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

Erkan Tairi
TU Wien
erkan.tairi@tuwien.ac.at

Pedro Moreno-Sanchez
TU Wien
pedro.sanchez@tuwien.ac.at

Matteo Maffei
TU Wien
matteo.maffei@tuwien.ac.at

***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 Trilero, the first PCH protocol to achieve all aforementioned properties. Trilero 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 Trilero) in the Universal Composability framework and present a provably secure instantiation based on adaptor signatures. 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 Trilero) 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 Trilero 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 Trilero has resulted in a proof-of-concept implementation in the COMIT Network, a blockchain technology focused on cross-currency payments.

1. Introduction

The increasing adoption of cryptocurrencies has raised scalability issues [1] 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 [1], which contrasts with the throughput of centralized payment networks such as Visa that supports peaks of up to 47,000 transactions per second [2].

Among the several efforts to mitigate these scalability issues [3], [4], [5], payment channels have emerged as the most widely deployed solution in practice. The core idea of payment channels is to let users lock a certain amount of coins (called *collateral*) in a "2-of-2 multisig address"[1] (called *channel*) controlled by them, 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 locked 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.

The problem with this simple construction is that in order to pay different people, a user should establish a payment channel with each of them, which is financially prohibitive, as this user would have to lock an amount of coins proportional to the number of users she wants to transact with. Furthermore, the coins locked at a payment channel with one user cannot be used to perform payments at a different payment channel with another user. For instance, assume that Alice has a payment channel with Bob where she owns 3 coins and another payment channel with Carol where she owns 5 coins. Now assume that the channel between Alice and Carol becomes depleted (i.e., Alice pays all 5 coins to Carol), then Alice cannot pay further to Carol, even though she still owns 3 coins in the channel with Bob. If Alice wants to pay more to Carol, she needs to (i) close the payment channel with Bob to unlock the 3 coins; (ii) create another payment channel with Carol to lock the 3 coins with her. Each of these two steps, however,

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

requires an on-chain transaction, reducing the scalability gain of payment channels in the first place. Payment channel networks (PCNs) 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. PCNs, but raise the issue of finding paths in a network and maintaining the network topology.

Payment channel hubs (PCHs). PCHs constitute a conceptually simpler solution to the aforementioned problem. Each party opens a channel with a central party, called the *tumbler*, which mediates payments between each pair of sender and receiver. By our running example, Alice could open a payment channel with the tumbler where she owns 8 coins, while the tumbler could open two channels, one with Bob and one with Carol. In this setting, now Alice can arbitrarily choose how to pay each of the 8 coins using tumbler as the intermediary. In general, if the sender wants to transfer x coins to the receiver, the sender pays $x + fee$ to the tumbler, which then forwards x coins to the receiver, where *fee* denotes a fee charged by the tumbler. Such a naïve construction, despite being still deployed in many gateways, suffers from obvious *security and privacy issues*: the tumbler could steal coins [6], [7] from honest parties (e.g., by simply not forwarding a payment) and identify who is paying to whom [7], [8].

In the PCH setting, security can be seen in terms of transaction atomicity and should protect the two participants who are sending coins. Atomicity is thus two-fold: (i) the tumbler should receive the money from the sender only if the tumbler has forwarded the corresponding amount to the receiver; (ii) the receiver should receive money from the tumbler only if the sender has paid the corresponding amount to the tumbler. Privacy covers unlinkability, that is, the tumbler should not able to link the sender and receiver of a given payment, an important property for PCHs to support privacy-preserving cryptocurrencies (e.g., Monero, Zcash or Mimblewimble). As these two properties seem contradictory (i.e., how can the tumbler ensure atomicity without knowing who pays to whom?), designing a secure and privacy-preserving PCH is a technical challenge.

Besides security and privacy, another fundamental property is *backwards compatibility*: the tumbler should rely on functionality available at as many cryptocurrencies available today as possible because only then the PCH setting can be deployed in such cryptocurrencies. Additionally, it opens the door to the tumbler being be able to mediate payments across different cryptocurrencies (e.g., the sender transfering bitcoins and the receiver getting ethers), thereby enabling cross-chain applications like exchanges. Achieving the backwards compatibility goal is technically challenging in the presence of cryptocurrencies such as Ripple or Mimblewimble where payments governed by a highly restricted functionality, namely digital signatures and time locks.

State-of-the-art in PCH. BOLT [9] is an off-chain protocol for PCHs that provides strong anonymity guarantees by relying a novel protocol based on zero-knowledge proofs. This approach, however, requires functionality that is only provided by Zcash.

Perun [10] builds the PCH upon virtual channels, an Ethereum smart contract-based construction that intuitively allows to fold two channels (e.g., Alice $\rightarrow$ Tumbler $\rightarrow$ Bob) into a single channel (Alice $\rightarrow$ Bob). This technique, however, inherently leaks the sender-receiver relation between Alice and Bob to the tumbler.

NOCUST [11] is an off-chain protocol that relies on an untrusted operator to manage the off-chain payments among parties. The operator periodically includes a summary on-chain that certifies the balances and the transactions for public verifiability. Similar to Perun, NOCUST does not provide unlinkability against a malicious intermediate (i.e., operator) and also lacks backwards compatibility.

TeeChain [12] is an off-chain protocol that leverages trusted execution environments (e.g., Intel SGX) to manage off-chain payments and resolve disputes among channel participants. Thus, parties need to run a TEE, which hinders the widespread deployment of this approach in practice.

TumbleBit [13] is a cryptographic protocol for PCHs that makes transactions unlinkable (i.e., the tumbler does not learn who is paying to whom). However, the cryptographic construction used in TumbleBit builds upon the cut-and-choose technique, and thus TumbleBit requires computation and communication costs that grow linearly in the security parameter. More precisely, TumbleBit's security level is computed via the binomial coefficient $\binom{m+n}{m}$, where m and n are the parameters of the cut-and-choose game. This implies that the parties need to compute and exchanges messages composed of $m+n$ elements, hence, linear in the parameters of the security game. For instance, TumbleBit requires messages of size $250-400$KB for a single payment, and running times of up to 10 seconds in the worst case (over a WAN setting with parties connected through Tor network). Moreover, TumbleBit relies on the hash-time lock contract (HTLC), a Bitcoin script-based construction that allows for payments conditioned on obtaining the preimage of a hash function. However, this limits the deployment of TumbleBit to the cryptocurrencies supporting HTLC, ruling out cryptocurrencies without support for scripting language, such as Ripple, Stellar or Mimblewimble.

We summarize the properties achieved by each PCH construction in Table 1. Most notably, all state-of-the-art PCHs fail to achieve at least one of the considered properties and, in particular, all of them fall short of backwards compatibility: while BOLT, Perun and NOCUST totally lack it, Teechain achieves it at the cost of adding a new trust assumption (TEE), whereas TumbleBit is restricted to blockchains supporting HTLC contracts, and suffers from a

TABLE 1: Comparison among state-of-the-art PCH.

	Atomicity	Unlinkability	Compatibility (Required functionality)
BOLT [9]	●	●	○ (Zcash)
Perun [10]	●	○	○ (Ethereum)
NOCUST [11]	●	○	○ (Ethereum)
Teechain [12]	●	●	◐[2] (Trusted Execution Environment)
TumbleBit [13]	●	●[1]	◐[3] (HTLC-based currencies)
A^{2}L	●	●[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).

high communication complexity.

This state of affairs in the state-of-the-art leads us to consider the following questions:

Q1: What is the minimal functionality from the scripting language required to design a PCH?

Q2: Is it possible to have a PCH backwards compatible with virtually all cryptocurrencies that is efficient and provides security and privacy?

Our contributions. This work answers Q1 by presenting the first PCH construction that requires only digital signatures and timelock functionality from the underlying cryptocurrency. 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). Additionally, we also answer Q2 in the affirmative as our protocol is the first secure, privacy-preserving, and backwards compatible PCH, whose communication complexity is constant (i.e., the number of elements that need to be exchanged are independent of the security parameter). Specifically,

- We introduce Trilero, a cryptographic PCH realization whose core technical ingredient is a novel cryptographic primitive called anonymous atomic locks (A^2L). Intuitively, A^2L realizes 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 model Trilero as well as its security and privacy properties (namely, atomicity and unlinkability) in the Universal Composability (UC) framework [14], providing the first formalization of the PCH problem in UC.
- We give an instantiation based on adaptor signatures, which in turn can be securely instantiated by well-known signature schemes such as Schnorr and ECDSA [15]. By dispensing from HTLCs, our instantiations offer the highest degree of backwards compatibility among the state-of-the-art PCHs: e.g., Ripple and Stellar support ECDSA and Schnorr but not HTLCs, whereas Mimblewimble supports Schnorr but not HTLCs.
- Our evaluation of A^2L shows that our construction requires a running time of ~0.6 seconds. Furthermore, the communication cost is less than 10KB . When compared to TumbleBit, the most backwards compatible PCH prior to this work, which has linear communication and computation complexity, 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 that TumbleBit do not implement, such as protection against griefing attacks. These results demonstrate that Trilero is the most efficient Bitcoin-compatible PCH. Additionally, Trilero has been implemented as a proof-of-concept in the COMIT Network (see Section 5), a technology focussed on payments across different cryptocurrencies.

2. Problem Description and Solution Overview

Notation. A payment-channel hub (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, $\varsigma.\mathsf{cash}\colon \varsigma.\mathsf{users} \to \mathbb{R}^{\geq 0}$ is a mapping from channel users to their respective amount of coins in the channel, and $\varsigma.\mathsf{state} = (\theta_1, \ldots, \theta_n)$ is the current state of the channel, which is composed of a list of *updates*.

Since our cryptographic construction is orthogonal to how payment channels are actually updated, we abstract away from the details to update a payment channel and refer the reader to [4], [15] for a detailed description. Instead, we hereby assume that payment channels support the two functionalities that we require for our PCH construction. First, we assume that the current channel state $\varsigma.\mathsf{state}$ requires a valid 2-of-2 multisignature from $\varsigma.\mathsf{users}$ to be added to the blockchain. We remark that this is only for presentation purposes and we detail in Appendix F how to modify our construction to be backwards compatible with cryptocurrencies missing support for multisignatures and only supporting standard digital signatures. Second, we hereby denote by $\mathsf{TLP}(\theta, t)$ a timelocked payment where the channel update θ (within $\varsigma.\mathsf{state}$) can be accepted before time t expires if $\varsigma.\mathsf{users}$ provide the corresponding valid 2-of-2 multisignature. After t expires, the update θ can only be rejected.

PCH functionality. A PCH allows for off-chain payments between two parties connected by an intermediary, called tumbler and denoted here as P_t. A payment from the sender P_s to the receiver P_r for amt coins through the channels ς and ς', such that $\varsigma.\mathsf{users} = \{P_s, P_t\}$ and $\varsigma'.\mathsf{users} = \{P_t, P_r\}$, requires that $\varsigma.\mathsf{cash}(P_s) \geq \mathsf{amt}$ and $\varsigma'.\mathsf{cash}(P_t) \geq \mathsf{amt}$. If these prerequisites are met, a payment updates the balances at both channels as follows: $\varsigma.\mathsf{cash}(P_s)$ $-=$ amt, $\varsigma.\mathsf{cash}(P_t)$ $+=$ amt, $\varsigma'.\mathsf{cash}(P_t)$ $-=$ amt and $\varsigma'.\mathsf{cash}(P_r)$ $+=$ amt. In other words, amt is moved from the sender to the tumbler in the channel ς and from the tumbler to the receiver in the channel ς'. 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 Section 5.1.2.

Challenges in PCH. Naively, P_s could pay P_t, and later on let P_t forward the payment to P_r. However, this falls short of both unlinkability and atomicity. For unlinkability, even in the presence of several simultaneous payments, even an honest-but-curious P_t learns that the two channels that are immediately updated are part of the same payment, i.e., who pays to whom. Hence different payments must be intertwined in a non-predictable way. For atomicity, if P_s

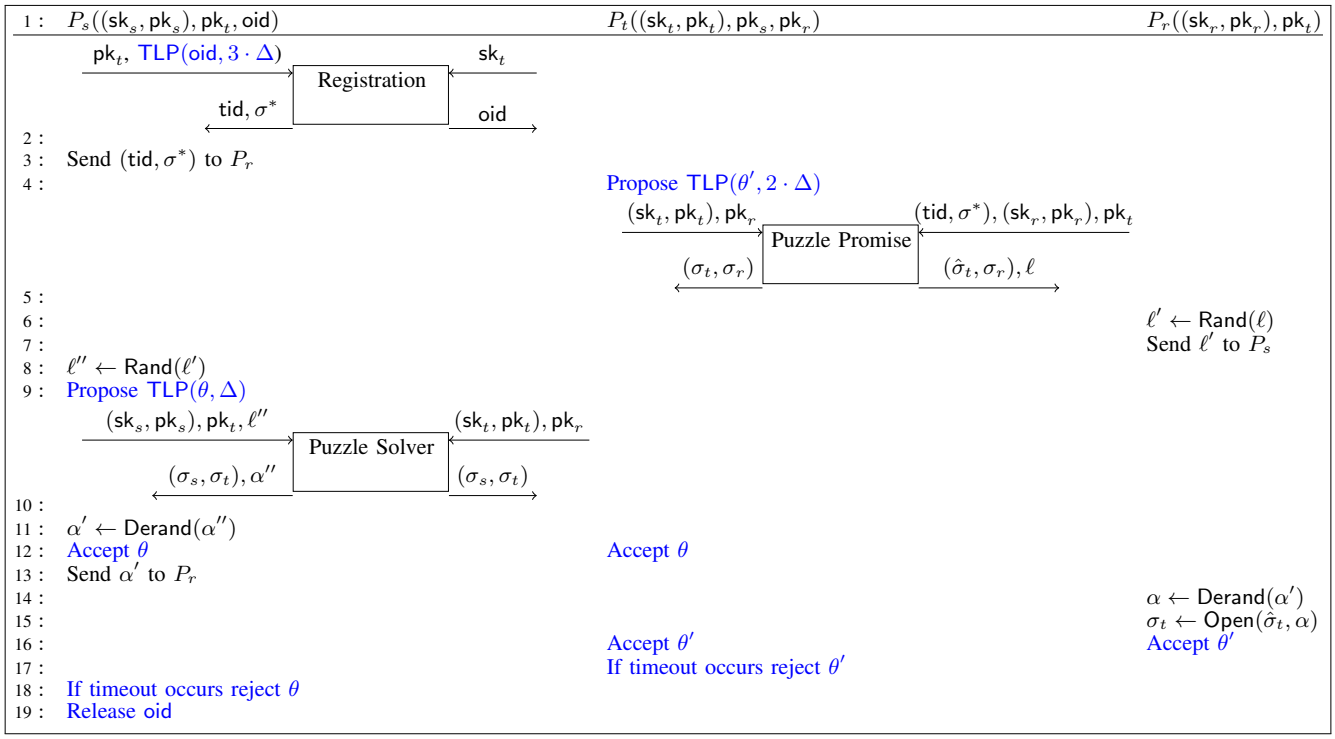

Figure 1: Overview of Trilero. Here θ and θ' denote the updates $(\varsigma.\mathsf{cash}(P_s) \mathrel{-}= \mathsf{amt}, \varsigma.\mathsf{cash}(P_t) \mathrel{+}= \mathsf{amt})$ and $(\varsigma'.\mathsf{cash}(P_t) \mathrel{-}= \mathsf{amt}, \varsigma'.\mathsf{cash}(P_r) \mathrel{+}= \mathsf{amt})$ on channels ς and ς', respectively. Here, $\mathsf{TLP}(\theta, \Delta)$ denotes a timelocked payment of the state denoted by θ with timeout Δ (i.e., the duration of an epoch). Black pseudocode are operations in A^{2}L.

pays P_t first, then a malicious P_t could get the money from P_s without paying P_r. Even if P_t pays first, a malicious P_s could abstain from paying P_t, which would thus incur in a loss. Hence, the update in the channel between P_t and P_r should be *conditioned* to the update in the channel between P_s and P_t. Achieving unlinkability and atomicity simultaneously is challenging, since one has to condition updates at two different payment channels without establishing any observable link between the two of them in the first place.

2.1. Our Approach

Our approach to achieve unlinkability and atomicity in PCH consists of two building blocks, as depicted in Figure 1. First, we design anonymous atomic locks (A^{2}L), a three-party cryptographic construction that allows the synchronization of the updates on two payment channels atomically while preserving unlinkability. Second, we introduce Trilero, a PCH protocol that performs the actual update on the channels by leveraging A^{2}L. In the following, we overview both building blocks.

2.1.1. Anonymous atomic locks (A^{2}L). Inspired from TumbleBit [13], we design A^{2}L in two phases: *puzzle promise* and *puzzle solver* (we omit *registration* for a moment). During these two phases, the update on the channel ς', between P_t and P_r, defined as $\theta' := (\varsigma'.\mathsf{cash}(P_t) \mathrel{-}= \mathsf{amt}, \varsigma'.\mathsf{cash}(P_r) \mathrel{+}= \mathsf{amt})$ (i.e., the tumbler P_t paying amt coins to the receiver P_r) is established first but its success is conditioned on the successful update of the channel ς, between P_s and P_t, defined as $\theta := (\varsigma.\mathsf{cash}(P_s) \mathrel{-}= \mathsf{amt}, \varsigma.\mathsf{cash}(P_t) \mathrel{+}= \mathsf{amt})$ (i.e., the sender P_s paying amt 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.

In a bit more detail, during the puzzle promise phase P_t gives P_r a cryptographic puzzle ℓ and an "almost valid" authorization $(\hat{\sigma}_t, \sigma_r)$ for update θ' on channel ς'. Here, "almost valid" means that P_r can covert $\hat{\sigma}_t$ in a valid signature σ_t only if P_r can provide a solution to ℓ. Note that at the beginning, only P_t knows this solution and P_r does not have an efficient way to obtain it.

In such a setting, P_s helps P_r to get the solution to the aforementioned challenge. In particular, P_s initiates the update θ on channel ς conditioned on the fact that P_t reveals α'', a solution to a randomized version of the cryptographic puzzle ℓ'', which is in turn a randomized solution of the original puzzle ℓ. Note here that it is important that P_s inputs ℓ'', that is a randomized version of the puzzle ℓ, as otherwise the tumbler P_t could trivially link sender and receiver by inspecting ℓ itself. If P_t accepts the update θ (i.e., P_t receives amt coins from P_s by computing a valid signature (σ_s, σ_t)), then P_s can obtain the puzzle solution and provide it to P_r who then accepts the pending

update in θ'. Otherwise, both channel updates are expected to be void after a certain predefined time expires. This mechanism thereby provides atomicity. Moreover, if there are several payments in parallel, the randomization of the cryptographic puzzles intuitively prevents P_t from linking the sender/receiver pairs.

This payment paradigm opens two challenges: (i) how to instantiate the required cryptographic puzzle so that it embeds the logic for "converting an almost valid signature into a fully valid signature reveals the solution to the cryptographic puzzle" while not only atomicity and unlinkability are preserved but also backwards compatibility and efficiency are obtained; and (ii) how to mitigate *griefing attack*, a DoS attack against the liquidity of the tumbler if many (possibly Sybil) receivers ask the tumbler for payment promises (thus forcing the tumbler to lock coins in a time-lock payment) that are never fulfilled later by the corresponding sender. In the following, we overview how we tackle both challenges.

Cryptographic puzzle in A^{2}L. Different to TumbleBit (and any other PCH in the literature), our instantiation of the cryptographic puzzle required in A^{2}L is a novel cryptographic protocol built upon the notion of *adaptor signatures* [15]. We give an intuitive description of adaptor signatures here and refer the reader to Section 3.3.1 for a more formal definition. An adaptor signature allows to create a partial signature $\hat{\sigma}$ on a message m with respect to a secret value α that is "almost valid" (i.e., *pre-sign* a message w.r.t. some statement A of a hard relation, where α is the corresponding witness of A). An adaptor signature requires the following two conditions to hold: (i) a party P can convert $\hat{\sigma}$ into a full and valid signature σ on its own only if it gets to know the secret (witness) α; (ii) a party P knowing the partial signature (pre-signature) $\hat{\sigma}$ and the corresponding valid signature σ can extract the secret value α. These two conditions are at the core of adaptor signatures and our construction as we overview here.

Strawman approach: We could construct A^{2}L naïvely leveraging adaptor signatures as follows: During the puzzle promise phase the tumbler P_t gives a pre-signature $\hat{\sigma}_t$ to the receiver P_r with respect to a statement A for which only P_t knows the corresponding witness α. This ensures that P_r can turn $\hat{\sigma}_t$ into a full and valid signature σ_t once P_r learns the secret α. P_r could then compute σ_r on its own, update the channel and effectively get paid from P_t. However, P_r does not have access to α yet.

Instead, P_r can share the statement A with the sender P_s, who can use it during the puzzle solver phase to give a pre-signature $\hat{\sigma}_s$ to P_t. At this point P_t can transform $\hat{\sigma}_s$ into a valid signature σ_s as it knows the witness α (this corresponds to condition (i) of adaptor signatures as mentioned above), and use it to update the channel and get paid from P_s. Next, P_s can extract α from $\hat{\sigma}_s$ and the published σ_s (this corresponds to condition (ii) of adaptor signatures as mentioned above), and share α with P_r. Finally, P_r can transform $\hat{\sigma}_t$, which it received from P_t during the puzzle promise, into a valid signature σ_t and get paid from P_t.

Issues with strawman approach: Although, this naïve approach provides the expected functionality and ensures atomicity (i.e., P_r only gets paid once P_t is paid by P_s), it completely breaks the sender/receiver unlinkability as the same statement A is used in both puzzle promise and puzzle solver phases, hence, P_t can trivially link that P_s pays to P_r. An easy fix would be to randomize the statement before starting the puzzle solver phase. In particular, we could use an instance of the discrete logarithm problem (i.e., $\{\exists\alpha \mid A = g^\alpha\}$), which is trivially randomizable and can be done so by either P_r or P_s before the start of the puzzle solver phase. Hence, during the puzzle promise phase the statement A is used, while in the puzzle solver phase the randomized statement A' is used. Although this approach maintain unlinkability (i.e., P_t cannot trivially link P_s and P_r), now P_t cannot complete the pre-signature that it receives from P_s, as the statement is A', and P_t is missing the corresponding witness.

This points us to the key challenge in A^{2}L which is that it must provide the following two properties: (i) it should ensure atomicity (P_r gets paid only if P_t is paid by P_s, which implies that P_t must know the solution for the puzzle and can thus complete the signature during puzzle solver); (ii) it should ensure unlinkability (P_t cannot link that P_s is paying to P_r, e.g., because exactly the same statement and solution is used for both puzzle promise and puzzle solver).

Add the encrypted solution to the puzzle: The key of our solution to this dichotomy is the use of an additively homomorphic encryption scheme as follows. During the puzzle promise phase, P_t sends the statement A as before and additionally sends an encryption of the witness α under its own public key (ciperhtext c_α), which makes it decryptable only by P_t. Then, P_r can randomize both the statement A and the ciphertext c_α with the same randomization factor before giving them to P_s, who starts the puzzle solver phase with P_t. At this point P_t can decrypt to obtain the randomized witness α' for the randomized statement A' and complete the pre-signature $\hat{\sigma}_s$ to get paid by P_s. Then, P_s can extract α', which it shares with P_r, who can derandomize to obtain α and complete $\hat{\sigma}_t$. Intuitively, this approach preserves both atomicity and unlinkability of A^{2}L.

Putting everything together: Next, we describe how to use A^{2}L to build Trilero. First, P_t samples a secret value α, and sets the puzzle (lock) ℓ to be an "encoded form" of α. In our case encoding corresponds to an encryption of α (denoted as c_α) along with DLOG $A = g^\alpha$ (i.e., $\ell := (A, c_\alpha)$), which makes it hard to obtain α from the encoded form, but allows for randomization. Next, P_t uses an adaptor signature scheme along with A to create a partial signature $\hat{\sigma}_t$ on the update θ' on channel ς', and provides P_r with $\hat{\sigma}_t$ and ℓ (this corresponds to the Puzzle Promise subprotocol in Figure 1). This is sufficient for P_r to check that it can finalize the signature (and thus the channel update) as soon as it gets to know α. For that, P_r randomizes ℓ and sends this randomized version $\ell' := (A', c'_\alpha)$ to P_s (lines 5-6 in Figure 1). In the puzzle solver phase, P_s also randomizes the lock ℓ' to obtain $\ell'' := (A'', c''_\alpha)$ and uses it to create a new adaptor signature $\hat{\sigma}_s$ on the update θ on channel ς such that the corresponding embedded secret value is α'', which

is a doubly-randomized version of α. Then, P_s gives $\hat{\sigma}_s$ and ℓ'' to P_t, who decrypts the ciphertext c''_α and extracts α''. Next, P_t computes a valid signature σ_s on the update θ on channel ς (this corresponds to the Puzzle Solver subprotocol in Figure 1). Once the valid signature σ_s is published, P_s extracts the value α'' from the pair $(\sigma_s, \hat{\sigma}_s)$, derandomizes it to obtain α', which corresponds to the decoded form of ℓ' (i.e., the discrete logarithm of A'), and sends it to P_r (lines 11-13 in Figure 1). Finally, P_r derandomizes α' to obtain α, which is the decoded form of the initial puzzle ℓ given to it by P_t, and computes a valid signature σ_t from the pre-signature $\hat{\sigma}_t$ using α (lines 14-15 in Figure 1).

Handling griefing attacks in A^{2}L. As mentioned earlier, the payment paradigm considered in this work (and also in TumbleBit [13]) opens the door for griefing attacks where the receiver P_r starts many puzzle promise operations, each of which requires that tumbler P_t locks amt coins, whereas the corresponding puzzle solver interactions are never carried out. Previous proposals to handle this DoS attack [13] 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 our focus here. Moreover, the transaction fee that P_r pays is smaller than amt and thus this mitigation does not fully address the griefing attack issue.

Our approach: We observe thus 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 amt 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 amt 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 Section 5.1).

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 collateral of amt coins locked for this interaction without revealing which P_s did lock the coins; and (ii) P_t should be able to check that the same collateral is not claimed twice.

Our DoS prevention approach is implemented in the *registration* phase. Here, P_s updates its channel with P_t to perform a TLP payment (a timelock payment) of amt coins to P_t, but P_s does not authorize it (i.e., P_s does not sign it). This effectively means that these amt coins are locked there until a certain timeout expires and only then P_s is able to reuse the coins. In Figure 1, we denote this output by oid. While doing that, P_t creates a fresh token tid and signs it using (blinded) randomizable signatures scheme. Intuitively, P_t can create a signature σ^* over a fresh random token tid without learning tid in the first place. P_r can later show tid to P_t as a proof that there exists a P_s who locked coins beforehand, but yet not revealing the exact P_s. We note that each token can only be used once as P_t keeps a list of all the revealed tokens, hence, this avoids replay attacks. We refer the reader to Section 3 for a more detailed description.

2.1.2. Trilero. The A^{2}L primitive is agnostic of the actual content of the channel updates (e.g., amount of coins) and it does not provide any timelock mechanism. Trilero realizes a PCH by augmenting A^{2}L to provide the two aforementioned functionalities, which are crucial for a PCH. First, Trilero ensures that channel updates θ and θ' reflect a payment from P_s to P_r through P_t for a fixed amount amt of coins, thereby ensuring that parties have an economic incentive to execute A^{2}L. Second, Trilero implements the timelock mechanism that we denoted by TLP to void a certain channel update if a payment operation fails (e.g., a party does not answer before a certain timeout), which ensures that the honest users do not lose coins or that coins do not get infinitely locked.

3. 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 [13], we assume that the protocols are run in phases and epochs. Each epoch is composed of four phases for us: (i) registration phase (i) puzzle promise phase (escrow phase in TumbleBit), (iii) puzzle solver phase (payment phase in TumbleBit), and (iv) open phase (cash-out phase in TumbleBit). In each epoch instances of our protocols are executed in their corresponding phases (e.g., registration protocol is executed during the registration 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 [13]. We stress that we only need this anonymous communication between the sender and receiver when exchanging the puzzle and its solution.

3.1. Security and Privacy Goals

Authenticity. The PCH should only start a payment procedure if a fresh and authentic token, previously issued by the tumbler P_t, is presented. This ensures that all payment requests are already backed up by some locked coins, and in turn avoids the griefing attacks described in Section 2.1.

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 [13].

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 argue unlinkability in terms of *interaction multi-graph* as defined in [13]. 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.

3.2. Trilero: Our PCH Instantiation

As mentioned in Section 2, Trilero realizes a PCH by appropriately setting the channel updates and timelock mechanism for the otherwise payment agnostic A^{2}L. In particular, Trilero carries out this task as shown in Figure 1 (blue pseudocode) and described in the following:

1) **Collateral setup**: In Trilero, before the registration phase of A^{2}L starts, P_s updates its channel with P_t to propose a timelock payment (TLP), represented by oid, that locks amt coins from the balance of P_s into oid. This oid update 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 to perform subsequent phases of the PCH protocol.

2) **Payment channel update proposals**: In Trilero, before the puzzle promise phase of A^{2}L starts, P_t updates its channel with P_r to propose a timelock payment, represented by θ', 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 time-lock 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.

3) **Payment channel update resolutions**: In Trilero, 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, Trilero first updates the channel between P_s and P_t to integrate the payment represented in θ. Afterwards, P_r can finalize the authorization of the update θ' and Trilero accordingly reflects this payment in the channel between P_t and P_r. On the other hand, if any of puzzle promise or puzzle solver fails, then Trilero rejects both payment proposals θ and θ', leaving balances at both channels as before the start of the execution of the payment.

4) **Collateral release**: Independently of the outcome of the previous phases, Trilero releases the coins locked by P_s at the beginning of the payment as collateral.

3.3. Anonymous Atomic Locks (A^{2}L)

3.3.1. Cryptographic Building Blocks. 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., $\mathsf{negl}(\lambda)$). 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 $\mathcal{F}_{\mathsf{COM}}$ [14], as described in Appendix D. In our protocols we use the Pedersen commitment scheme [16], 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 A. A non-interactive zero-knowledge proof scheme NIZK [17] consists of PPT algorithms $\mathsf{NIZK} = (\mathsf{P_{NIZK}}, \mathsf{V_{NIZK}})$, where $\mathsf{P_{NIZK}}$ is the prover algorithm, such that $\pi \leftarrow \mathsf{P_{NIZK}}(x,w)$, and $\mathsf{V_{NIZK}}$ is the verification algorithm, such that $\{0,1\} := \mathsf{V_{NIZK}}(x,\pi)$. 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 D.

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 $\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) =$

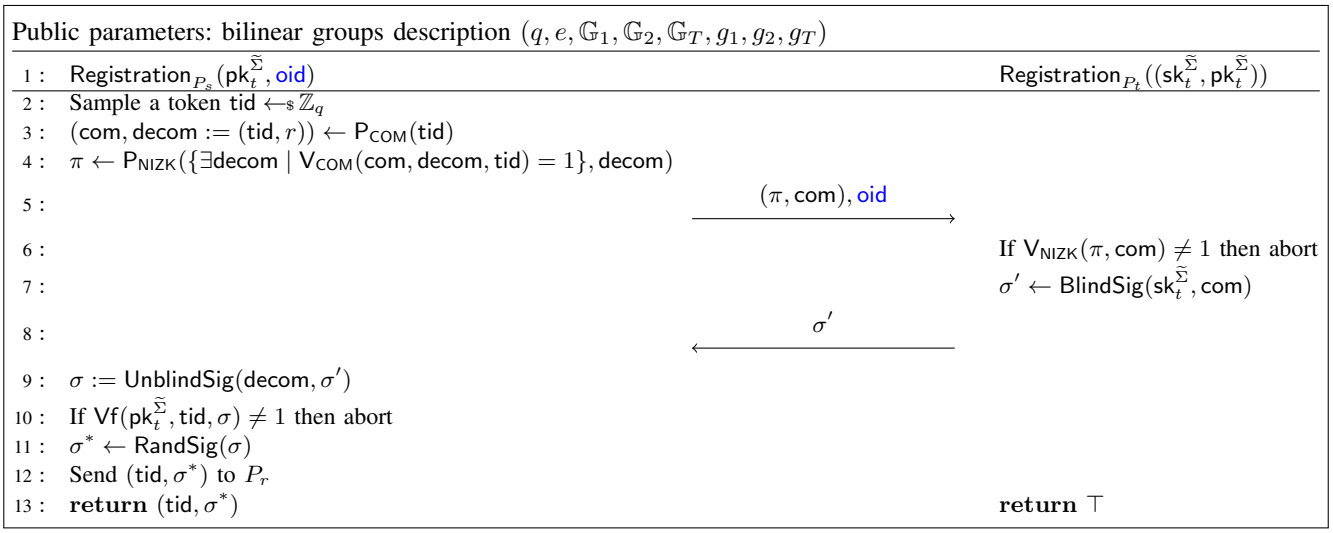

Figure 2: Registration protocol of A^{2}L. Blue part is related to the payment (i.e., non-cryptographic operation).

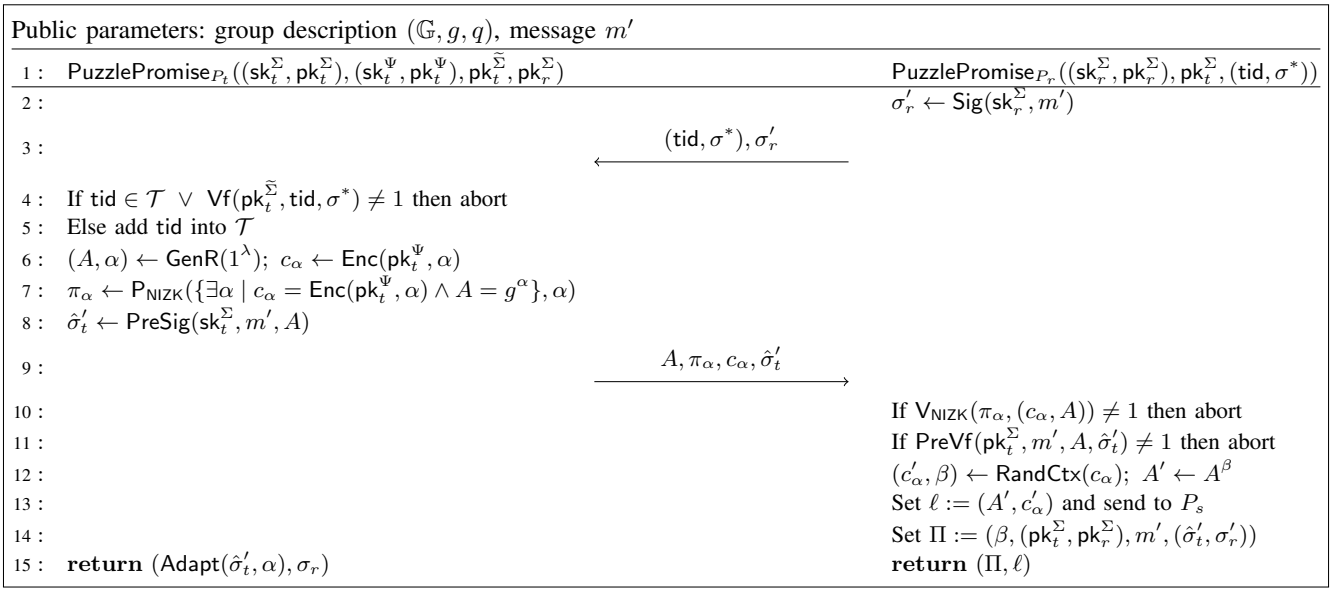

Figure 3: Puzzle promise protocol of A^{2}L.

$m \mid (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(1^\lambda)] = 1$, 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) [18] (more precisely, HSM-CL described in [19]), 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 B. Additionally, we assume existence of a function 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

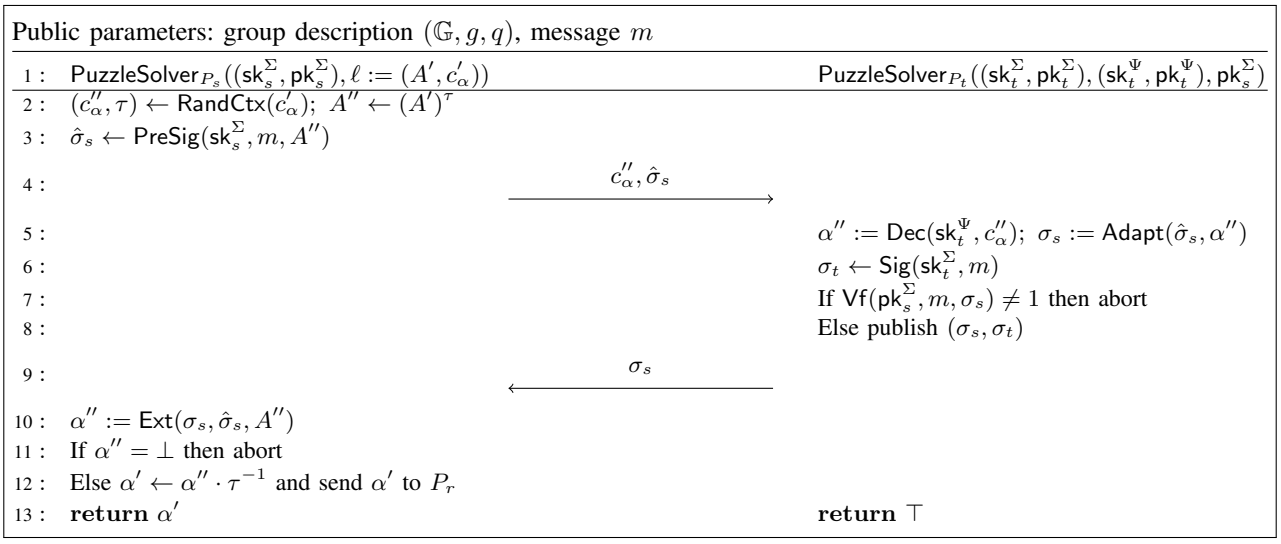

Figure 4: Puzzle solver protocol of A^{2}L.

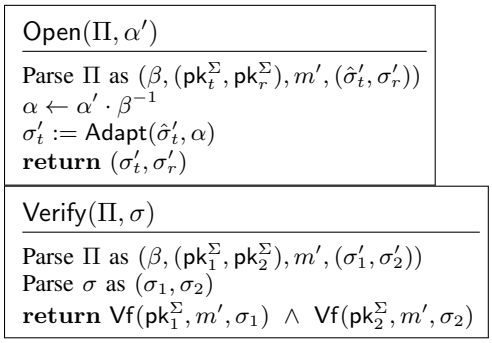

Figure 5: Open and verify algorithms of A^{2}L.

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 [15], 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 A for a more formal and detailed treatment of adaptor signatures. Lastly, we point out that the authors of [15] 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) [20] signature scheme, which

works over Type-III bilinear groups.

3.3.2. A^{2}L Construction. Our A^{2}L construction is composed of three protocols: registration, puzzle promise and puzzle solver. We detail each of them in the following.

Registration. The registration protocol, which can be seen in Figure 2, is used to defend against the griefing attacks as explained in Section 2.1. 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 output oid before the start of the protocol.

Our protocol is inspired from anonymous credentials [21], 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 (in order to protect against the grifieng attacks explained in Section 2.1). 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 as it happens off-chain.

Our registration protocol makes use of a (blinded) randomizable signature scheme $\widetilde{\Sigma}$ as described in Section 3.3.1, which we instantiate with Pointcheval-Sanders (PS) [20], a commitment scheme, for which we use Pedersen commitment [16], 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 2). 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 2). Here, it is important that tid is hidden (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 3) 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 [20] signature scheme).

Next, P_s unblinds σ' using the decommitment information decom to obtain a valid signature σ on the token tid (line 9 in Figure 2). Lastly, P_s randomizes σ to obtain σ^* and sends the pair (tid, σ^*) to the receiver P_r (lines 11-12 in Figure 2), 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' and σ are the same. Therefore, if P_r gives σ to P_t at the beginning of the puzzle promise protocol for validation, then P_t can easily link P_s and P_t. This is the reason why we randomize σ and only share the randomized signature σ^* 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 2 it is randomized by P_s as part of the registration protocol).

Puzzle promise. Once the registration protocol is completed the puzzle promise protocol shown in Figure 3 starts. 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 as described in Section 3.3.1. The puzzle promise protocol starts with P_r sending the pair (tid, σ^*) to P_t along with its own valid signature σ'_r on a previously agreed message m', which is the agreed transaction (lines 2-3 in Figure 3). P_t makes sure that the signatures are valid and 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 previously seen token identifiers. We note that since we expect our protocols to run in epochs we can reduce the storage requirement of P_t by letting it generate a new key pair, publish $\mathsf{pk}_t^{\widetilde{\Sigma}}$ so that it is available to others, and then reset the list $\mathcal{T}$. 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.

Next in the protocol, P_t samples a statement/witness pair (A, α), for a statement $A := g^\alpha$, creates the cryptographic challenge (puzzle) $\ell := (A := g^\alpha, c_\alpha)$, where c_α is an encryption of α using a homomorphic encryption scheme Ψ under P_t's public key pk_t^Ψ, along with a NIZK proof π_α proving that c_α encrypts the witness of A (lines 6-7 in Figure 3). Furthermore, 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 ℓ along with the adaptor signature $\hat{\sigma}'_t$ with P_r (lines 8-9 in Figure 3). 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 3).

Once P_r is convinced of the validity of $\hat{\sigma}'_t$, it randomizes the puzzle ℓ using the operation of the underlying group and the homomorphic properties of Ψ to obtain the puzzle ℓ', which it shares with P_s (lines 12-13 in Figure 3). This finalizes the puzzle promise protocol, and allows P_s to start the puzzle solver protocol with P_t, as shown in Figure 4.

Puzzle solver. In the puzzle solver protocol, P_s further randomizes ℓ' into ℓ'' (line 2 in Figure 4) to preserve its own anonymity and thwart attacks involving collusion of P_t and P_r (see Section 5.1.1). Then, P_s gives an adaptor signature $\hat{\sigma}_s$ to P_t (lines 3-4 in Figure 4), which is adapted with the newly randomized puzzle, along with the ciphertext element of the puzzle, which in this case is c''_α . The ciphertext c''_α of the puzzle is crucial as it is encrypted under the public key of P_t, and hence, P_t can decrypt it to obtain α'', the doubly randomized version of the value α (i.e., the secret value required by P_r to complete the adaptor signature $\hat{\sigma}'_t$ obtained from the puzzle promise protocol). 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 4).

All that is remaining 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 4). 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, and then get rid of one layer of the randomization to obtain α', which it shares with P_r (lines 10-12 in Figure 4). 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 5.

3.3.3. Discussion. Our protocol achieves backwards compatibility with virtually all current cryptocurrencies. Backwards compatibility 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 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 signatures, such as Ripple, Stellar or Mimblewimble following the threshold version of adaptor signatures [22]. We describe in Appendix F 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 [22], 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 [23]. We discuss further deployment aspects for cross-chain payments in Section 5.1.

4. Security Analysis

We formalize the security and privacy of A^{2}L and Trilero in the universal composability (UC) framework [14]. We rely on the synchronous version of global UC framework (GUC) [24]. 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 D, we formally prove the following theorem.

Theorem 1. *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* Ψ *be an* IND-CPA *secure encryption 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.*

Furthermore, we define an ideal functionality $\mathcal{F}_{\mathsf{PCH}}$, which defines the ideal behavior of our PCH protocol Trilero. 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 E.2.

Theorem 2. *The protocol described in Figure 8, UC-realizes* $\mathcal{F}_{\mathsf{PCH}}$ *in the* $(\mathcal{F}_{\mathsf{A^2L}}, \mathcal{F}_{\mathsf{GDC}}, \mathcal{F}_{\mathsf{smt}})$*-hybrid model.*

4.1. 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 2), which then during the puzzle promise protocol (see Figure 3) 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 user 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 A) hardness of the relation R (which is implied by the security of the adaptor signature scheme), and the security of the homomorphic encryption scheme Ψ.

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. Since P_t only shares with P_r a pre-signature

$\hat{\sigma}'_t$, the statement A corresponding to $\hat{\sigma}'_t$, and a ciphertext c_α which encrypts the witness of A, 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) find the witness for a hard relation; (iii) obtain the plaintext encrypted under the public key of P_t. 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 the malicious party can obtain the witness α for A and adapt the pre-signature $\hat{\sigma}'_t$ into a valid signature σ'_t. However, this implies that we can break the discrete logarithm (DLOG) problem (as the underlying hard relation in our case is DLOG), which is believed to be a hard problem to solve. Both of these are related to the security of the underlying adaptor signature scheme. Lastly, if the third approach succeeds, then the malicious party can obtain the witness from the ciphertext c_α and adapt the pre-signature. Though, then we can 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) secret value that P_r needs to get paid or it reveals an invalid secret value 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 A 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 3.1) 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 common denomination (of amount amt as described in Section 3), P_t cannot correlate the transaction values to learn any non-trivial information. Next, in Section 3 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 information-theoretically unlinkable. This is due to the fact that the randomized puzzles ℓ' and ℓ'' are equally likely to be randomizations of any of the puzzles ℓ produced by P_t during the puzzle promise phase. Lastly, in Section 3 we assumed that P_s and P_r communicate through a secure and anonymous communication channel, hence, P_t cannot eavesdrop and use the network information to link P_s and P_r.

5. Performance Analysis

Implementation details. The implementation is written in C, and it relies on the RELIC library [25] for the cryptographic operations (with GMP [26]) as the underlying arithmetic library), and on the PARI library [27] 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 [18], [19] for 128-bit security level as described in [19, Section 4]. We instantiated the (blinded) randomizable signature scheme $\widetilde{\Sigma}$ with Pointcheval-Sanders (PS) [20] signature scheme over the pairing-friendly curve *BN P-256*. Note that it is not necessary that this curve is the same as the underlying cryptocurrency (e.g., Bitcoin) 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 [28] and made non-interactive using the Fiat-Shamir heuristic [29]. Lastly, we have instantiated the commitment scheme COM for the registration protocol (see Figure 2) using the Pedersen commitment scheme [16]. 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/etairi/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 2.

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 [15]. Next, we compare our constructions with the state-of-the-art payment hub TumbleBit [13]. 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-precessing 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 [15]. 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 [13] do not include any protection against the griefing attack explained in Section 2.1, 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.

TABLE 2: Performance of Schnorr- and ECDSA-based constructions. 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	Schnorr	0.008	0.183	0.118	0.307
(preprocessing)	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)

Third, we point that the main bottleneck with respect to computation and communication in our constructions is CL encryption [18] and CLDL zero-knowledge argument of knowledge (AoK) [30] (denoted as π_α in our construction). 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 generate many random α values, along with their corresponding ciphertext c_α and proof π_α during its idle time, so that during the actual protocol run these values do not need to be computed. 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 2.

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: https://github.com/comit-network/a2l-poc.

5.1. System Discussion

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

5.1.1. Limitations of Unlinkability. In this section, we discuss the unlinkability limitations inherent to the PCH setting, and thus also affecting Trilero. We remark that these limitations are inherent to any tumbler protocol, as shown for instance in TumbleBit [13]. Furthermore, even with these limitations, Trilero 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 3.1.

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 [13], this type of attacks (called Potato attack in [13]) could be mitigated by aggregating payments or adding noise à la differential privacy.

5.1.2. Practical Deployment. In this section, we discuss the practical considerations for real-life deployment of Trilero.

Hub vs tumbler functionality. Trilero, 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 Trilero is the most efficient PCH among those tumbler protocols with emphasis on privacy.

Variable payment amounts and fees. Trilero sacrifices the support of arbitrary payment amounts in favor of achieving unlinkability. While for readability, we have described Trilero 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., 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, Trilero can be extended to let the tumbler P_t charge a fee for each puzzle promise/solver pair that it processes. In particular, Trilero 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 Trilero) supports the authorization of transactions across different cryptocurrencies. However, deploying Trilero 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 Trilero 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 3, 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 [13], [31]. 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 3. 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.

6. Related Work

On-chain tumblers. Several prior works exist where a centralized tumbler assists users to mix their coins [6], [7], [8], [32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42]. However, all these constructions heavily rely on on-chain transactions to operate, thus, hindering scalability. A^{2}L operates instead with off-chain payments, aiding the scalability of current blockchains. Also, while the aforementioned systems are restricted to one (or few) cryptocurrencies, A^{2}L relies only on widely deployed cryptographic

primitives such as digital signatures and timelocks, improving thereby the backwards compatibility of our solution and paving the way to interoperable cross-chain applications.

Payment channel networks (PCNs). In a PCN [4], parties performs payments through a path of opened channels between sender and receiver. Recent works have studied their security, privacy, and concurrency guarantees [22], [43]. We consider this research line as orthogonal to our work, since the underlying protocol requires to reveal the predecessor and successor nodes in the path to the intermediaries, which is exactly the privacy notion in a PCH, with only one intermediary (i.e., the tumbler).

7. Conclusion

We presented A^2L a novel three-party protocol to synchronize the updates between the payment channels involved in a PCH, and using which we build Trilero, a secure, privacy-preserving, interoperable, and backwards compatible PCH. Our construction relies on an adaptor signature scheme, which can be instantiations from Schnorr or ECDSA signatures. [15]. We defined and proved security and privacy of Trilero and A^2L in the UC framework. We further demonstrated that Trilero 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, Trilero provides backwards compatibility with virtually all cryptocurrencies today.

Acknowledgements

We gratefully thank Lloyd Fournier and Ida Tucker for the helpful discussions. This work has been partially supported by the European Research Council (ERC) under the European Unions Horizon 2020 research (grant agreement No 771527-BROWSEC); by Netidee through the project EtherTrust (grant agreement 2158) and PROFET (grant agreement P31621); by the Austrian Research Promotion Agency through the Bridge-1 project PR4DLT (grant agreement 13808694); by COMET K1 SBA, ABC; by Chaincode Labs; by the Austrian Science Fund (FWF) through the Meitner program and project W1255-N23.

References

[1] 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.

[2] 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.

[3] 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.

[4] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," Technical Report, 2016.

[5] 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.

[6] L. Valenta and B. Rowan, "Blindcoin: Blinded, Accountable Mixes for Bitcoin," in *Financial Cryptography and Data Security*, 2015, pp. 112–126.

[7] 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.

[8] gmaxwell (pseudonym), "CoinSwap: Transaction graph disjoint trustless trading," Forum post, 2013, https://bitcointalk.org/index.php?topic=321228.0.

[9] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," in *CCS*, 2017.

[10] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment hubs over cryptocurrencies," Cryptology ePrint Archive, Report 2017/635, 2017.

[11] R. Khalil, A. Gervais, and G. Felley, "Nocust - a securely scalable commit-chain," Cryptology ePrint Archive, Report 2018/642, 2018.

[12] 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.

[13] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "TumbleBit: An untrusted bitcoin-compatible anonymous payment hub," in *NDSS*, 2017.

[14] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," Cryptology ePrint Archive, Report 2000/067, 2000.

[15] 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.

[16] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Advances in Cryptology – CRYPTO' 91*, 1991, p. 129–140.

[17] M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge and its applications," in *STOC*. ACM, 1988, pp. 103–112.

[18] G. Castagnos and F. Laguillaumie, "Linearly homomorphic encryption from ddh," in *CT-RSA*, 2015, pp. 487–505.

[19] 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.

[20] D. Pointcheval and O. Sanders, "Short randomizable signatures," in *Topics in Cryptology - CT-RSA 2016*, 2016, pp. 111–126.

[21] 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.

[22] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," 2019.

[23] S. Noether, "Discrete logarithm equality across groups," 2018, https://www.getmonero.org/resources/research-lab/pubs/MRL-0010.pdf.

[24] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in *Theory of Cryptography Conference*, 2007, pp. 61–85.

[25] D. F. Aranha and C. P. L. Gouvêa, "RELIC is an Efficient LIbrary for Cryptography," Github Project, 2020.

[26] T. Granlund and the GMP development team, *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6th ed., 2019.

[27] *PARI/GP version* `2.12.0`, The PARI Group, Univ. Bordeaux, 2019.

[28] I. Damgård, "On the σ-protocols," Lecture Notes, University of Aarhus, Department for Computer Science, 2002.

[29] 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.

[30] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker, "Bandwidth-efficient threshold ec-dsa," pp. 266–296, 2020.

[31] 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.

[32] J. H. Ziegeldorf, F. Grossmann, M. Henze, N. Inden, and K. Wehrle, "CoinParty: Secure Multi-Party Mixing of Bitcoins," in *CODASPY*. ACM, 2015, pp. 75–86.

[33] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin," in *ESORICS'14*, 2014, pp. 345–364.

[34] ——, "P2P Mixing and Unlinkable Bitcoin Transactions," in *NDSS*. The Internet Society, 2017.

[35] T. Ruffing and P. Moreno-Sanchez, "ValueShuffle: Mixing Confidential Transactions for Comprehensive Transaction Privacy in Bitcoin," in *Financial Cryptography and Data Security*, 2017, pp. 133–154.

[36] I. A. Seres, D. A. Nagy, C. Buckland, and P. Burcsi, "MixEth: Efficient, trustless coin mixing service for Ethereum," Tech. Rep. 341, 2019.

[37] G. Bissias, A. P. Ozisik, B. N. Levine, and M. Liberatore, "Sybil-Resistant Mixing for Bitcoin," in *WPES*. ACM, 2014, pp. 149–158.

[38] S. Meiklejohn and R. Mercer, "Möbius: Trustless Tumbling for Transaction Privacy," Tech. Rep. 881, 2017.

[39] P. Moreno-Sanchez, T. Ruffing, and A. Kate, "Pathshuffle: Credit mixing and anonymous payments for ripple," *PoPETs*, vol. 2017, no. 3, p. 110, 2017.

[40] E. Heilman, F. Baldimtsi, and S. Goldberg, "Blindly Signed Contracts: Anonymous On-Blockchain and Off-Blockchain Bitcoin Transactions," Tech. Rep. 056, 2016.

[41] P. Fauzi, S. Meiklejohn, R. Mercer, and C. Orlandi, "Quisquis: A new design for anonymous cryptocurrencies," in *Advances in Cryptology – ASIACRYPT 2019*. Cham: Springer International Publishing, 2019, pp. 649–678.

[42] M. Tran, L. Luu, M. S. Kang, I. Bentov, and P. Saxena, "Obscuro: A bitcoin mixer using trusted execution environments," in *Proceedings of ACSAC 2018*, 2018, pp. 692–701.

[43] 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.

[44] Y. Lindell, "Fast secure two-party ecdsa signing," Cryptology ePrint Archive, Report 2017/552, 2017.

[45] 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.

[46] J. Katz, U. Maurer, B. Tackmann, and V. Zikas, "Universally composable synchronous computation," in *TCC*, 2013, pp. 477–498.

[47] J. Camenisch and A. Lysyanskaya, "A formal treatment of onion routing," in *Advances in Cryptology – CRYPTO 2005*, pp. 169–187.

[48] "Lightning network specifications," https://github.com/lightningnetwork/lightning-rfc.

Appendix

1. 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 [15] 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 [22]. 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 1 (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 2 (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 3 (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:*

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

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 4 (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:*

$\mathsf{aSigForge}_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda)$

$\mathcal{Q} := \emptyset$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(1^\lambda)$
$m \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{S}}(\cdot), \mathcal{O}_{\mathrm{pS}}(\cdot,\cdot)}(\mathsf{pk})$
$(Y, y) \leftarrow \mathsf{GenR}(1^\lambda)$
$\hat{\sigma} \leftarrow \mathsf{PreSig}(\mathsf{sk}, m, Y)$
$\sigma \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{S}}(\cdot), \mathcal{O}_{\mathrm{pS}}(\cdot,\cdot)}(\hat{\sigma}, Y)$
return $(m \notin \mathcal{Q} \wedge \mathsf{Vf}(\mathsf{pk}, m, \sigma))$

$\mathcal{O}_{\mathrm{S}}(m)$

$\sigma \leftarrow \mathsf{Sig}(\mathsf{sk}, m)$
$\mathcal{Q} := \mathcal{Q} \cup \{m\}$
return σ

$\mathcal{O}_{\mathrm{pS}}(m, Y)$

$\hat{\sigma} \leftarrow \mathsf{PreSig}(\mathsf{sk}, m, Y)$
$\mathcal{Q} := \mathcal{Q} \cup \{m\}$
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 3, since we require that even maliciously produced pre-signatures can always be completed into valid signatures. The following definition formalizes this property.

Definition 5 (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 6 (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*

$\mathsf{aWitExt}_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda)$

$\mathcal{Q} := \emptyset$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(1^\lambda)$
$(m, Y) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{S}}(\cdot), \mathcal{O}_{\mathrm{pS}}(\cdot,\cdot)}(\mathsf{pk})$
$\hat{\sigma} \leftarrow \mathsf{PreSig}(\mathsf{sk}, m, Y)$
$\sigma \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{S}}(\cdot), \mathcal{O}_{\mathrm{pS}}(\cdot,\cdot)}(\hat{\sigma})$
$y' := \mathsf{Ext}(\mathsf{pk}, \sigma, \hat{\sigma}, Y)$
return $(m \notin \mathcal{Q} \wedge (Y, y') \notin R$
$\quad \wedge \mathsf{Vf}(\mathsf{pk}, m, \sigma))$

$\mathcal{O}_{\mathrm{S}}(m)$

$\sigma \leftarrow \mathsf{Sig}(\mathsf{sk}, m)$
$\mathcal{Q} := \mathcal{Q} \cup \{m\}$
return σ

$\mathcal{O}_{\mathrm{pS}}(m, Y)$

$\hat{\sigma} \leftarrow \mathsf{PreSig}(\mathsf{sk}, m, Y)$
$\mathcal{Q} := \mathcal{Q} \cup \{m\}$
return $\hat{\sigma}$

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 7 (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.*

2. Castagnos-Laguillaumie Encryption Scheme

The main reason for using the Castagnos-Laguillaumie (CL) [18], [19] 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 [44] 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 [19].

3. Security and Privacy Model

3.1. Preliminaries. We define our security and privacy model modularly by leveraging the Universal Composability (UC) framework from Canetti [14]. More precisely, we rely on the synchronous version of global UC framework (GUC) [24]. 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 3.3. 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 called Trilero in Appendix E. The security proofs for A^{2}L and Trilero are given in Appendix D and Appendix E.2, 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 [13], [22], [43], we consider the static corruption model, that is, the adversary commits to the identifiers of the parties it wishes to corrupt ahead of time.

Communication model. We consider a synchronous communication network, where communication proceeds in discrete rounds. We follow [45] (which in turn follows [46]), and formalizes the notion of rounds via an 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 [45], we treat the clock functionality as a global ideal functionality defined in the GUC model [24]. 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. Hence, if a party P sends a message m to party Q in round r, party Q receives the message at the beginning of round $r+1$. Furthermore, Q is sure that the message was sent by P. 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 [45].

Additionally, we use the secure transmission functionality $\mathcal{F}_{\mathsf{smt}}$, as defined in [14], 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 [47], 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 ideal functionality $\mathcal{F}_{\mathsf{GC}}$ [15], which defines generalized channels, which can be seen as an extension of payment channels. The ideal functionality provides all the backbone necessary for handling payment channels, such as the following interfaces: Create, used for opening a payment channel, Close, used for closing a payment channel, and Update, used to update the balances of the parties involved in the payment channel.

Universal composability. We briefly overview the notion of secure realization in the UC framework [14]. 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. 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.

Definition 8 (Universal Composability). *A protocol π UC-realizes an ideal functionality $\mathcal{F}$ if for any* PPT *adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$, such that for any environment $\mathcal{E}$, the ensembles $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{E}}$ and $\mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are computationally indistinguishable.*

3.2. 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 6, 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 tokens.

$\mathcal{F}_{\mathsf{A^2L}}$ provides three interfaces, which are depicted in Figure 6. The Registration interface allows a party to obtain

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 $(\mathsf{promise}, \mathsf{pid})$ to $P_r, \mathcal{S}$ and $(\mathsf{promise}, \mathsf{pid}')$ to $P_s, \mathcal{S}$.

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 $(\mathsf{solve}, \top)$ to P_s 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 $\perp$ to P_r and abort.
- Else send $\top$ to P_r.

Figure 6: Ideal functionality $\mathcal{F}_{\mathsf{A^2L}}$.

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 3.1. 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 continues 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 (lock) can only be solved, if there has been a corresponding execution of puzzle solver with for this 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, it is this randomized puzzle pid' that 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 3), which can be realized with $\mathcal{F}_{\mathsf{anon}}$ [47] 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 [9], [13].

3.3. Payment Channel Hub (PCH). Here, we formalize the notion of payment channel hub (PCH), more precisely, we give a formalization of Trilero. We use the notation described in Section 2 for payment channels.

Ideal functionality. $\mathcal{F}_{\mathsf{PCH}}$ ideal functionality makes use of $\mathcal{F}_{\mathsf{GDC}}, \mathcal{F}_{\mathsf{GC}}$, and our previously defined $\mathcal{F}_{\mathsf{A^2L}}$ ideal functionalities, hence, it is defined in $(\mathcal{F}_{\mathsf{GDC}}, \mathcal{F}_{\mathsf{GC}}, \mathcal{F}_{\mathsf{A^2L}})$-hybrid model. $\mathcal{F}_{\mathsf{PCH}}$ is shown in Figure 7.

$\mathcal{F}_{\mathsf{PCH}}$ 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 [43], [48], which in our case are handled via simulator calls to the $\mathcal{F}_{\mathsf{GC}}$ ideal functionality defined in [15]. Lastly, Pay handles the payment operation from the sender P_s to the receiver P_r through the via P_t by making use of $\mathcal{F}_{\mathsf{A^2L}}$.

Discussion. We discuss here how the ideal functionality captures the security and privacy notions of interest for payment hubs as defined in Section 3.1.

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.

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

Figure 7: Ideal functionality $\mathcal{F}_{\mathsf{PCH}}$.

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 C.2 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, therefore, the amounts to not help in differing the payments.

4. Security Analysis of A^{2}L

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 [14], and in the presence of *malicious adversaries* with *static corruptions*. We recall the theorem stated in Section 4, which we prove here.

Theorem 1. *Let* COM *be a secure commitment scheme,* NIZK *be a non-interactive zero-knowledge scheme,* $\Sigma, \tilde{\Sigma}$ *be* EUF-CMA *secure signature schemes,* R *be a hard relation,* $\Xi_{R,\Sigma}$ *be a secure adaptor signature scheme, and* Ψ *be an* IND-CPA *secure encryption 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 3.3).

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 (tid, σ), 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,

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

  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, $\sigma := \mathsf{UnblindSig}(\mathsf{decom}, \sigma')$. If $\mathsf{Vf}(\mathsf{pk}_t^{\widetilde{\Sigma}}, \mathsf{tid}, \sigma) \neq 1$, then simulate P_s aborting. Otherwise, randomize the signature, $\sigma^* \leftarrow \mathsf{RandSig}(\sigma)$, store a copy of (tid, σ^*) 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 $\mathsf{V}_{\mathsf{NIZK}}(\pi, \mathsf{com}) \neq 1$, 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 (tid, σ^*) that was previously stored, sign the message (transaction), $\sigma'_r \leftarrow \mathsf{Sig}(\mathsf{sk}_r^{\Sigma}, m')$, and send $(\mathsf{promise{-}req}, (\mathsf{tid}, \sigma^*), \sigma'_r)$ to P_t.
- Upon $(\mathsf{promise}, A, \pi_\alpha, c_\alpha, \hat{\sigma}'_t)$ from $\mathcal{A}$ (on behalf of P_t), check if $\mathsf{V}_{\mathsf{NIZK}}(\pi_\alpha, (c_\alpha, A)) \neq 1$ 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 ciphertext and statement,

  $$(c'_\alpha, \beta) \leftarrow \mathsf{RandCtx}(c_\alpha),$$
  $$A' \leftarrow A^\beta,$$

  store $\Pi := (\beta, (\mathsf{pk}_t^{\Sigma}, \mathsf{pk}_r^{\Sigma}), m', (\hat{\sigma}'_t, \sigma'_r))$ and $\ell := (A', c'_\alpha)$, and send ℓ to P_s.

Case P_t is honest and P_r is corrupted

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

- If $\mathsf{tid} \in \mathcal{T}$ or $\mathsf{Vf}(\mathsf{pk}_t^{\widetilde{\Sigma}}, \mathsf{tid}, \sigma^*) \neq 1$, 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 $(A, \alpha) \leftarrow \mathsf{GenR}(1^\lambda)$, encrypt α and prove that the encrypted value is the witness of the statement A,

  $$c_\alpha \leftarrow \mathsf{Enc}(\mathsf{pk}_t^{\Psi}, \alpha),$$
  $$\pi_\alpha \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists \alpha \mid c_\alpha = \mathsf{Enc}(\mathsf{pk}_t^{\Psi}, \alpha) \wedge A = g^\alpha\}, \alpha),$$

  pre-sign the message (transaction) $\hat{\sigma}'_t \leftarrow \mathsf{PreSig}(\mathsf{sk}_t^{\Sigma}, m', A)$, and send $(\mathsf{promise}, A, \pi_\alpha, c_\alpha, \hat{\sigma}'_t)$ 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 $\ell := (A', c'_\alpha)$ that was previously stored, randomize the ciphertext and statement,

  $$(c''_\alpha, \tau) \leftarrow \mathsf{RandCtx}(c'_\alpha),$$
  $$A'' \leftarrow (A')^\tau,$$

  pre-sign the message (transaction), $\hat{\sigma}_s \leftarrow \mathsf{PreSig}(\mathsf{sk}_s^{\Sigma}, m, A'')$. Send $(\mathsf{solve{-}req}, c_{\alpha''}, \hat{\sigma}_s)$ 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 $(\mathsf{solve{-}req}, c''_\alpha, \hat{\sigma}_s)$ to P_t, proceed as follows:

- Decrypt the input ciphertext, adapt the input pre-signature, and sign the message (transaction),

  $$\alpha'' := \mathsf{Dec}(\mathsf{sk}_t^{\Psi}, c''_\alpha),$$
  $$\sigma_s := \mathsf{Adapt}(\hat{\sigma}_s, \alpha''),$$
  $$\sigma_t \leftarrow \mathsf{Sig}(\mathsf{sk}_t^{\Sigma}, m).$$

  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 $\tilde{\Sigma}$. 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 $\mathsf{pk}_t^{\tilde{\Sigma}}$ 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 $(\mathsf{tid}^*, \sigma^*)$, 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 $(\mathsf{tid}^*, \sigma^*)$ as input, and furthermore, we have that $\mathsf{Vf}(\mathsf{pk}_t^{\Sigma}, \mathsf{tid}^*, \sigma^*) = 1$ 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. Moreover, as every message (token identifier) uniquely identifies a session, we have that $(\mathsf{tid}^*, \sigma^*)$ 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 $\Pr[\mathsf{fail} \mid \mathcal{H}_3] \geq \frac{1}{\mathsf{poly}(\lambda)}$, and consider the following intermediate hybrid.

- Hybrid $\mathcal{H}_3^*$: The initial ciphertext c_α is computed as an encryption of a fixed string (e.g., 0) padded to the appropriate length.

We note that the experiment no longer uses the secret key sk_t^{Ψ}, thus by the IND-CPA security of the homomorphic encryption scheme Ψ we have that

$$\Pr[\mathsf{fail} \mid \mathcal{H}_3^*] \geq \Pr[\mathsf{fail} \mid \mathcal{H}_3] - \mathsf{negl}(\lambda).$$

At this point all that remains is to show that the probability of fail happening in $\mathcal{H}_3^*$ cannot be inverse polynomial. This is implied by the hardness of the relation R and unforgeability of the adaptor signature scheme $\Xi_{R,\Sigma}$. Since the unforgeability of the adaptor signature also implies the hardness of the relation R, we 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_r^{Σ} 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_t^{Σ}, 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_t^{Σ}, 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_t^{Σ}.

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_s^Σ 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 $(\beta, (\mathsf{pk}_t^\Sigma, \mathsf{pk}_r^\Sigma), m', (\hat{\sigma}_t', \sigma_r'))$, 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 i 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 1. □

5. Description and Security Analysis of Trilero

5.1. Protocol Description of Trilero. In Trilero, we combine $\mathrm{A^2L}$ 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, Trilero relies on two contracts built upon $\mathrm{A^2L}$.

In a bit more detail, first P_t and P_r execute the PuzzlePromise protocol from $\mathrm{A^2L}$ to get the input required to establish the following $\mathrm{A^2L}$-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 is the output (along with ℓ) from the PuzzlePromise protocol in $\mathrm{A^2L}$, 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 that $\mathcal{B}$ supports 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 (Schnorr and ECDSA in our case), 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 lock ℓ (as output by the PuzzlePromise protocol) to P_s. Then, P_s and P_t execute the PuzzleSolver protocol to get the input required to establish the following $\mathrm{A^2L}$-Solve$(P_s, P_t, \ell, \mathsf{amt}, \varsigma, t)$ contract:

1) If before t, P_t sends P_s the randomized solution α' to the cryptographic challenge encoded in ℓ, 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).

Finally, P_s gets the randomized solution α' to the challenge encoded in the lock ℓ. Then, P_s sends α' to P_r who can complete the $\mathrm{A^2L}$-Promise contract with the final valid signature $\sigma := \mathsf{Open}(\Pi, \alpha')$.

Our Trilero protocol can be seen in Figure 8, where we omit the key material to ease readability and assume that they are given as implicit input to the subprotocols.

5.2. Security Analysis of Trilero. Here we prove the following theorem above Trilero, which was previously stated in Section 4

Theorem 2. *The protocol described in Figure 8, UC-realizes* $\mathcal{F}_{\mathsf{PCH}}$ *in the* $(\mathcal{F}_{\mathsf{A^2L}}, \mathcal{F}_{\mathsf{GDC}}, \mathcal{F}_{\mathsf{smt}})$*-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 3.1 and discussed in Appendix C.3). 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 D. Furthermore, as was discussed in

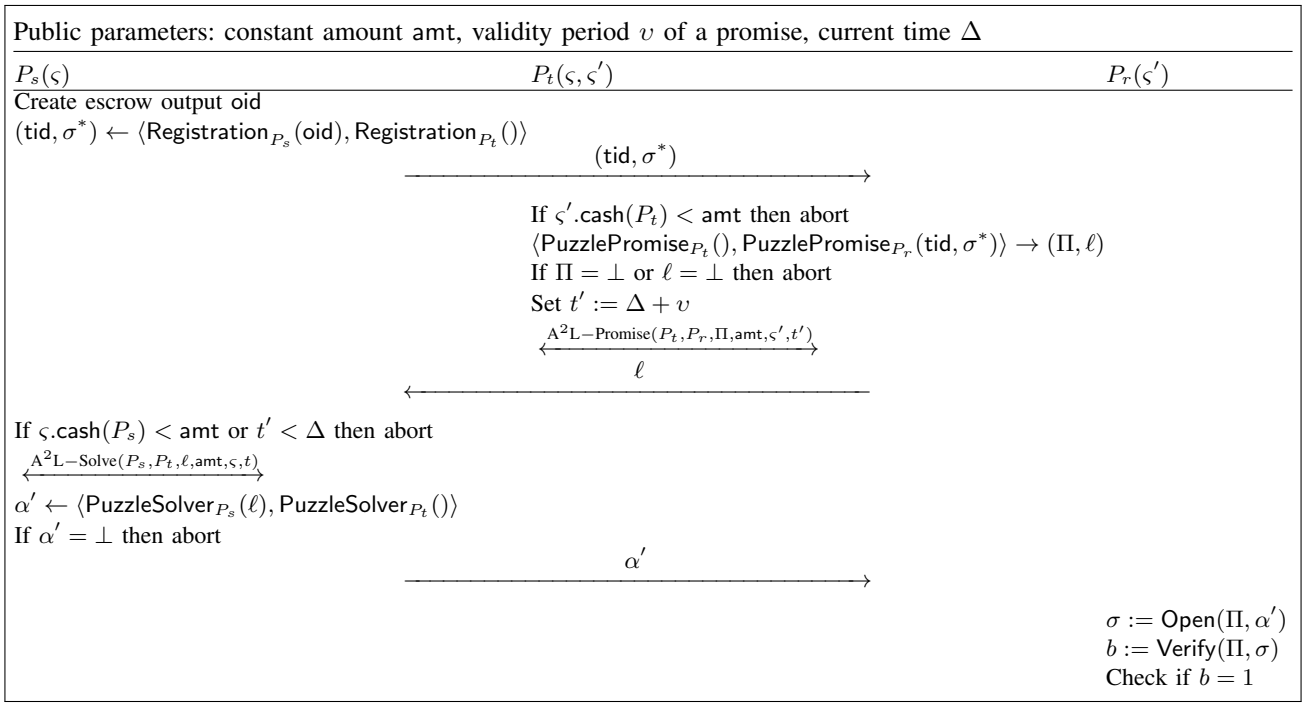

Figure 8: Trilero protocol (keys removed as inputs to subprotocols for readability).

Appendix C.2, $\mathcal{F}_{\mathsf{A^2L}}$ also satisfies the unlinkability property, hence, the same argument for unlinkability applies here too, with the exception of the operations of $\mathcal{F}_{\mathsf{PCH}}4$ 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 $\mathcal{F}_{\mathsf{PCH}}$, and since the parties do not obtain any secret input $\mathcal{S}$ becomes trivial to realize. □

6. 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 2.

6.1. Schnorr-based Construction. Let $\mathbb{G}$ be an elliptic curve group of prime order q with a generator g, and let $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$ 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 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 sued during puzzle solver). This shared key generation can be done as explained in [22].

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, encrypts it under its own public key using the homomorphic encryption scheme, and sends the ciphertext c_α along with $A := g^\alpha$ to P_r (lines 1-7 in Figure 9). Here we require a zero-knowledge proof (denoted by π_α in the puzzle promise protocol) proving that the ciphertext c_α encrypts the discrete logarithm of A (line 4 in Figure 9). If we do not have such a proof, then P_t can perform the following attack to link a potential payer and payee. At a particular epoch, P_t chooses a payee P_r^* it wants to attack, and when performing the puzzle solver protocol with this party it encrypts a value that is different from the discrete logarithm of A. Then, during the puzzle solver protocol, when a payer P_s^* performs the protocol with P_t, the check $(A')^\tau = A''$ (line 16 in Figure 10) will fail, and P_s^* 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

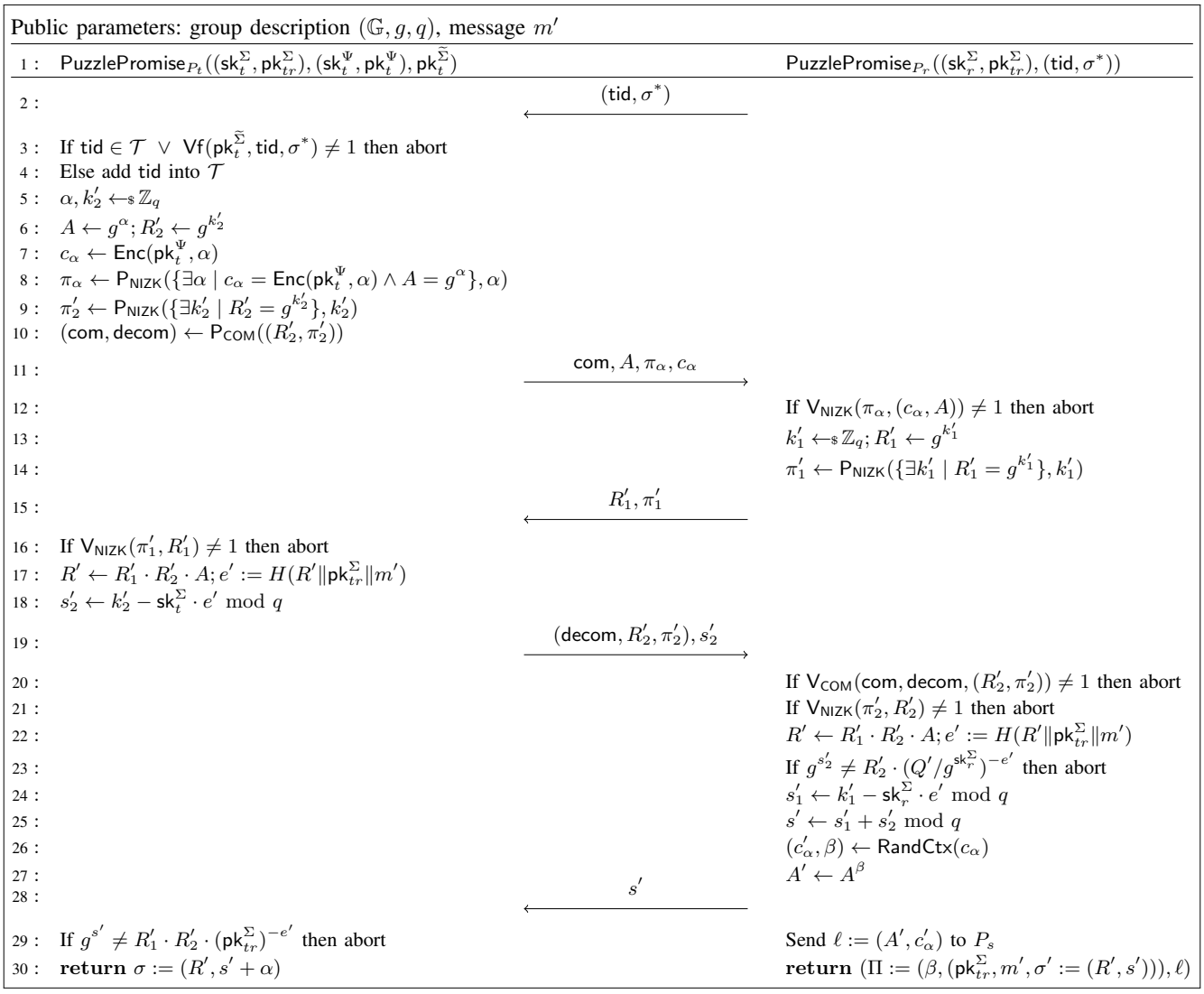

Figure 9: Puzzle promise protocol of Schnorr-based construction.

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 $c_\alpha := \mathsf{Enc}(\mathsf{pk}_t^\Psi, \alpha)$ and $A := g^\alpha$ from P_t at the beginning of the puzzle promise protocol, and at the end of the protocol P_r randomizes c_α as $(c_\alpha', \beta) \leftarrow \mathsf{RandCtx}(c_\alpha)$ (this is possible due to the homomorphic properties of CL encryption scheme that we are using), and using the obtained β, randomizes A as $A' \leftarrow A^\beta$. The puzzle promise protocol finishes with P_r sending these randomized values to P_s (lines 22 and 24 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 samples a random value τ, and randomizes the ciphertext it received from P_r, as $(c_\alpha'', \tau) \leftarrow \mathsf{RandCtx}(c_\alpha')$ (line 6 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 c_α'' to P_t (lines 7-11 in Figure 10). At this point, P_t decrypts c_α'' to obtain the value $\alpha'' := \alpha \cdot \beta \cdot \tau$ (line 10 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 24-25 in

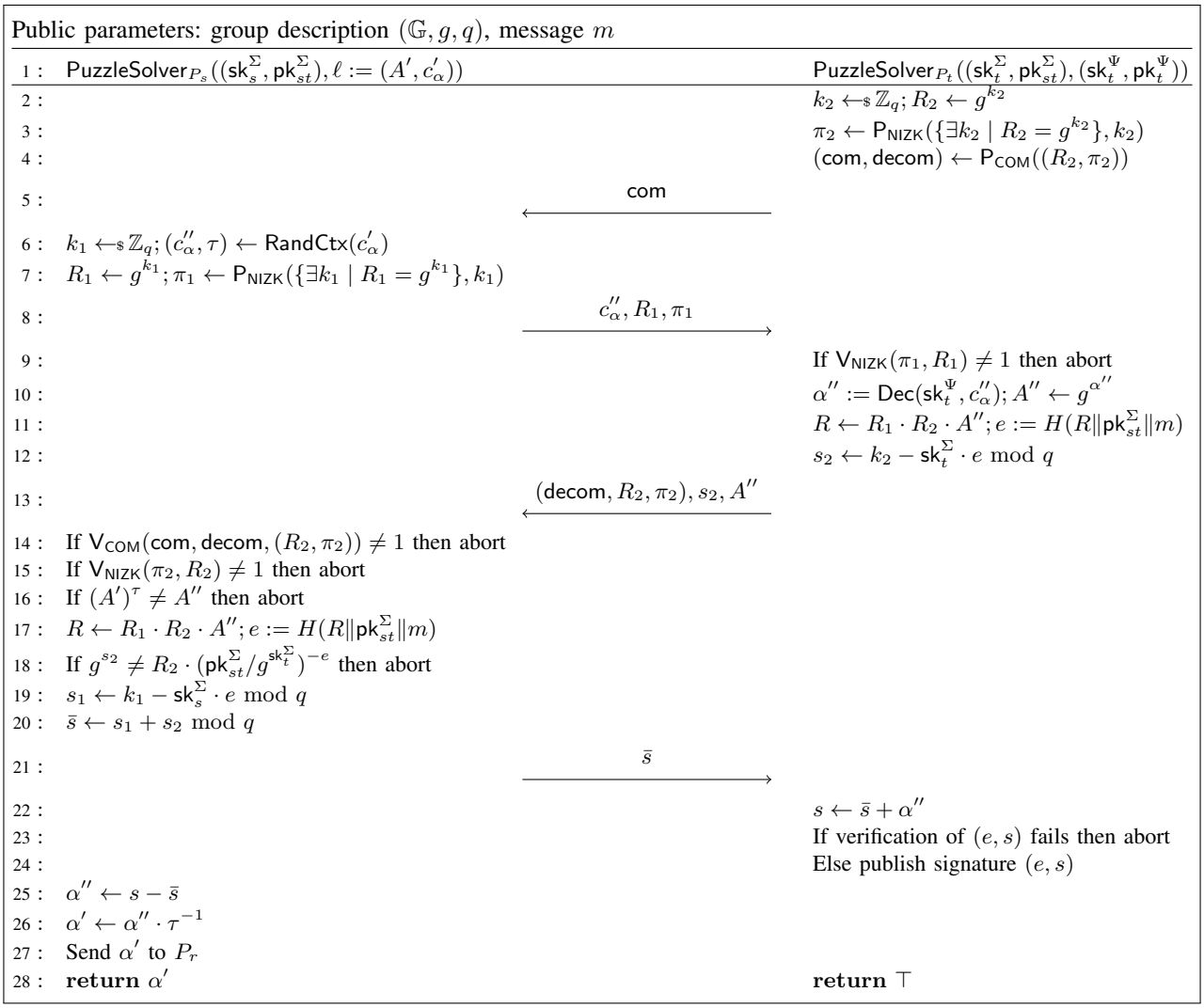

Figure 10: Puzzle solver protocol of Schnorr-based construction.

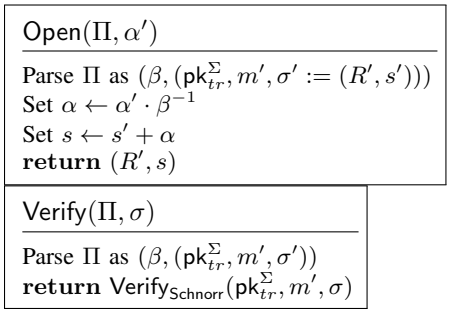

Figure 11: Open and verify algorithms of Schnorr-based construction.

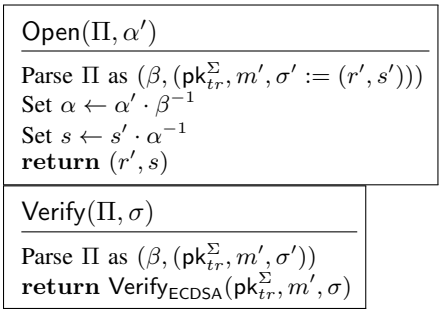

Figure 12: Open and verify algorithms of ECDSA-based construction.

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 26-27 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.

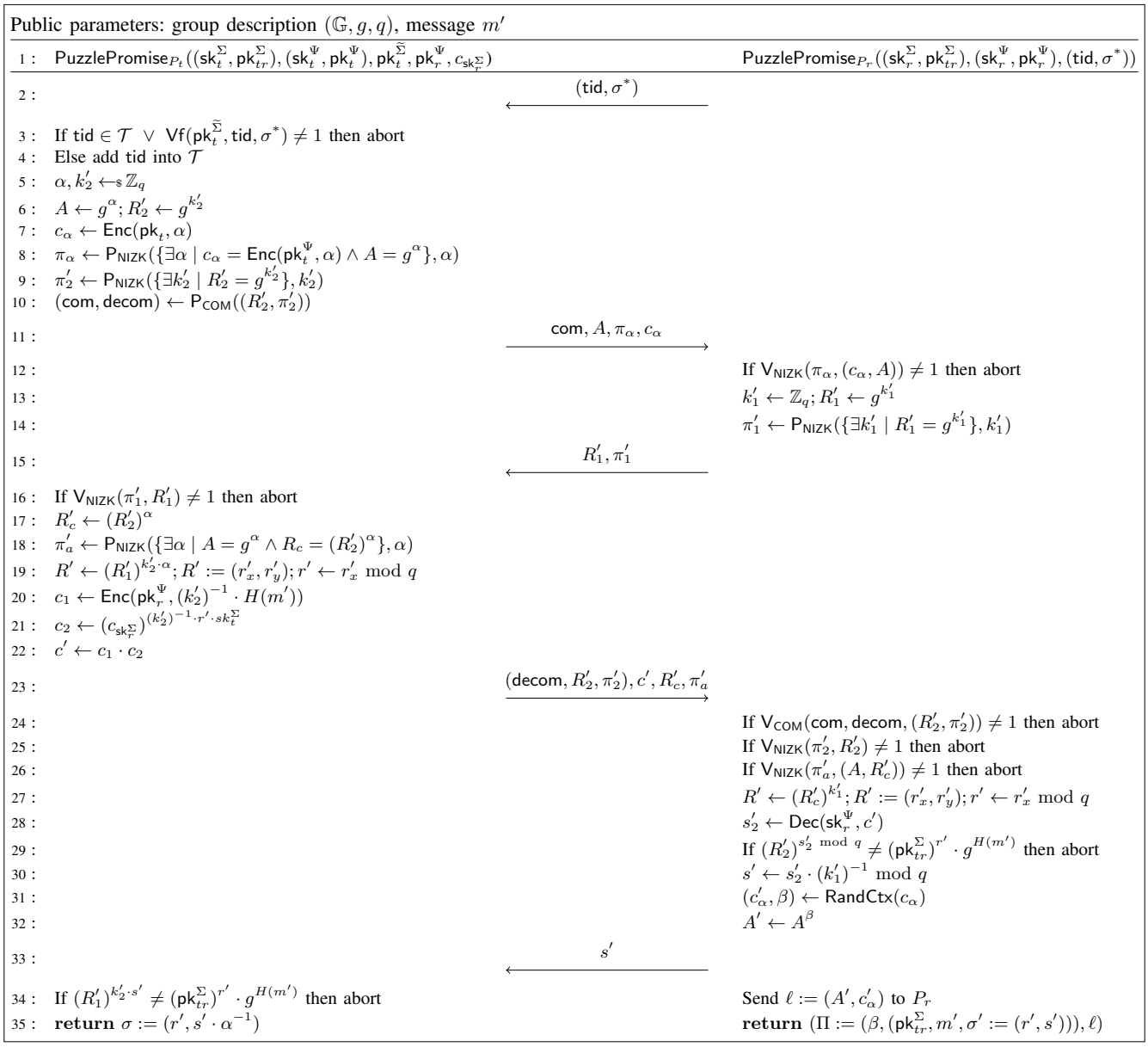

Figure 13: Puzzle promise protocol of ECDSA-based construction.

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

Let $\mathbb{G}$ be an elliptic curve group of order q with a generator g, and let $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$ be a collision resistant hash function. Additionally, let COM, 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 [19], [44]), 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 $\mathsf{pk}_{st}^{\Sigma}$ computed between P_s and P_t.

The puzzle promise protocol runs similarly 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 $R' = k_1' \cdot k_2' \cdot \alpha$, where α is unknown to P_r (lines 4-14 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

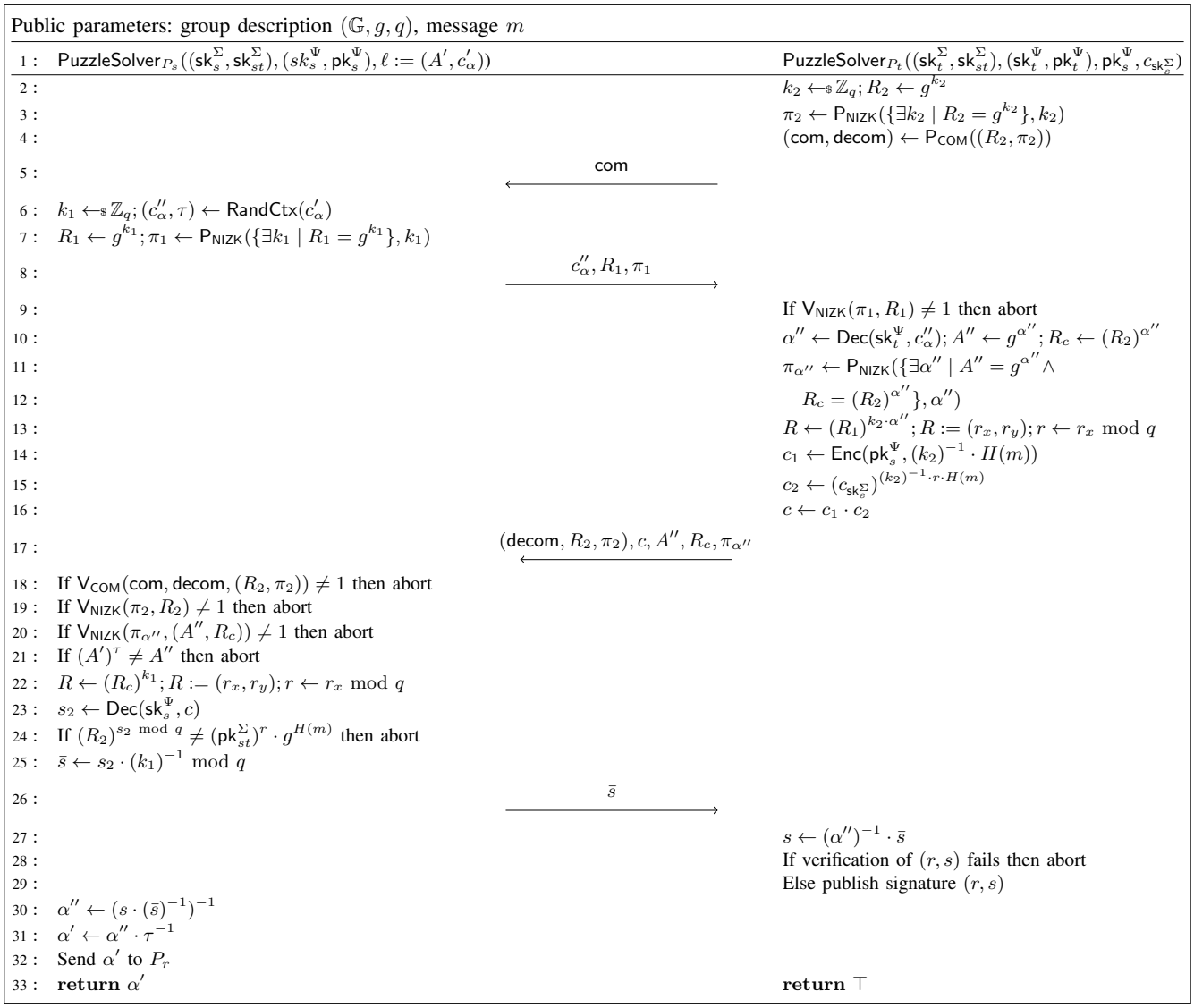

Figure 14: Puzzle solver protocol of ECDSA-based construction.

encryption scheme. However, P_t does not embed the inverse of α into the signature (line 15 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 20-24 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 $c_\alpha := \mathsf{Enc}(\mathsf{pk}_t^\Psi, \alpha)$ and $A := g^\alpha$ from P_t at the beginning of the puzzle promise protocol, and at the end of the protocol P_r randomizes the values c_α and A using a randomness β. The puzzle promise protocol finishes with P_r sending these randomized values 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.