$ *satisfies pre-signature adaptability if for any* $\lambda \in \mathbb{N}$*, any message* $m \in \{0,1\}^*$*, any statement/witness pair* $(Y, y) \in R$*, any key pair* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(1^\lambda)$ *and any pre-signature* $\hat{\sigma} \leftarrow \{0,1\}^*$ *with* $\mathsf{PreVf}(\mathsf{pk}, m, Y, \hat{\sigma}) = 1$*, we have:* $\Pr[\mathsf{Vf}(\mathsf{pk}, m, \mathsf{Adapt}(\hat{\sigma}, y)) = 1] = 1$.

The last property that we are interested in is *witness extractability*. Informally, it guarantees that a valid signature/pre-signatue pair $(\sigma, \hat{\sigma})$ for a message/statement pair (m, Y) can be used to extract the corresponding witness y of Y.

Definition 11 (Witness Extractability)**.** *An adaptor signature scheme* $\Xi_{R,\Sigma}$ *is* witness extractable *if for every* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that the following holds:* $\Pr[\mathsf{aWitExt}_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$*, where the experiment* $\mathsf{aWitExt}_{\mathcal{A},\Xi_{R,\Sigma}}$ *is defined as follows*

$\mathsf{aWitExt}_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda)$	$\mathcal{O}_{\mathrm{S}}(m)$
$1: \mathcal{Q} := \emptyset$	$1: \sigma \leftarrow \mathsf{Sig}(\mathsf{sk}, m)$
$2: (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(1^\lambda)$	$2: \mathcal{Q} := \mathcal{Q} \cup \{m\}$
$3: (m, Y) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{S}}(\cdot),\mathcal{O}_{\mathrm{pS}}(\cdot,\cdot)}(\mathsf{pk})$	$3:$ **return** σ
$4: \hat{\sigma} \leftarrow \mathsf{PreSig}(\mathsf{sk}, m, Y)$	$\mathcal{O}_{\mathrm{pS}}(m, Y)$
$5: \sigma \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{S}}(\cdot),\mathcal{O}_{\mathrm{pS}}(\cdot,\cdot)}(\hat{\sigma})$	$1: \hat{\sigma} \leftarrow \mathsf{PreSig}(\mathsf{sk}, m, Y)$
$6: y' := \mathsf{Ext}(\mathsf{pk}, \sigma, \hat{\sigma}, Y)$	$2: \mathcal{Q} := \mathcal{Q} \cup \{m\}$
$7:$ **return** $(m \notin \mathcal{Q} \wedge (Y, y') \notin R$	$3:$ **return** $\hat{\sigma}$
$8: \quad \wedge \mathsf{Vf}(\mathsf{pk}, m, \sigma))$	

Although, the witness extractability experiment aWitExt looks similar to the experiment aSigForge, there is one important difference, namely, the adversary is allowed to choose the forgery statement Y. Hence, we can assume that the adversary knows a witness for Y, and therefore, can generate a valid signature on the forgery message m. However, this is not sufficient to win the experiment. The adversary wins *only* if the valid signature does not reveal a witness for Y.

Combining the three properties described above, we can define a secure adaptor signature scheme as follows.

Definition 12 (Secure Adaptor Signature Scheme)**.** *An adaptor signature scheme* $\Xi_{R,\Sigma}$ *is secure, if it is* aEUF-CMA *secure, pre-signature adaptable and witness extractable.*

E. Castagnos-Laguillaumie Encryption Scheme

The main reason for using the Castagnos-Laguillaumie (CL) [37], [38] encryption scheme as opposed to any other linearly homomorphic encryption scheme is that it can be instantiated to work over $\mathbb{Z}_q$, for a q that is the same as the order of the elliptic curve group used in Schnorr and ECDSA signature schemes. If one uses an encryption scheme with a plaintext space larger than the group order q, then several problems appear. For example, two- party ECDSA construction of Lindell [56] uses Paillier, which has a plaintext space $\mathbb{Z}_N$, for a composite N much larger than q. In that case to enforce correctness and security of the protocol the value of N needs to be chosen large enough, so that no wrap around occurs, and one needs to prove in zero-knowledge that the encrypted value is within the right range, which requires an expensive range proof. We can avoid these issues by using the CL encryption scheme instantiated with the plaintext space $\mathbb{Z}_q$. Another advantage of CL is that in the security proofs challenger's access to the secret key does not compromise the indistinguishability of ciphertexts, as it relies on a computational assumption and a statistical argument. For more information about the problems arising from using an encryption scheme with a larger modulus than the elliptic curve group order, and how these problems are addressed by the CL encryption scheme we refer the reader to [38].

F. Full Security Analysis

1) Security Analysis of Randomizable Puzzle: We recall the theorem stated in Section V-B, for which we provide a proof sketch here.

Theorem 1. *Let* $\mathbb{G}$ *be a DLOG-hard group, and* Ψ *be an* IND-CPA *secure encryption scheme, then Construction 1 is a correct, secure and private randomizable puzzle scheme.*

Proof (sketch). Correctness follows straightforwardly from the correctness of the encryption scheme Ψ. For security, we first replace $\mathsf{Enc}(\mathsf{pk}^\Psi, \zeta)$ with $\mathsf{Enc}(\mathsf{pk}^\Psi, 0)$ in PGen. This is indistinguishable due to IND-CPA security of Ψ. Next, what remains is to argue that one cannot retrieve the discrete logarithm of A, which is implied by the hardness of DLOG in $\mathbb{G}$. Regarding privacy, we can observe that for a puzzle Z with a solution ζ, our PRand algorithm will produce a randomized puzzle Z' with a solution $\zeta \cdot r$, for a random $r \in \mathcal{S}$. Note that in our case $\mathcal{S} = \mathbb{Z}_q$ is a field, hence, we have that $\zeta \cdot r \in \mathcal{S}$. Moreover, the randomness r completely masks the solution ζ. Hence, a randomized puzzle Z' is equally likely to be the randomized version of any puzzle Z, which implies that Z and Z' are information-theoretically unlinkable. $\square$

2) Security Analysis of A^2L : Throughout this section we denote by $\mathsf{poly}(\lambda)$ any function that is bounded by a polynomial in λ, where $\lambda \in \mathbb{N}$ is the security parameter. We denote any function that is negligible in the security parameter by $\mathsf{negl}(\lambda)$. We say an algorithm is PPT if it is modeled as a probabilistic Turing machine whose running time is bounded by some function $\mathsf{poly}(\lambda)$.

We prove security according to the UC framework [31], and in the presence of *malicious adversaries* with *static corruptions*. We recall the theorem stated in Section VII, which we prove here.

Theorem 2. *Let* COM *be a secure commitment scheme,* NIZK *be a non-interactive zero-knowledge scheme,* $\Sigma, \widetilde{\Sigma}$ *be* EUF-CMA *secure signature schemes, R be a hard relation, $\Xi_{R,\Sigma}$ be a secure adaptor signature scheme, and* RP *be a secure and private randomizable puzzle scheme, then the construction in Figures 2 to 5 UC-realizes the ideal functionality $\mathcal{F}_{\mathsf{A^2L}}$ in the $(\mathcal{F}_{\mathsf{GDC}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{anon}})$-hybrid model.*

Proof. Throughout the following proof, we implicitly assume that all messages of the adversary are well-formed and we treat the malformed messages as aborts. The proof is composed of a series of hybrids, where we gradually modify the initial experiment.

Hybrid $\mathcal{H}_0$: This corresponds to the original construction (as described in Section VI-A).

Hybrid $\mathcal{H}_1$: All calls to the commitment scheme COM are replaced with calls to the ideal functionality $\mathcal{F}_{\mathsf{COM}}$.

Ideal Functionality $\mathcal{F}_{\mathsf{COM}}$

Commit: On input $(\mathsf{commit}, \mathsf{sid}, x)$ from party P_i, where $i \in \{1,2\}$, if some $(\mathsf{commit}, \mathsf{sid}, \cdot)$ is already recorded, then ignore the message. Else, record (sid, i, x) and send $(\mathsf{receipt}, \mathsf{sid})$ to party P_{3-i}.
Decommit: On input $(\mathsf{decommit}, \mathsf{sid})$ from party P_i, where $i \in \{1,2\}$, if (sid, i, x) is recorded, then send $(\mathsf{decommit}, \mathsf{sid}, x)$ to party P_{3-i}.

Hybrid $\mathcal{H}_2$: All calls to the non-interactive zero-knowledge scheme NIZK are replaced with calls to the ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$, which works with a relation R.

Ideal Functionality $\mathcal{F}_{\mathsf{NIZK}}$

On input $(\mathsf{prove}, \mathsf{sid}, x, w)$ from party P_i, where $i \in \{1,2\}$, if $(x, w) \notin R$ or sid has been previously used, then ignore the message. Otherwise, send $(\mathsf{proof}, \mathsf{sid}, x)$ to P_{3-i}.

Hybrid $\mathcal{H}_3$: For an honest tumbler P_t and sender P_s, a corrupted receiver P_r, check if P_r returns some pair $(\mathsf{tid}, \sigma_{\mathsf{tid}})$, before an execution of the registration protocol (between P_t and P_r), such that it does not cause the honest P_t to abort during the promise protocol. If this is the case, abort the experiment and output fail.

Hybrid $\mathcal{H}_4$: For an honest tumbler P_t and sender P_s, a corrupted receiver P_r and a promise Π output from the puzzle promise protocol, if P_r returns some $\sigma := (\sigma_t, \sigma_r)$, such that $\mathsf{Verify}(\Pi, \sigma) = 1$, before a solution α' is output from an execution of the puzzle solver protocol, such that $\mathsf{Verify}(\Pi, \mathsf{Open}(\Pi, \alpha')) = 1$, then the experiment aborts.

Hybrid $\mathcal{H}_5$: For an honest sender P_s and receiver P_r, a promise Π output from the puzzle promise protocol and a solution α' output from the puzzle solver protocol, if the parties do not abort and $\mathsf{Verify}(\Pi, \mathsf{Open}(\Pi, \alpha')) \neq 1$, then the experiment aborts.

Simulator $\mathcal{S}$: The simulator $\mathcal{S}$ simulates the honest parties as in the previous hybrid, except that its actions are dictated by the interaction with the ideal functionality $\mathcal{F}_{\mathsf{A^2L}}$. More concretely, we define our simulator $\mathcal{S}$ as follows.

Simulator for registration

Case P_s is honest and P_t is corrupted

Upon P_s sending $(\mathsf{Registration}, P_r)$ to $\mathcal{F}_{\mathsf{A^2L}}$, proceed as follows:

- Sample a token $\mathsf{tid} \leftarrow\!\!\$\, \mathbb{Z}_q$ and output $\mathsf{oid} \leftarrow\!\!\$\, \{0,1\}^*$, commit to the token and prove knowledge of the opening,

$$(\mathsf{com}, \mathsf{decom} := (\mathsf{tid}, r)) \leftarrow \mathsf{P_{COM}}(\mathsf{tid}),$$
$$\pi \leftarrow \mathsf{P_{NIZK}}(\{\exists \mathsf{decom} \mid \mathsf{V_{COM}}(\mathsf{com}, \mathsf{decom}, \mathsf{tid}) = 1\}, \mathsf{decom}\},$$

  and send $(\mathsf{registration{-}req}, ((\pi, \mathsf{com}), \mathsf{oid}))$ to P_t.
- Upon $(\mathsf{registered}, \sigma^*)$ from $\mathcal{A}$ (on behalf of P_t), unblind the signature, $\sigma_{\mathsf{tid}} := \mathsf{UnblindSig}(\mathsf{decom}, \sigma^*)$. If $\mathsf{Vf}(\mathsf{pk}_t^{\widetilde{\Sigma}}, \mathsf{tid}, \sigma_{\mathsf{tid}}) \neq 1$, then simulate P_s aborting. Otherwise, randomize the signature, $\sigma'_{\mathsf{tid}} \leftarrow \mathsf{RandSig}(\sigma_{\mathsf{tid}})$, store a copy of $(\mathsf{tid}, \sigma'_{\mathsf{tid}})$ and send it to P_r.

Case P_t is honest and P_s is corrupted

Upon P_s sending $(\mathsf{registration{-}req}, ((\pi, \mathsf{com}), \mathsf{oid}))$ to P_t, proceed as follows:

- If $\mathsf{V_{NIZK}}(\pi, \mathsf{com}) \neq 1$, then simulate P_t aborting. Otherwise, if P_t sends $(\mathsf{registration{-}res}, \top)$ to $\mathcal{F}_{\mathsf{A^2L}}$, then compute $\sigma^* \leftarrow \mathsf{BlindSig}(\mathsf{sk}_t^{\widetilde{\Sigma}}, \mathsf{com})$, and send $(\mathsf{registered}, \sigma^*)$ to P_s. Else stop.

Simulator for puzzle promise

Case P_r is honest and P_t is corrupted

Upon P_r sending $(\mathsf{PuzzlePromise}, P_s, \mathsf{tid})$ to $\mathcal{F}_{\mathsf{A^2L}}$, proceed as follows:

- Extract $(\mathsf{tid}, \sigma'_{\mathsf{tid}})$ that was previously stored, sign the message (transaction), $\sigma'_r \leftarrow \mathsf{Sig}(\mathsf{sk}_r^{\Sigma}, m')$, and send $(\mathsf{promise{-}req}, ((\mathsf{tid}, \sigma'_{\mathsf{tid}}), \sigma'_r))$ to P_t.
- Upon $(\mathsf{promise}, (Z := (A, c_\alpha), \pi_\alpha, \hat{\sigma}'_t))$ from $\mathcal{A}$ (on behalf of P_t), check if $\mathsf{V_{NIZK}}(\pi_\alpha, Z) \neq 1$ or $\mathsf{PreVf}(\mathsf{pk}_t^{\Sigma}, m', A, \hat{\sigma}'_t) \neq 1$. If this is the case, then simulate P_r aborting. Otherwise, randomize the puzzle $(Z', \beta) \leftarrow \mathsf{PRand}(\mathsf{pp}, Z)$, store $\Pi := (\beta, (\mathsf{pk}_t^{\Sigma}, \mathsf{pk}_r^{\Sigma}), m', (\hat{\sigma}'_t, \sigma'_r))$, $Z := (A, c_\alpha)$ and $Z' := (A', c'_\alpha)$, and send Z' to P_s.

Case P_t is honest and P_r is corrupted

Upon P_r sending $(\mathsf{promise\!-\!req}, ((\mathsf{tid}, \sigma'_{\mathsf{tid}}), \sigma'_r))$ to P_t, proceed as follows:

- If $\mathsf{tid} \in \mathcal{T}$ or $\mathsf{Vf}(\mathsf{pk}_t^{\widetilde{\Sigma}}, \mathsf{tid}, \sigma'_{\mathsf{tid}}) \neq 1$, then simulate P_t aborting. Otherwise, if P_t sends $(\mathsf{promise\!-\!res}, \top)$ to $\mathcal{F}_{\mathsf{A^2L}}$, then add tid into $\mathcal{T}$, generate new statement/witness pair $(A, \alpha) \leftarrow \mathsf{GenR}(1^\lambda)$, generate a new puzzle using α as the solution and prove its correctness,

$$Z \leftarrow \mathsf{PGen}(\mathsf{pp}, \alpha),$$
$$\pi_\alpha \leftarrow \mathsf{P_{NIZK}}(\{\exists \alpha \mid \mathsf{PSolve}(\mathsf{td}, Z) = \alpha\}, \alpha),$$

pre-sign the message (transaction) $\hat{\sigma}'_t \leftarrow \mathsf{PreSig}(\mathsf{sk}_t^\Sigma, m', A)$, and send $(\mathsf{promise}, (Z := (A, c_\alpha), \pi_\alpha, \hat{\sigma}'_t))$ to P_s. Else stop.

Simulator for puzzle solver

Case P_s is honest and P_t is corrupted

Upon P_r sending $(\mathsf{PuzzleSolver}, P_r, \mathsf{pid})$ to $\mathcal{F}_{\mathsf{A^2L}}$, proceed as follows:

- Extract $Z' := (A', c'_\alpha)$ that was previously stored, randomize the puzzle $(Z'', \tau) \leftarrow \mathsf{PRand}(\mathsf{pp}, Z')$, such that $Z'' := (A'', c''_\alpha)$, pre-sign the message (transaction), $\hat{\sigma}_s \leftarrow \mathsf{PreSig}(\mathsf{sk}_s^\Sigma, m, A'')$. Send $(\mathsf{solve\!-\!req}, (Z'', \hat{\sigma}_s))$ to P_t.
- Upon $(\mathsf{solve}, \sigma_s)$ from $\mathcal{A}$ (on behalf of P_t), extract the witness, $\alpha'' \leftarrow \mathsf{Ext}(\sigma_s, \hat{\sigma}_s, A'')$, and if $\alpha'' = \perp$, then simulate P_s aborting. Otherwise, compute $\alpha' \leftarrow \alpha'' \cdot \tau^{-1}$ and send α' to P_r.

Case P_t is honest and P_s is corrupted

Upon P_s sending $(\mathsf{solve\!-\!req}, (Z'', \hat{\sigma}_s))$ to P_t, proceed as follows:

- Solve the puzzle, adapt the input pre-signature, and sign the message (transaction),

$$\alpha'' := \mathsf{PSolve}(\mathsf{td}, Z'')$$
$$\sigma_s := \mathsf{Adapt}(\hat{\sigma}_s, \alpha''),$$
$$\sigma_t \leftarrow \mathsf{Sig}(\mathsf{sk}_t^\Sigma, m).$$

If $\mathsf{Vf}(\mathsf{pk}_s^\Sigma, m, \sigma_s) \neq 1$, then simulate P_t aborting. Otherwise, if P_t sends $(\mathsf{solve\!-\!res}, \top)$ to $\mathcal{F}_{\mathsf{A^2L}}$, then send σ_s to P_s. Else stop.

Next, we proceed to proving the indistinguishability of the neighboring experiments for the environment $\mathcal{E}$.

Lemma 1. *For all* PPT *distinguishers* $\mathcal{E}$ *it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_0,\mathcal{A},\mathcal{E}} \approx \mathsf{EXEC}_{\mathcal{H}_1,\mathcal{A},\mathcal{E}}.$$

Proof. The proof follows directly from the security of the commitment scheme COM. □

Lemma 2. *For all* PPT *distinguishers* $\mathcal{E}$ *it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_1,\mathcal{A},\mathcal{E}} \approx \mathsf{EXEC}_{\mathcal{H}_2,\mathcal{A},\mathcal{E}}.$$

Proof. The proof follows directly from the security of the non-interactive zero-knowledge scheme NIZK. □

Lemma 3. *For all* PPT *distinguishers* $\mathcal{E}$ *it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_2,\mathcal{A},\mathcal{E}} \approx \mathsf{EXEC}_{\mathcal{H}_3,\mathcal{A},\mathcal{E}}.$$

Proof. We note that the two hybrids differ if the experiment outputs fail, hence, it suffices to bound the probability that such an event occurs. Observe that the event fail happens in the case that an honest tumbler P_t does not abort the puzzle promise protocol when executed with a token not obtained from the registration protocol. We can bound the probability that this happens by a reduction against the existential unforgeability of the randomizable signature scheme $\widetilde{\Sigma}$. Assume towards contradiction that $\Pr[\mathsf{fail} \mid \mathcal{H}_2] \geq \frac{1}{\mathsf{poly}(\lambda)}$, then we can construct the following reduction. The reduction receives as input a public key pk, and samples an index $j \in [1, q]$, where $q \in \mathsf{poly}(\lambda)$ is a bound on the total number of interactions. The reduction sets the public key $\mathsf{pk}_t^{\widetilde{\Sigma}}$ generated in the j-th interaction to the challenge pk. All calls to the signing algorithm are redirected to the signing oracle. If the registration procedure is called, then the reduction aborts. If the event fail happens, then the reductions returns the corresponding $(\mathsf{tid}^*, \sigma^*_{\mathsf{tid}})$, otherwise it aborts.

The reduction is clearly efficient, and whenever j is guessed correctly, the reduction does not abort. Since fail happens it means that the registration protocol is not executed, and puzzle promise protocol is called with $(\mathsf{tid}^*, \sigma^*_{\mathsf{tid}})$ as input, and furthermore, we have that $\mathsf{Vf}(\mathsf{pk}_t^{\widetilde{\Sigma}}, \mathsf{tid}^*, \sigma^*_{\mathsf{tid}}) = 1$ and $\mathsf{tid}^* \notin \mathcal{T}$, which implies that P_t does not abort the execution of the puzzle promise. As the size of $\mathcal{T}$ is $\mathsf{poly}(\lambda)$ bounded and the token space is $\mathbb{Z}_q$ (for a prime q at least λ bits), we have that $\Pr[\mathsf{tid}^* \notin \mathcal{T} \mid \mathsf{tid}^* \leftarrow_{\$} \mathbb{Z}_q] = 1 - \frac{|\mathcal{T}|}{|\mathbb{Z}_q|}$, which is overwhelming. However, as every message (token identifier) uniquely identifies a session, we have that $(\mathsf{tid}^*, \sigma^*_{\mathsf{tid}})$ is a valid forgery. By assumption this happens with probability at least $\frac{1}{q \cdot \mathsf{poly}(\lambda)}$, which is a contradiction and proves that $\Pr[\mathsf{fail} \mid \mathcal{H}_2] \leq \mathsf{negl}(\lambda)$. □

Lemma 4. *For all* PPT *distinguishers* $\mathcal{E}$ *it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_3,\mathcal{A},\mathcal{E}} \approx \mathsf{EXEC}_{\mathcal{H}_4,\mathcal{A},\mathcal{E}}.$$

Proof. Let fail be the event that triggers an abort in $\mathcal{H}_4$ but not in $\mathcal{H}_3$. In the following we are going to show that the probability that such an event happens can be bounded by a negligible function in the security parameter. Assume towards contradiction that $\Pr[\mathsf{fail} \mid \mathcal{H}_3] \geq \frac{1}{\mathsf{poly}(\lambda)}$. To show that the the probability of fail happening in $\mathcal{H}_3$ cannot be inverse polynomial we need to reduce it to the security of the RP scheme, hardness of the relation R and unforgeability of the adaptor signature scheme $\Xi_{R,\Sigma}$. In Section V-B we already proved the security of our RP construction, which relied on the indistinguihsability of the homomorphic encryption scheme Ψ and hardness of DLOG, which also corresponds to our hard relation R. Hence, we know that the adversary's advantage in breaking the security of the RP is $\mathsf{negl}(\lambda)$. Moreover, the security of RP and the unforgeability of the adaptor signature also implies the hardness of the relation R, therefore, all that remains for us is to show that the probability of fail happening in $\mathcal{H}_3$ cannot be inverse polynomial via a reduction to the unforgeability of the adaptor signature scheme $\Xi_{R,\Sigma}$. The reduction receives as input a public key pk, pre-signature $\hat{\sigma}$ and a statement Y, and samples an index $j \in [1, q]$, where $q \in \mathsf{poly}(\lambda)$ is a bound on the total number of interactions. The reduction replaces $\hat{\sigma}'_t$ with $\hat{\sigma}$ and A with Y in the puzzle promise, and sets the public key pk_t^Σ generated in the j-th interaction to pk. All calls to the pre-signing and signing algorithm are redirected to the pre-signing and signing oracles, respectively. If the puzzle solver procedure is called, then the reduction aborts. If the event fail happens, then the reduction returns the corresponding $\sigma^* := (\sigma^*_t, \sigma^*_r)$, otherwise it aborts.

The reduction is clearly efficient, and whenever j is guessed correctly, the reduction does not abort. Since fail happens we have that $\mathsf{Verify}(\Pi, \sigma^*) = 1$ and the puzzle solver protocol is not executed. Recall that P_r is corrupted, and hence, σ^*_r is computed honestly with sk^Σ_r as in the protocol, which implies that $\sigma_r = \sigma^*_r$. Therefore, what remains to show is that σ^*_t is a valid forgery under pk^Σ_t, which follows from the fact that every message uniquely identifies a session (so the message is not queried before). However, by assumption this happens with probability at least $\frac{1}{q \cdot \mathsf{poly}(\lambda)}$, which is a contradiction and proves that $\Pr[\mathsf{fail} \mid \mathcal{H}^*_3] \leq \mathsf{negl}(\lambda)$. □

Lemma 5. *For all* PPT *distinguishers* $\mathcal{E}$ *it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_4,\mathcal{A},\mathcal{E}} \approx \mathsf{EXEC}_{\mathcal{H}_5,\mathcal{A},\mathcal{E}}.$$

Proof. Let fail be the event that triggers an abort in $\mathcal{H}_5$ but not in $\mathcal{H}_4$. We note that such an event can happen in two scenarios. First, if a corrupted P_t comes up with a pre-signature $\hat{\sigma}'_t$ during the puzzle promise protocol, which succeeds in pre-verification under the key pk^Σ_t, but then adapting this pre-signature inside Open produces an invalid signature. Second, if a corrupted P_t produces a valid signature σ_s during the puzzle solver protocol, which when extracted outputs an invalid witness. However, if the former happens, then we have an adversary against the pre-signature adaptability, and if the latter happens, then we have an adversary against the witness extractability of the adaptor signature scheme $\Xi_{R,\Sigma}$. In the following we are going to show that the probability that such an event happens can be bounded by a negligible function in the security parameter. Assume towards contradiction that $\Pr[\mathsf{fail} \mid \mathcal{H}_5] \geq \frac{1}{\mathsf{poly}(\lambda)}$, and consider the following intermediate hybrid.

- Hybrid $\mathcal{H}^*_4$: The pre-signature in the puzzle promise protocol is set to $\hat{\sigma}'_t \leftarrow\!\!\$\, \{0,1\}^*$, such that pre-verification of $\hat{\sigma}'_t$ succeeds under the public key pk^Σ_t.

By the pre-signature adaptability property of the adaptor signature scheme $\Xi_{R,\Sigma}$ we have that

$$\Pr[\mathsf{fail} \mid \mathcal{H}^*_4] = \Pr[\mathsf{fail} \mid \mathcal{H}_4].$$

At this point all that remains is to show that the probability of fail happening in $\mathcal{H}^*_4$ cannot be inverse polynomial. This is done via the following reduction against the witness extractability of the adaptor signature scheme $\Xi_{R,\Sigma}$. Assume towards contradiction that $\Pr[\mathsf{fail} \mid \mathcal{H}^*_4] \geq \frac{1}{\mathsf{poly}(\lambda)}$, then we can construct the following reduction. The reduction receives as input a public key pk and a pre-signature $\hat{\sigma}$. It samples an index $j \in [1, q]$, where $q \in \mathsf{poly}(\lambda)$ is bound on the total number of interactions. The reduction replaces the pre-signature $\hat{\sigma}_s$ from the puzzle solver protocol with $\hat{\sigma}$ and sets the public key pk^Σ_s generated in the j-th interaction to pk. All the calls to the pre-signing and signing algorithms are redirected to the pre-signing and signing oracles, respectively. If the event fail happens, then the reductions returns the signature σ_s of P_s, and otherwise it aborts.

The reduction is clearly efficient, and whenever j is guessed correctly, the reduction does not abort. Since fail happens we have that no party aborted, but $\mathsf{Verify}(\Pi, \mathsf{Open}(\Pi, \alpha')) \neq 1$. Recall that the open algorithm parses Π as $(\beta, (\mathsf{pk}^\Sigma_t, \mathsf{pk}^\Sigma_r), m', (\hat{\sigma}'_t, \sigma'_r))$, computes $\sigma'_t := \mathsf{Adapt}(\hat{\sigma}'_t, \alpha)$, for $\alpha = \alpha' \cdot \beta^{-1}$, and returns $\sigma := (\sigma'_t, \sigma'_r)$. Since P_r is honest we have that σ_r is honestly generated and its verification succeeds. Hence, it remains to show that the computed σ'_t is invalid. From the intermediate hybrid $\mathcal{H}^*_4$ and the pre-siganture adaptability property of the adaptor signature scheme $\Xi_{R,\Sigma}$, we know that the adapt algorithm works as expected. This implies that the only way we can have an invalid σ'_t is if the computed α is not a valid witness the statement A. We have that $\alpha = \alpha'' \cdot \tau^{-1} \cdot \beta^{-1}$, and since P_s and P_r are honest, this implies that the extracted α'' is invalid (i.e., is not a witness of A''). Hence, σ_t is a valid signature that does not reveal a witness for A''. However, by assumption this happens with probability at least $\frac{1}{q \cdot \mathsf{poly}(\lambda)}$, which is a contradiction and proves that $\Pr[\mathsf{fail} \mid \mathcal{H}^*_4] \leq \mathsf{negl}(\lambda)$. □

Lemma 6. *For all* PPT *distinguishers* $\mathcal{E}$ *it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_5,\mathcal{A},\mathcal{E}} \approx \mathsf{EXEC}_{\mathcal{F}_{\mathsf{A^2L}},\mathcal{S},\mathcal{E}}.$$

Proof. The two experiments are identical, and the change here is only syntactical. Hence, indistinguishability follows. □

This concludes the proof of Theorem 2.

□

3) Security Analysis of PCH: Here we prove the following theorem about our PCH construction, which was previously stated in Section VII

Theorem 3. *The protocol in Figure 6, UC-realizes* $\mathcal{F}_{\mathsf{PCH}}$ *in the* $(\mathcal{F}_{\mathsf{GDC}}, \mathcal{F}_{\mathsf{GC}}, \mathcal{F}_{\mathsf{clock}}, \mathcal{F}_{\mathsf{A^2L}})$*-hybrid model.*

Proof. The proof consists of the observation that the ideal functionality $\mathcal{F}_{\mathsf{A^2L}}$ enforces authenticity, atomicity and unlinkability properties of a PCH (that are defined in Section II-B and discussed in Appendix C3). Authenticity guarantees that only payments that were previously backed up by some locked coins are processed. Atomicity guarantees that either all the balances are updated or none of them, which ensures that no party loses or gains more than it should. Both of these properties are satisfied by $\mathcal{F}_{\mathsf{A^2L}}$ as was proven in Appendix F2. Furthermore, as was discussed in Appendix C2, $\mathcal{F}_{\mathsf{A^2L}}$ also satisfies the unlinkability property, hence, the same argument for unlinkability applies here too, with the exception of the operations of $\mathcal{F}_{\mathsf{PCH}}$ that are outside $\mathcal{F}_{\mathsf{A^2L}}$. However, we note that the only information that is sent outside of $\mathcal{F}_{\mathsf{A^2L}}$ consists of amounts and timeouts, and since we use constant amounts along with synchronized phases and epochs, this information by itself does not break our unlinkability notion. Moreover, here the job of the simulator $\mathcal{S}$ consists of interacting with $\mathcal{F}_{\mathsf{A^2L}}$ and $\mathcal{F}_{\mathsf{GC}}$ ideal functionalities on behalf of $\mathcal{F}_{\mathsf{PCH}}$, which is trivial to realize.

□

G. Threshold Variants

We present here variants of our construction that are based on 2-of-2 threshold signatures. Hence, at the end of the protocols the parties obtain a single signature. We note that our registration protocol does not depend on the underlying signature scheme used, therefore, it remains unchanged from Figure 5.

1) Schnorr-based Construction: Let $\mathbb{G}$ be an elliptic curve group of prime order q with a generator g, and let $H : \{0,1\}^* \to \mathbb{Z}_q$ be a collision resistant hash function. Additionally, let COM and NIZK and be a commitment scheme and a non-interactive zero-knowledge scheme, respectively. The Schnorr-based puzzle promise and puzzle solver protocols are shown in Figure 9 and Figure 10, respectively.

The construction requires that the parties have generated shared Schnorr public keys (i.e., pk_{tr} between P_t and P_r to be used during puzzle promise, and pk_{st} between P_s and P_t to be used during puzzle solver). This shared key generation can be done as explained in [40].

The puzzle promise protocol is run between the tumbler P_t and the receiver P_r as before. They initially agree on a message encoding a transaction that transfers coins from P_t to P_r. Additionally, P_t chooses a secret value α, generates the puzzle Z using α and sends it to P_r (lines 5-11 in Figure 9). Here we require a zero-knowledge proof (denoted by π_α in the puzzle promise protocol) proving that the puzzle has a correct solution (i.e., ciphertext c_α encrypts the discrete logarithm of A for our randomizable puzzle construction from Section V-B) (line 8 in Figure 9). If we do not have such a proof, then P_t can perform the following attack to link a potential honest payer and payee. At a particular epoch, P_t chooses a payee P_r^* it wants to attack, and when performing the puzzle promise protocol with this party it replaces the value A from the puzzle Z with a random group element. Then, during the puzzle solver protocol, when a payer P_s^* performs the protocol with P_t, the check $A'' = g^{\alpha''}$ (line 12 in Figure 10) will fail, and P_t^* will cause an abort. Although, in this case (due to our atomicity property) no payment will go through, P_t can still link a payee P_r^* of its choice with its corresponding potential payer P_s^* in a given epoch.

Next, the parties execute a coin tossing protocol to agree on a randomness $R' = k_1' + k_2' + \alpha$, where α is unknown to P_r. The randomness here is composed additively due to the linear structure of Schnorr. The randomness R' is computed by parties exchanging $g^{k_1'}$ and $g^{k_2'}$, and additionally making use of the value A. The computation of R' together with the corresponding consistency proof is piggybacked in the coin tossing (lines 5-13 in Figure 9). At this point, P_t computes its side of the two-party Schnorr signature, but does not include the secret α into the signature (line 14 in Figure 9). Now, P_r is able to validate this partial signature that it receives from P_t, and also to compute an "almost valid" signature by performing its part of the two-party signature. This means that P_r computes a tuple $(e', s' := k_1' + k_2' - e' \cdot (x_1' + x_2'))$, and that the complete signature is of the form $(e', s' + \alpha)$ (lines 18-21 in Figure 9). However, P_r does not have α, so it cannot complete the signature. Nevertheless, P_r receives the puzzle $Z := (A, c_\alpha)$ from P_t at the beginning of the puzzle promise protocol, and at the end of the protocol P_r randomizes Z as $(Z', \beta) \leftarrow \mathsf{PRand}(\mathsf{pp}, Z)$. The puzzle promise protocol finishes with P_r sending these randomized puzzle Z' to P_s (lines 25 and 27 in Figure 9).

The puzzle solver protocol is executed between the sender P_s and the tumbler P_t. At the beginning of the protocol, P_s randomizes the puzzle Z' it received from P_r, as $(Z'', \tau) \leftarrow \mathsf{PRand}(\mathsf{pp}, Z')$ (line 8 in Figure 10). Once this is done, P_s and P_t perform a coin tossing protocol similar to the one performed between P_r and P_t in the puzzle promise protocol, but additionally P_s sends the randomized puzzle Z'' to P_t (lines 2-9 in Figure 10). At this point, P_t solves the randomized puzzle Z'' using its trapdoor td to obtain the value $\alpha'' := \alpha \cdot \beta \cdot \tau$ (line 11 in Figure 10). The rest of the protocol continues similar to the puzzle promise protocol, where P_t and P_s compute a common randomness, and then perform a two-party Schnorr signature. However, this time P_t incorporates the decrypted value α'' as part of the randomness. After the two-party Schnorr signature completes and P_t publishes it (allowing P_t to receive the payment from P_s), P_s is able to extract the value α'' from the published signature (lines 25-26 in Figure 10). It removes her part of the randomization from α'' as $\alpha' \leftarrow \alpha'' \cdot \tau^{-1}$, and sends this value to P_r (lines 27-28 in Figure 10), who can also remove its part of the randomization and obtain the initial $\alpha \leftarrow \alpha' \cdot \beta^{-1}$. Once P_r obtains α, it can complete the "almost valid" signature that it computed at the end of the puzzle promise protocol, as seen in Figure 11, which allows it to claim the coins that were promised by P_t.

2) ECDSA-based Construction: The ECDSA signature does not have a linear structure as Schnorr, which makes the design of our protocol more challenging.

Let $\mathbb{G}$ be an elliptic curve group of order q with a generator g, and let $H : \{0,1\}^* \to \mathbb{Z}_q$ be a collision resistant hash function. Additionally, let $\mathsf{COM}, \mathsf{NIZK}$, and Ψ be a commitment scheme, a non-interactive zero-knowledge scheme, and a homomorphic encryption scheme, respectively. The ECDSA-based puzzle promise and puzzle solver protocols are shown in Figure 13 and Figure 14, respectively.

Our ECDSA-based instantiation shares similar ideas with our Schnorr-based instantiation. The parties again need to have a shared public keys. However, in order to compute two-party ECDSA signature (as described in [38], [56]), one of the parties need to have in an encrypted form the secret key of the other party. For example, during the puzzle solver protocol we assume that the tumbler P_t has as input a ciphertext $c_{\mathsf{sk}_s^\Sigma}$, which is an encryption of the secret key sk_s^Σ of the sender P_s and that forms the part of the joint public key pk_{st}^Σ computed between P_s and P_t.

The puzzle promise protocol runs similar to the Schnorr-based puzzle promise protocol, except that the randomness is composed multiplicatively due to the structure of ECDSA. More precisely, the parties agree on a randomness $R' =$

Public parameters: group description $(\mathbb{G}, g, q)$, message m'

$\mathsf{PuzzlePromise}_{P_t}((\mathsf{sk}_t^{\Sigma}, \mathsf{pk}_{tr}^{\Sigma}), (\mathsf{pp}, \mathsf{td}), \mathsf{pk}_t^{\tilde{\Sigma}})$		$\mathsf{PuzzlePromise}_{P_r}((\mathsf{sk}_r^{\Sigma}, \mathsf{pk}_{tr}^{\Sigma}), (\mathsf{tid}, \sigma'_{\mathsf{tid}}))$
	$\xleftarrow{(\mathsf{tid}, \sigma'_{\mathsf{tid}})}$	
If $\mathsf{tid} \in \mathcal{T} \ \vee\ \mathsf{Vf}(\mathsf{pk}_t^{\tilde{\Sigma}}, \mathsf{tid}, \sigma'_{\mathsf{tid}}) \neq 1$ then abort		
Else add tid into $\mathcal{T}$		
$\alpha, k'_2 \leftarrow_{\$} \mathbb{Z}_q$		
$R'_2 \leftarrow g^{k'_2};\ Z := (A, c_\alpha) \leftarrow \mathsf{PGen}(\mathsf{pp}, \alpha)$		
$\pi_\alpha \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists \alpha \mid \mathsf{PSolve}(\mathsf{td}, Z) = \alpha\}, \alpha)$		
$\pi'_2 \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists k'_2 \mid R'_2 = g^{k'_2}\}, k'_2)$		
$(\mathsf{com}, \mathsf{decom}) \leftarrow \mathsf{P}_{\mathsf{COM}}((R'_2, \pi'_2))$		
	$\xrightarrow{\mathsf{com}, Z := (A, c_\alpha), \pi_\alpha}$	
		If $\mathsf{V}_{\mathsf{NIZK}}(\pi_\alpha, Z) \neq 1$ then abort
		$k'_1 \leftarrow_{\$} \mathbb{Z}_q; R'_1 \leftarrow g^{k'_1}$
		$\pi'_1 \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists k'_1 \mid R'_1 = g^{k'_1}\}, k'_1)$
	$\xleftarrow{R'_1, \pi'_1}$	
If $\mathsf{V}_{\mathsf{NIZK}}(\pi'_1, R'_1) \neq 1$ then abort		
$R' \leftarrow R'_1 \cdot R'_2 \cdot A; e' := H(R' \Vert \mathsf{pk}_{tr}^{\Sigma} \Vert m')$		
$s'_2 \leftarrow k'_2 - \mathsf{sk}_t^{\Sigma} \cdot e' \mod q$		
	$\xrightarrow{(\mathsf{decom}, R'_2, \pi'_2), s'_2}$	
		If $\mathsf{V}_{\mathsf{COM}}(\mathsf{com}, \mathsf{decom}, (R'_2, \pi'_2)) \neq 1$ then abort
		If $\mathsf{V}_{\mathsf{NIZK}}(\pi'_2, R'_2) \neq 1$ then abort
		$R' \leftarrow R'_1 \cdot R'_2 \cdot A; e' := H(R' \Vert \mathsf{pk}_{tr}^{\Sigma} \Vert m')$
		If $g^{s'_2} \neq R'_2 \cdot (Q'/g^{\mathsf{sk}_r^{\Sigma}})^{-e'}$ then abort
		$s'_1 \leftarrow k'_1 - \mathsf{sk}_r^{\Sigma} \cdot e' \mod q$
		$s' \leftarrow s'_1 + s'_2 \mod q$
		$(Z', \beta) \leftarrow \mathsf{PRand}(\mathsf{pp}, Z)$
	$\xleftarrow{s'}$	
		Send $Z' := (A', c'_\alpha)$ to P_s
If $g^{s'} \neq R'_1 \cdot R'_2 \cdot (\mathsf{pk}_{tr}^{\Sigma})^{-e'}$ then abort		Set $\Pi := (\beta, (\mathsf{pk}_{tr}^{\Sigma}, m', \sigma' := (R', s')))$
return $\sigma := (R', s' + \alpha)$		**return** $(\Pi, (Z, Z'))$

Fig. 9: Puzzle promise protocol of Schnorr-based construction. Blue parts are related to the griefing protection (see Section VI-B)

$k'_1 \cdot k'_2 \cdot \alpha$, where α is unknown to P_r (lines 5-14 in Figure 13). Once the randomness is computed, P_t performs its side of the two-party ECDSA signature using c_{tr}^{Σ} (the encryption of sk_r^{Σ}) and the homomorphic properties of CL encryption scheme. However, P_t does not embed the inverse of α into the signature (lines 19-22 in Figure 13). Now, P_r is able to compute an "almost valid" signature by decrypting the ciphertext that it received from P_t and performing his part of the signature. This means that P_r computes a tuple $(r', s' := \frac{r' \cdot x'_1 \cdot x'_2 + H(m')}{k'_1 \cdot k'_2})$, and that the complete signature is of the form $(r', s' \cdot \alpha^{-1})$ (lines 27-30 in Figure 13). Since P_r does not have α, he cannot complete the signature. Exactly as in the Schnorr-based construction, P_r receives a randomizable puzzle Z from P_t at the beginning of the puzzle promise protocol, and at the end of the protocol P_r randomizes it to obtain the Z'. The puzzle promise protocol finishes with P_r sending the randomized puzzle Z' to P_s (lines 25-26 and 28 in Figure 13).

The puzzle solver protocol is similar to Schnorr-based puzzle solver protocol, with the sole difference that P_s and P_t compute a two-party ECDSA signature instead of a two-party Schnorr signature.

Public parameters: group description $(\mathbb{G}, g, q)$, message m

$\mathsf{PuzzleSolver}_{P_s}((\mathsf{sk}_s^\Sigma, \mathsf{pk}_{st}^\Sigma), \mathsf{pp}, Z' := (A', c'_\alpha))$		$\mathsf{PuzzleSolver}_{P_t}((\mathsf{sk}_t^\Sigma, \mathsf{pk}_{st}^\Sigma), (\mathsf{pp}, \mathsf{td}))$
		$k_2 \leftarrow_\$ \mathbb{Z}_q; R_2 \leftarrow g^{k_2}$
		$\pi_2 \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists k_2 \mid R_2 = g^{k_2}\}, k_2)$
		$(\mathsf{com}, \mathsf{decom}) \leftarrow \mathsf{P}_{\mathsf{COM}}((R_2, \pi_2))$
	$\xleftarrow{\mathsf{com}}$	
$k_1 \leftarrow_\$ \mathbb{Z}_q; R_1 \leftarrow g^{k_1}$		
$\pi_1 \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists k_1 \mid R_1 = g^{k_1}\}, k_1)$		
$(Z'' := (A'', c''_\alpha), \tau) \leftarrow \mathsf{PRand}(\mathsf{pp}, Z')$		
	$\xrightarrow{Z'' := (A'', c''_\alpha), R_1, \pi_1}$	
		If $\mathsf{V}_{\mathsf{NIZK}}(\pi_1, R_1) \neq 1$ then abort
		$\alpha'' := \mathsf{PSolve}(\mathsf{td}, Z'')$
		If $A'' \neq g^{\alpha''}$ then abort
		$R \leftarrow R_1 \cdot R_2 \cdot A''; e := H(R \| \mathsf{pk}_{st}^\Sigma \| m)$
		$s_2 \leftarrow k_2 - \mathsf{sk}_t^\Sigma \cdot e \bmod q$
	$\xleftarrow{(\mathsf{decom}, R_2, \pi_2), s_2}$	
If $\mathsf{V}_{\mathsf{COM}}(\mathsf{com}, \mathsf{decom}, (R_2, \pi_2)) \neq 1$ then abort		
If $\mathsf{V}_{\mathsf{NIZK}}(\pi_2, R_2) \neq 1$ then abort		
$R \leftarrow R_1 \cdot R_2 \cdot A''; e := H(R \| \mathsf{pk}_{st}^\Sigma \| m)$		
If $g^{s_2} \neq R_2 \cdot (\mathsf{pk}_{st}^\Sigma / g^{\mathsf{sk}_t^\Sigma})^{-e}$ then abort		
$s_1 \leftarrow k_1 - \mathsf{sk}_s^\Sigma \cdot e \bmod q$		
$\bar{s} \leftarrow s_1 + s_2 \bmod q$		
	$\xrightarrow{\bar{s}}$	
		$s \leftarrow \bar{s} + \alpha''$
		If verification of (e, s) fails then abort
		Else publish signature (e, s)
$\alpha'' \leftarrow s - \bar{s}$		
$\alpha' \leftarrow \alpha'' \cdot \tau^{-1}$		
Send α' to P_r		
return α'		**return** $\top$

Fig. 10: Puzzle solver protocol of Schnorr-based construction.

$\mathsf{Open}(\Pi, \alpha')$

Parse Π as $(\beta, (\mathsf{pk}_{tr}^\Sigma, m', \sigma' := (R', s')))$
Set $\alpha \leftarrow \alpha' \cdot \beta^{-1}$
Set $s \leftarrow s' + \alpha$
return (R', s)

$\mathsf{Verify}(\Pi, \sigma)$

Parse Π as $(\beta, (\mathsf{pk}_{tr}^\Sigma, m', \sigma'))$
return $\mathsf{Vf}_{\mathsf{Schnorr}}(\mathsf{pk}_{tr}^\Sigma, m', \sigma)$

Fig. 11: Open and verify algorithms of Schnorr-based construction.

$\mathsf{Open}(\Pi, \alpha')$

Parse Π as $(\beta, (\mathsf{pk}_{tr}^\Sigma, m', \sigma' := (r', s')))$
Set $\alpha \leftarrow \alpha' \cdot \beta^{-1}$
Set $s \leftarrow s' \cdot \alpha^{-1}$
return (r', s)

$\mathsf{Verify}(\Pi, \sigma)$

Parse Π as $(\beta, (\mathsf{pk}_{tr}^\Sigma, m', \sigma'))$
return $\mathsf{Vf}_{\mathsf{ECDSA}}(\mathsf{pk}_{tr}^\Sigma, m', \sigma)$

Fig. 12: Open and verify algorithms of ECDSA-based construction.

Public parameters: group description $(\mathbb{G}, g, q)$, message m'

$\mathsf{PuzzlePromise}_{P_t}((\mathsf{sk}_t^{\Sigma}, \mathsf{pk}_{tr}^{\Sigma}), (\mathsf{pp}, \mathsf{td}), \mathsf{pk}_t^{\tilde{\Sigma}}, \mathsf{pk}_r^{\Psi}, c_{\mathsf{sk}_r^{\Sigma}})$		$\mathsf{PuzzlePromise}_{P_r}((\mathsf{sk}_r^{\Sigma}, \mathsf{pk}_{tr}^{\Sigma}), (\mathsf{sk}_r^{\Psi}, \mathsf{pk}_r^{\Psi}), (\mathsf{tid}, \sigma'_{\mathsf{tid}}))$
	$\xleftarrow{(\mathsf{tid}, \sigma'_{\mathsf{tid}})}$	
If $\mathsf{tid} \in \mathcal{T} \ \vee\ \mathsf{Vf}(\mathsf{pk}_t^{\tilde{\Sigma}}, \mathsf{tid}, \sigma'_{\mathsf{tid}}) \neq 1$ then abort		
Else add tid into $\mathcal{T}$		
$\alpha, k'_2 \leftarrow\$ \mathbb{Z}_q$		
$A \leftarrow g^{\alpha}; R'_2 \leftarrow g^{k'_2}$		
$Z := (A, c_{\alpha}) \leftarrow \mathsf{PGen}(\mathsf{pp}, \alpha)$		
$\pi_{\alpha} \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists \alpha \mid \mathsf{PSolve}(\mathsf{td}, Z) = \alpha\}, \alpha)$		
$\pi'_2 \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists k'_2 \mid R'_2 = g^{k'_2}\}, k'_2)$		
$(\mathsf{com}, \mathsf{decom}) \leftarrow \mathsf{P}_{\mathsf{COM}}((R'_2, \pi'_2))$		
	$\xrightarrow{\mathsf{com}, Z := (A, c_{\alpha}), \pi_{\alpha}}$	
		If $\mathsf{V}_{\mathsf{NIZK}}(\pi_{\alpha}, (c_{\alpha}, A)) \neq 1$ then abort
		$k'_1 \leftarrow \mathbb{Z}_q; R'_1 \leftarrow g^{k'_1}$
		$\pi'_1 \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists k'_1 \mid R'_1 = g^{k'_1}\}, k'_1)$
	$\xleftarrow{R'_1, \pi'_1}$	
If $\mathsf{V}_{\mathsf{NIZK}}(\pi'_1, R'_1) \neq 1$ then abort		
$R'_c \leftarrow (R'_2)^{\alpha}$		
$\pi'_a \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists \alpha \mid A = g^{\alpha} \wedge R_c = (R'_2)^{\alpha}\}, \alpha)$		
$R' \leftarrow (R'_1)^{k'_2 \cdot \alpha}; R' := (r'_x, r'_y); r' \leftarrow r'_x \bmod q$		
$c_1 \leftarrow \mathsf{Enc}(\mathsf{pk}_r^{\Psi}, (k'_2)^{-1} \cdot H(m'))$		
$c_2 \leftarrow (c_{\mathsf{sk}_r^{\Sigma}})^{(k'_2)^{-1} \cdot r' \cdot sk_t^{\Sigma}}$		
$c' \leftarrow c_1 \cdot c_2$		
	$\xrightarrow{(\mathsf{decom}, R'_2, \pi'_2), c', R'_c, \pi'_a}$	
		If $\mathsf{V}_{\mathsf{COM}}(\mathsf{com}, \mathsf{decom}, (R'_2, \pi'_2)) \neq 1$ then abort
		If $\mathsf{V}_{\mathsf{NIZK}}(\pi'_2, R'_2) \neq 1$ then abort
		If $\mathsf{V}_{\mathsf{NIZK}}(\pi'_a, (A, R'_c)) \neq 1$ then abort
		$R' \leftarrow (R'_c)^{k'_1}; R' := (r'_x, r'_y); r' \leftarrow r'_x \bmod q$
		$s'_2 \leftarrow \mathsf{Dec}(\mathsf{sk}_r^{\Psi}, c')$
		If $(R'_2)^{s'_2 \bmod q} \neq (\mathsf{pk}_{tr}^{\Sigma})^{r'} \cdot g^{H(m')}$ then abort
		$s' \leftarrow s'_2 \cdot (k'_1)^{-1} \bmod q$
		$(Z', \beta) \leftarrow \mathsf{PRand}(\mathsf{pp}, Z)$
	$\xleftarrow{s'}$	
		Send $Z' := (A', c'_{\alpha})$ to P_r
If $(R'_1)^{k'_2 \cdot s'} \neq (\mathsf{pk}_{tr}^{\Sigma})^{r'} \cdot g^{H(m')}$ then abort		$\Pi := (\beta, (\mathsf{pk}_{tr}^{\Sigma}, m', \sigma' := (r', s')))$
return $\sigma := (r', s' \cdot \alpha^{-1})$		**return** $(\Pi, (Z, Z'))$

Fig. 13: Puzzle promise protocol of ECDSA-based construction. Blue parts are related to the griefing protection (see Section VI-B)

Public parameters: group description $(\mathbb{G}, g, q)$, message m		
$\mathsf{PuzzleSolver}_{P_s}((\mathsf{sk}_s^\Sigma, \mathsf{sk}_{st}^\Sigma), (\mathsf{sk}_s^\Psi, \mathsf{pk}_s^\Psi), \mathsf{pp}, Z' := (A', c'_\alpha))$		$\mathsf{PuzzleSolver}_{P_t}((\mathsf{sk}_t^\Sigma, \mathsf{sk}_{st}^\Sigma), (\mathsf{pp}, \mathsf{td}), \mathsf{pk}_s^\Psi, c_{\mathsf{sk}_s^\Sigma})$
		$k_2 \leftarrow_\$ \mathbb{Z}_q; R_2 \leftarrow g^{k_2}$
		$\pi_2 \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists k_2 \mid R_2 = g^{k_2}\}, k_2)$
		$(\mathsf{com}, \mathsf{decom}) \leftarrow \mathsf{P}_{\mathsf{COM}}((R_2, \pi_2))$
	$\xleftarrow{\mathsf{com}}$	
$k_1 \leftarrow_\$ \mathbb{Z}_q;\ R_1 \leftarrow g^{k_1}$		
$\pi_1 \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists k_1 \mid R_1 = g^{k_1}\}, k_1)$		
$(Z'' := (A'', c''_\alpha), \tau) \leftarrow \mathsf{PRand}(\mathsf{pp}, Z')$		
	$\xrightarrow{Z'' := (A'', c''_\alpha), R_1, \pi_1}$	
		If $\mathsf{V}_{\mathsf{NIZK}}(\pi_1, R_1) \neq 1$ then abort
		$\alpha'' \leftarrow \mathsf{PSolve}(\mathsf{td}, Z''); R_c \leftarrow (R_2)^{\alpha''}$
		If $A'' \neq g^{\alpha''}$ then abort
		$\pi_{\alpha''} \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists \alpha'' \mid A'' = g^{\alpha''} \wedge$
		$R_c = (R_2)^{\alpha''}\}, \alpha'')$
		$R \leftarrow (R_1)^{k_2 \cdot \alpha''}; R := (r_x, r_y); r \leftarrow r_x \bmod q$
		$c_1 \leftarrow \mathsf{Enc}(\mathsf{pk}_s^\Psi, (k_2)^{-1} \cdot H(m))$
		$c_2 \leftarrow (c_{\mathsf{sk}_s^\Sigma})^{(k_2)^{-1} \cdot r \cdot H(m)}$
		$c \leftarrow c_1 \cdot c_2$
	$\xleftarrow{(\mathsf{decom}, R_2, \pi_2), c, R_c, \pi_{\alpha''}}$	
If $\mathsf{V}_{\mathsf{COM}}(\mathsf{com}, \mathsf{decom}, (R_2, \pi_2)) \neq 1$ then abort		
If $\mathsf{V}_{\mathsf{NIZK}}(\pi_2, R_2) \neq 1$ then abort		
If $\mathsf{V}_{\mathsf{NIZK}}(\pi_{\alpha''}, (A'', R_c)) \neq 1$ then abort		
$R \leftarrow (R_c)^{k_1}; R := (r_x, r_y); r \leftarrow r_x \bmod q$		
$s_2 \leftarrow \mathsf{Dec}(\mathsf{sk}_s^\Psi, c)$		
If $(R_2)^{s_2 \bmod q} \neq (\mathsf{pk}_{st}^\Sigma)^r \cdot g^{H(m)}$ then abort		
$\bar{s} \leftarrow s_2 \cdot (k_1)^{-1} \bmod q$		
	$\xrightarrow{\bar{s}}$	
		$s \leftarrow (\alpha'')^{-1} \cdot \bar{s}$
		If verification of (r, s) fails then abort
		Else publish signature (r, s)
$\alpha'' \leftarrow (s \cdot (\bar{s})^{-1})^{-1}$		
$\alpha' \leftarrow \alpha'' \cdot \tau^{-1}$		
Send α' to P_r		
return α'		**return** $\top$

Fig. 14: Puzzle solver protocol of ECDSA-based construction.