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

Erkan Tairi, Pedro Moreno-Sanchez, Matteo Maffei
TU Wien
{erkan.tairi,pedro.sanchez,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 fall short of other fundamental properties, such as interoperability, fungibility, and efficiency.**

In this work, we present A^{2}L, the first PCH to achieve all aforementioned properties. A^{2}L builds on a novel cryptographic primitive that 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. We prove the security and privacy guarantees of A^{2}L in the Universal Composability framework and present two provably secure instantiations based on Schnorr and ECDSA 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 and computation complexity that is linear, as opposed to binomial, in the security parameter. In practice, our ECDSA-based construction is 5x faster and requires 65x less bandwidth, while our Schnorr-based construction is 8x faster and requires 95x less bandwidth.

I. Introduction

The increasing adoption of cryptocurrencies has raised scalability issues [11] that go beyond the rapidly growing blockchain size. For instance, the permissionless nature of the consensus algorithm underlying widely deployed cryptocurrencies such as Bitcoin and Ethereum strictly limits their transaction throughput to tens of transactions per second at best [11], which contrasts with the throughput of centralized payment networks such as Visa that supports peaks of up to 47,000 transactions per second [48].

Among the several efforts to mitigate these scalability issues [28], [29], [42], 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 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 economical 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.

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

The problem with this simple construction is that in order to pay different people, a user should establish a channel with each of them, which is computationally and financially prohibitive, as this party would have to lock an amount of coins proportional to the number of users she wants to transact with.

A. Payment Channel Hubs (PCHs)

PCHs offer a solution to the aforementioned problem. The idea is to let each user open a channel with a central party, called the *tumbler*, which is in charge of mediating payments between each pair of users. In particular, if the sender wants to transfer x coins to the receiver, the sender pays $x + \mathit{fee}$ to the tumbler, which then forwards x coins to the receiver, where *fee* denotes a fee charged by the tumbler to conduct the transaction. Such a naïve construction, despite being still deployed in many gateways, suffers from obvious *security and privacy issues*: the tumbler could steal coins [5], [49] from honest users (e.g., by simply not forwarding a payment) as well as identify who is paying to whom [1], [5].

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 (the tumbler should not able to link the sender and receiver of a given payment) and value privacy (the tumbler should not learn the transaction value). As these 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 *interoperability*: the tumbler should be able to mediate payments in different cryptocurrencies (e.g., the sender transfering bitcoins and the receiver getting ethers), thereby enabling cross-chain applications like exchanges and cross-currency mixing.

Finally, a desirable property in any currency is *fungibility*, which means that all coins should be indistinguishable from

	Atomicity	Unlinkability	Value Privacy	Fungibility	Interoperability (Required functionality)
BOLT [21]	●	●	●	●	○ (Zcash)
Perun [16]	●	○	◐[1]	○	○ (Ethereum)
NOCUST [27]	●	○	◐[1]	○	○ (Ethereum)
Teechain [30]	●	●	●	●	◐[2] (Trusted Execution Environment)
TumbleBit [22]	●	●	◐[1]	○	◐[3] (HTLC-based currencies)
A^2L	●	●	◐[1]	●	● (Digital signature verification and time locks)

[1] The tumbler learns the payment values; [2] Every user must run a TEE; [3] Not supported by scriptless cryptocurrencies (e.g., Monero).

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

each other: in the specific case of PCHs, payments performed through the tumbler should look the same as standard payments, as otherwise, e.g., coins produced by a tumbler might be considered tainted and not accepted by certain users.

B. State-of-the-art in PCH

BOLT [21] is an off-chain cryptographic protocol for PCHs that provides strong anonymity and value privacy guarantees by leveraging the zero-knowledge proofs of the underlying Zcash cryptocurrency. BOLT also inherits the fungibility guarantees provided by Zcash.[2] BOLT, however, is only compatible with ZCash since it requires zero-knowledge proofs.

Perun [16] is an off-chain channel system that relies on Turing-complete smart contracts to support payment channels. Moreover, Perun builds the PCH upon virtual channels, a 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. Perun achieves a weak value privacy property, since the value of the individual transactions between Alice and Bob is hidden, but the aggregated value (over the lifespan of the channel) is revealed. Additionally, Perun lacks fungibility, as transactions encode a logic that makes them distinguishable from transactions performed by other contracts, as well as interoperability, as it works only in Ethereum.

NOCUST [27], is an off-chain payment protocol that relies on an untrusted operator to manage the off-chain payments among users. The operator periodically includes a summary on-chain that includes the balances and the transactions for public verifiability. As Perun, NOCUST does not provide unlinkability against a malicious intermediate (i.e., operator) who also learns the transacted values. NOCUST also lacks fungibility and interoperability for the same reasons as already mentioned for PERUN.

TeeChain [31] is a payment channel protocol that leverages trusted execution environments (e.g., Intel SGX) to manage off-chain payments and the handling of disputes. Thus, users are required to run a TEE, which hinders the widespread deployment of this approach.

TumbleBit [22] is a cryptographic protocol for PCHs that makes transactions unlinkable (i.e., the tumbler does not learn who is paying whom). By fixing the same value for all transactions, TumbleBit achieves a value privacy property that is weaker than the one provided by Bolt, called privacy of the compatible interaction graph: the tumbler learns how many coins each party sends and receives in aggregated form, but not how much is sent in each transaction. However, due to the underlying cut-and-choose technique, TumbleBit requires computation and communication costs that grow binomially in the security parameter. For instance, enforcing only 80 bits of security requires messages of size between 250 and 400 KB for a single payment, which implies running times of up to 10 seconds. Moreover, TumbleBit relies on the hash-time lock contract (HTLC), a Bitcoin script-based construction that allows for payments conditioned on solving a cryptographic challenge, that is, obtaining the preimage of a hash function. This, however, limits the deployment of TumbleBit to those cryptocurrencies supporting HTLC, ruling out scriptless ones such as Ripple, Stellar or Monero. Furthermore, it hinders fungibility as multisig HTLC-based payments are clearly distinguishable from standard payments.

We summarize the properties achieved by each PCH construction in Table I. Most notably, all state-of-the-art PCHs fail to achieve at least one of the aforementioned properties and, in particular, all of them fall short of interoperability: 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, further suffering from a high communication and computation complexity.

This state of affairs leads to the following question: *Is it possible to have a PCH interoperable with virtually all cryptocurrencies that is practical and achieves the security and privacy notions of interest?*

C. Our Contributions

This work answers the previous question in the affirmative and presents the first secure, privacy-preserving, interoperable, and fungibility-preserving PCH cryptographic construction, whose communication and computational complexity is just linear in the security parameter. Specifically,

- We introduce a novel cryptographic primitive called anonymous atomic locks (A^2L), which intuitively 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. We model A^2L as well as its security and anonymity properties (namely, atomicity and unlinkability) in the Universal Composability (UC) framework [7]. We leverage the resulting composability guarantees to show how A^2L can be employed as cornerstone in the design of a fully-fledged PCH.
- We give two concrete instantiations, one based on Schnorr and another one based on the ECDSA signature scheme.

[2]Here we consider only coins held at shielded addresses that have not been tainted by combining them with unshielded addresses [25].

While Schnorr provides the most efficient protocol in terms of communication and computation overhead, ECDSA is arguably the most widely deployed signature scheme in practice, thereby achieving a high degree of interoperability (e.g., we can realize a tumbler receiving bitcoins and forwarding ethers). Notice also that it is possible to combine Schnorr and ECDSA-based constructions if they are instantiated over the same group [36]. By dispensing from HTLCs, our instantiation offers the highest degree of interoperability among the state-of-the-art PCHs: e.g., Ripple and Stellar support ECDSA and Schnorr but not HTLCs, and we further show how to integrate A^{2}L in a scriptless cryptocurrency such as DLSAG-based Monero [38], a variant of Monero proposed to add support for payment channels that is being considered as a hard fork in the Monero community [40].

- Our A^{2}L instantiations incur communication and computation costs that are linear in the security parameter. Additionally, we implemented both of them, showing that they require a running time of less than 300ms for ECDSA and 80ms for Schnorr. Furthermore, they require 21.3KB for ECDSA and less than 15.3KB for Schnorr. When compared to TumbleBit, the most interoperable PCH prior to this work, which requires binomial communication and computation complexity, ECDSA-based A^{2}L is 3x faster and requires 15x less bandwidth while Schnorr-based A^{2}L is 8x faster and requires 21x less bandwidth. These results demonstrate that A^{2}L is the most efficient Bitcoin-compatible PCH. Furthermore, A^{2}L transactions are indistinguishable from standard transactions in that they rely on neither multisigs nor HTLCs, thereby offering fungibility guarantees.

II. Problem Definition

In this section, we introduce and formalize the notion of anonymous atomic lock (A^{2}L)..

Key Ideas. An A^{2}L is a three-party cryptographic primitive composed of five protocols: KGen, Promise, Pay, Open, and Verify. Their behaviour is illustrated in Figure 1. KGen realizes the opening of a payment channel between a user and the tumbler. This (on-chain) protocol is carried out once to open the channel while the rest of the protocols can be carried arbitrary many times (off-chain) while the channel is open.

The overall process starts with the execution of the promise protocol between the tumbler and the receiver. This protocol is crucial for atomicity as it allows the tumbler to commit to a payment (i.e., promise Π) to the receiver that is only enforceable if the receiver solves a cryptographic challenge ℓ (e.g., obtaining the discrete logarithm of an element), which we call *lock* in this paper. The tumbler is the only one knowing the solution of this lock ℓ at this point. At the same time, this protocol also ensures that as soon as the receiver knows the solution of the lock, the promise can be fulfilled and he can get the coins, incentivizing thereby the receiver to enter in the next phase, which is triggered by sending the lock ℓ to the sender.

At this point, the sender can perform the pay operation with the tumbler to obtain the solution of the lock. However, note that if the sender naively inputs ℓ into the pay operation,

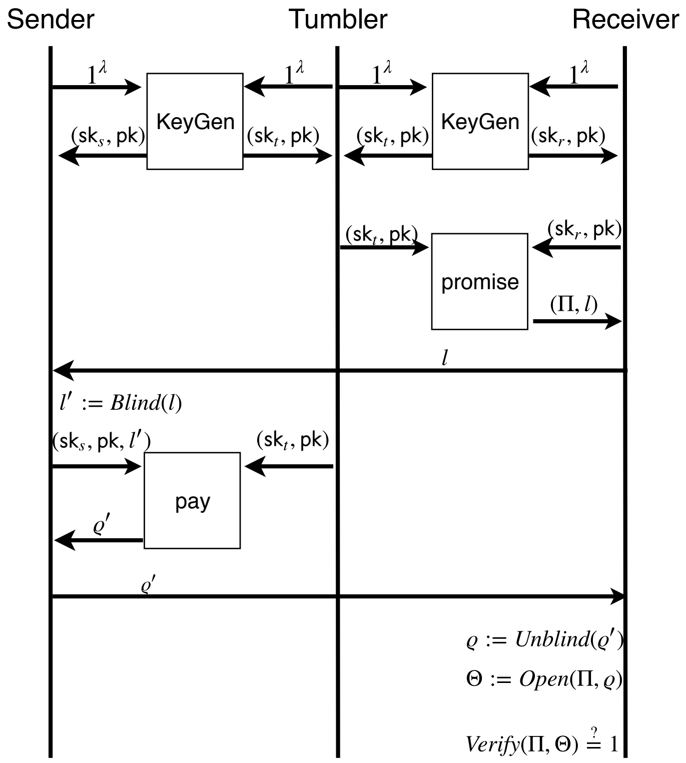

Sender
Tumbler
Receiver
1^λ
1^λ
1^λ
1^λ
KeyGen
KeyGen
$(\mathsf{sk}_s, \mathsf{pk})$
$(\mathsf{sk}_t, \mathsf{pk})$
$(\mathsf{sk}_t, \mathsf{pk})$
$(\mathsf{sk}_r, \mathsf{pk})$
$(\mathsf{sk}_t, \mathsf{pk})$
$(\mathsf{sk}_r, \mathsf{pk})$
promise
(Π, l)
l
$l' := Blind(l)$
$(\mathsf{sk}_s, \mathsf{pk}, l')$
$(\mathsf{sk}_t, \mathsf{pk})$
pay
ϱ'
ϱ'
$\varrho := Unblind(\varrho')$
$\Theta := Open(\Pi, \varrho)$
$Verify(\Pi, \Theta) \overset{?}{=} 1$

Fig. 1: Example of usage of the API provided by A^{2}L.

the tumbler trivially learns the link between the sender and the receiver. Thus, the sender randomizes ℓ into ℓ' before engaging into the pay protocol. The pay protocol ensures that the tumbler gets a payment from the sender only if the tumbler reveals ϱ' to the sender, which encodes the (blinded) solution to the cryptographic challenge encoded in ℓ. This invariant is crucial for atomicity, as the tumbler could otherwise get the coins from the sender and release an invalid solution.

Finally, the sender sends the randomized solution ϱ' to the receiver. Upon reception, the receiver unblinds ϱ', extracts the promise fullfillment Θ and uses it to finalize Π, that is, the initially committed payment promise from the tumbler (i.e., the receiver used Θ to get the money from the tumbler). We remark that here we use blind and unblind operations to highlight the key ideas about how privacy is preserved, but in the definition and instantiation of A^{2}L, we let the blind and unblind operations be internally done by the Promise and Pay protocols, since unlinkability is an inherent goal of A^{2}L.

Formal definition. Formally, A^{2}L is defined with respect to an intermediary P_t (the tumbler) and two parties P_s and P_r (the sender and the receiver) from a universe $\mathbb{P}$.

Definition 1 (Anonymous Atomic Lock (A^{2}L)). *An A^{2}L $\mathbb{L} =$ (KGen, Promise, Pay, Open, Verify) consists of the following protocols (for an intermediary P_t and two parties $P_s, P_r \in \mathbb{P}$):*

- $\{(\mathsf{sk}_t, \mathsf{pk}_{i,t}), (\mathsf{sk}_i, \mathsf{pk}_{i,t})\} \leftarrow \langle \mathsf{KGen}_{P_t}(1^\lambda), \mathsf{KGen}_{P_i}(1^\lambda)\rangle$*: On input the security parameter 1^λ, the key generation protocol returns a shared public key $\mathsf{pk}_{i,t}$ and a secret key sk_t (sk_i, respectively) to P_t (resp. P_i).*
- $\{\cdot, (\Pi, \ell)\} \leftarrow \langle \mathsf{Promise}_{P_t}(\mathsf{sk}_t, \mathsf{pk}_{r,t}), \mathsf{Promise}_{P_r}(\mathsf{sk}_r, \mathsf{pk}_{r,t})\rangle$*:*

On input two secret keys $\mathsf{sk}_t, \mathsf{sk}_r$, and a public key $\mathsf{pk}_{r,t}$, the promise protocol is executed between two parties (namely, P_t and P_r), and it returns a promise Π and a lock ℓ to P_r.

- $\{\varrho, \cdot\} \leftarrow \langle \mathsf{Pay}_{P_s}(\mathsf{sk}_s, \mathsf{pk}_{s,t}, \ell), \mathsf{Pay}_{P_t}(\mathsf{sk}_t, \mathsf{pk}_{s,t})\rangle$*: On input two secret keys sk_s and sk_t, a public key $\mathsf{pk}_{s,t}$, and a lock ℓ, the payment protocol is executed between two parties (namely, P_s and P_t) and it returns the solution ϱ of the lock ℓ to P_s.*
- $\Theta \leftarrow \mathsf{Open}(\Pi, \varrho)$*: On input a promise Π and a solution ϱ, the opening algorithm returns the promise fullfillment Θ.*
- $\{0,1\} \leftarrow \mathsf{Verify}(\Pi, \Theta)$*: On input a promise Π and a promise fullfillment Θ, the verification algorithm returns a bit $b \in \{0,1\}$.*

Correctness. Intuitively, A^{2}L is correct if the receiver gets the money paid by the sender through the tumbler with overwhelming probability. For a more detailed and formal correctness definition, we refer the reader to Appendix A.

III. Security Model

In this section, we formalize the security and privacy notions of interest for A^{2}L in the universal composability framework [7] in order to account for concurrent executions and allow thereby for the composition of A^{2}L with other blockchain protocols.

A. Attacker Model

We model parties as interactive Turing machines (ITMs), which communicate with a trusted functionality $\mathcal{F}$ via secure and authenticated communication channels. We model the adversary $\mathcal{A}$ as a PPT machine. The adversary can corrupt a party P through an interface $\mathsf{corrupt}(\cdot)$ that takes as input a party identifier P_i and provides the attacker with the internal state of P. Furthermore, all subsequent incoming and outgoing communication of P is routed through $\mathcal{A}$. As commonly done in the literature [15], [23], [35], [36], we consider the static corruption model, that is, the adversary commits to the identifiers of the users he wishes to corrupt ahead of time.

B. Ideal Functionality

We formalize below the ideal functionality $\mathcal{F}_{\mathsf{A^2L}}$ of our anonymous atomic lock construction.

Communication Model. Communication happens through the secure transmission functionality $\mathcal{F}_{\mathsf{smt}}$, as defined in [7], which informs the adversary whenever a communication between any two parties happens, and allows the adversary to delay the delivery of the messages arbitrarily. However, the adversary cannot read nor change the content of the messages.

Additionally, we assume existence of an anonymous communication channel as defined in [6], which we denote here as $\mathcal{F}_{\mathsf{anon}}$. In the ideal functionality we use the interfaces $\mathsf{send}_{\mathsf{smt}}$ and $\mathsf{receive}_{\mathsf{smt}}$ to exchange messages through the $\mathcal{F}_{\mathsf{smt}}$ functionality, and interfaces $\mathsf{send}_{\mathsf{anon}}$ and $\mathsf{receive}_{\mathsf{anon}}$ to exchange messages via $\mathcal{F}_{\mathsf{anon}}$.

We consider a synchronous communication network, where communication proceeds in discrete rounds, as defined in [26] and denoted here as $\mathcal{F}_{\mathsf{syn}}$. The parties are always aware of the current round, and if a party P sends a message in round r, the recipient party receives the message in the beginning of round $r+1$. The adversary can change the order of messages, but we assume that the order of messages between honest parties cannot be changed (which can easily be realized using message counters). For simplicity, we assume that computation is instantaneous.

Our Model. As previously described, we use $\mathcal{F}_{\mathsf{anon}}$, $\mathcal{F}_{\mathsf{smt}}$ and $\mathcal{F}_{\mathsf{syn}}$, thus, our functionality is defined in the $(\mathcal{F}_{\mathsf{anon}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{syn}})$-hybrid model.

$\mathcal{F}_{\mathsf{A^2L}}$ works in interaction with a universe of parties $\mathbb{P}$. However, since we model interactions among three parties, we use individual symbols for readability, namely P_s (sender), P_r (receiver) and P_t (tumbler). Additionally, $\mathcal{F}_{\mathsf{A^2L}}$ manages a list $\mathcal{P}$ (initially set to $\mathcal{P} := \emptyset$), to keep track of each promise and its corresponding promise fullfillment. The entries in the list have the format $(\Pi, \ell, \Theta, \varrho, P_i)$, where Π is a promise, ℓ is a lock, Θ is a promise fullfillment, ϱ is the solution for the lock, and P_i is the party involved in the promise with the tumbler (intermediary) P_t. Additionally, for clarity of exposition, we denote by $\mathtt{rand}(\cdot)$ and $\mathtt{derand}(\cdot)$ the randomization and the corresponding de-randomization functions, which given as input a (possibly randomized) value, return the (de-)randomized version of it. These functions are used inside the Promise and Pay interfaces as defined in Figure 2.

$\mathcal{F}_{\mathsf{A^2L}}$ provides five interfaces, which are depicted in Figure 2. The KGen interface allows the tumbler and the other party to establish a link between themselves. The Promise interface allows a party to obtain a promise and a lock from the tumbler. The Pay interface allows a party to acquire the solution of a given lock. The Open interface allows a party to fulfill a promise. Finally, the Verify interface verifies that the promise and the promise fullfillment match each other.

1) Alternative Approaches: Naturally, there exist alternative approaches to model anonymous atomic locks. A first possibility would be to define two separate ideal functionalities, one for the promise phase and the other one for the payment phase, similar to the model in [22]. However, the ideal functionalities in [22] by themselves only satisfy a property that they call fairness, which is analogous to our atomicity notion (defined in Section III-C). In order to achieve any meaningful privacy notion they have to assume that the puzzle input given to their payment functionality is blinded beforehand. In contrast, our ideal functionality achieves unlinkability by design, as a lock gets randomized inside the ideal functionality itself (see Section III-C for more details).

Another possible approach is to construct a single 2-of-2 signature ideal functionality, and then instantiate it with different signatures that satisfy the desired properties. However, it is harder to fit this approach into our setting, as our primary property is unlinkability, hence, we need a way to correlate the signature from the promise protocol with the signature from the payment protocol in an unlinkable fashion. It is not obvious how to do this with a 2-of-2 signature functionality.

```
KGen(sid)
Upon invocation by $P_i$, where $P_i \in \{P_s, P_r\}$:
send_smt $(\mathsf{sid}, P_i)$ to $P_t$
receive_smt $(\mathsf{sid}, b)$ from $P_t$
if $b = \bot$ then send $(\mathsf{sid}, \bot)$ to $P_i$ and abort
else send_smt $(\mathsf{sid}, P_i, P_t)$ to $P_i$

Open(sid, $\Pi, \varrho'$)
Upon invocation by $P_r$:
if $\Pi = \bot$ or $\varrho' = \bot$ then abort
set $\varrho \leftarrow \mathsf{derand}(\varrho')$
if $\exists(\Pi^*, -, \Theta^*, \varrho^*, P_i^*) \in \mathcal{P}$ such that $\Pi^* = \Pi$ and $\varrho^* = \varrho$ and $P_i^* = P_r$, then send $(\mathsf{sid}, \Theta^*)$ to $P_r$
else send $(sid, \bot)$ to $P_r$ and abort

Promise(sid, $P_s$)
Upon invocation by $P_r$:
send_smt $(\mathsf{sid}, P_r)$ to $P_t$
receive_smt $(\mathsf{sid}, \Pi, \ell, \varrho, \Theta)$ from $P_t$
if $\Pi = \bot$ or $\ell = \bot$ or $\varrho = \bot$ or $\Theta = \bot$ then abort
insert $(\Pi, \ell, \Theta, \varrho, P_r)$ into $\mathcal{P}$
send_smt $(\mathsf{sid}, \Pi, \ell)$ to $P_r$
send_anon $(\mathsf{sid}, \ell, P_r)$ to $P_s$

Pay(sid, $\ell, P_r$)
Upon invocation by $P_s$:
if $\ell = \bot$ then abort
set $\ell' \leftarrow \mathsf{rand}(\ell)$
send_smt $(\mathsf{sid}, \ell')$ to $P_t$
receive_smt $(\mathsf{sid}, \varrho')$ from $P_t$
if $\varrho' = \bot$ or $\nexists(-, \ell^*, -, \varrho^*, P^*) \in \mathcal{P}$ such that $\ell^* = \ell$ and $\varrho^* = \mathsf{derand}(\varrho')$ and $P^* = P_r$ then send_smt $(\mathsf{sid}, \bot)$ to $P_s$ and abort
else send_smt $(\mathsf{sid}, \varrho')$ to $P_s$
and send_anon $(\mathsf{sid}, \varrho')$ to $P_r$

Verify(sid, $\Pi, \Theta$)
Upon invocation by $P_r$:
if $\Pi = \bot$ or $\Theta = \bot$ then abort
if $\exists(\Pi^*, -, \Theta^*, \varrho^*, P_i^*) \in \mathcal{P}$ such that $\Pi^* = \Pi$ and $\Theta^* = \Theta$ and $P_i^* = P_r$, then send $(\mathsf{sid}, \top)$ to $P_r$
else send $(\mathsf{sid}, \bot)$ to $P_r$
```

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

C. Discussion

We define the security and privacy notions of interest for our $\mathcal{F}_{\mathsf{A^2L}}$ functionality.

Atomicity. Loosely speaking, the system should ensure that a lock can only be opened if there has been a payment for it before. This protects the tumbler from a malicious receiver. This is enforced by $\mathcal{F}_{\mathsf{A^2L}}$ because it keeps track of each promise along with the solution of the lock and the promise fullfillment. $\mathcal{F}_{\mathsf{A^2L}}$ checks whether the solution given to the Open interface corresponds to one of the existing entries in the list $\mathcal{P}$. Since, a party obtains a solution only from a call to the Pay interface and $\mathcal{F}_{\mathsf{A^2L}}$ is trusted, this ensures that Pay has to be instantiated before Open in order for Open to succeed.

Additionally, the system should ensure that if a payment can be received by the tumbler then the receiver can fulfill a matching promise previously issued by the tumbler. This protects the sender from a malicious tumbler. Assume that the Pay interface is invoked on a lock ℓ previously issued by a Promise. If $\mathcal{F}_{\mathsf{A^2L}}$ does not abort, then $\mathcal{F}_{\mathsf{A^2L}}$ ensures that it returns the correct solution ϱ matching the promise Π. In other words, if Open is invoked on input Π and ϱ, $\mathcal{F}_{\mathsf{A^2L}}$ ensures the existence of an entry in $\mathcal{P}$ containing both.

Unlinkability. Intuitively, unlinkability means that the tumbler does not learn information that allows it to associate the sender and the receiver of a payment. This property is enforced by $\mathcal{F}_{\mathsf{A^2L}}$ since the lock ℓ that is created by the tumbler in the Promise interface gets randomized by $\mathcal{F}_{\mathsf{A^2L}}$ within the Pay interface before it is sent back to the tumbler.

Additionally, since we assume existence of anonymous communication channel between parties (i.e., the $\mathcal{F}_{\mathsf{anon}}$ functionality), the intermediary cannot use the network information to correlate between sender and receiver. We remark that this assumption is indispensable for unlinkability and is commonly adopted in the PCH-related literature [21], [22].

Ideal functionality for PCH. In Appendix C we show how to define a fully-fledged PCH ideal functionality based on $\mathcal{F}_{\mathsf{A^2L}}$. This can be realized by interfacing $\mathcal{F}_{\mathsf{A^2L}}$ with the already existing ideal functionality for blockchains [15] and the logic for payments [35], which in turn amounts to the management of balances and timeouts.

D. Universal Composability

We now review the notion of secure realization in the UC framework [7]. Intuitively, a protocol realizes an ideal functionality if the adversary has no way to distinguish between the ideal functionality and the real-world protocol, where a simulator is in charge of translating the messages produced by the ideal functionality for the computational adversary. Here $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{E}}$ denotes the ensemble of the outputs of the environment $\mathcal{E}$ when interacting with the adversary $\mathcal{A}$ and users running protocol π.

Definition 2 (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.*

IV. Our Protocols

In this section, we present our A^{2}L instantiations. In particular, we give an overall intuition in Section IV-A, we discuss the building blocks in Section IV-B, we detail the Schnorr-based instantiation in Section IV-C and the ECDSA-based instantiation in Section IV-D. Due to the lack of space, we defer our Monero-based instantiation to Appendix E.

A. Intuition

We have divided our construction into two main protocols, *promise* and *payment*. The promise protocol is executed between Tumbler and Bob to create a promise (e.g., a transaction that sends coins from Tumbler to Bob) and a two-party signature for such promise that is "almost valid" meaning that Bob can finish it only if he gets to know a value α. Additionally, Tumbler sends Bob the value α in a ciphertext encrypted with Tumbler's public key. It is important to note that at this point, Bob cannot yet complete the signature as he can neither forge the signature nor he can decrypt the ciphertext because he does not know the Tumbler's decryption key. Instead, Bob re-randomizes the ciphertext (and hence the encrypted value), and sends it to Alice.

This is where the payment protocol comes into play, which is executed between Alice and Tumbler. Before the start of the payment protocol, Alice also randomizes the ciphertext on her side and sends this to Tumbler. If we do not also randomize at Alice's side, then Tumbler colluding with Bob can learn the true identity of Alice. This attack simply requires Bob revealing his randomized data to Tumbler. We note that this attack only makes sense in a scenario where Alice wants to pay without revealing her true identity (e.g., if Alice is a Tor user).

Once Tumbler receives the re-randomized ciphertext, it decrypts the ciphertext to obtain the doubly randomized version of the value α (i.e., the value required by Bob to compute the remaining part of the signature of the promise transaction).

In a nutshell, Alice then uses the payment protocol to buy the aforementioned randomized secret value from the Tumbler. In a bit more detail, Tumbler and Alice create a new message (e.g., a transaction that sends coins from Alice to Tumbler) and compute a two-party signature protocol modified in such a manner that Tumbler can obtain the signature (and thus the coins) only if it reveals the randomized secret value to Alice. After this protocol is finished, Alice can remove her part of the randomness from the secret, and send it to Bob, who can also remove his part of the randomness, getting thereby the value α and completing the signature for the promise transaction.

B. 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. We review here the cryptographic primitives used in our protocols.

Commitment Scheme. A commitment scheme COM consists of a commitment algorithm $(\mathsf{com}, \mathsf{decom}) \leftarrow \mathsf{Commit}(m)$, and a verification algorithm $\{0,1\} \leftarrow \mathsf{V}_{\mathsf{COM}}(\mathsf{com}, \mathsf{decom}, m)$. The commitment algorithm allows a prover to commit to a message m without revealing it. The verification algorithm allows a prover to convince a verifier by confirming that the message m was committed previously by revealing the decommitment information decom. The security of a COM scheme is modeled by the ideal functionality $\mathcal{F}_{\mathsf{COM}}$ [7].

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., $L = \{x \mid \exists w \text{ s.t. } R(x,w) = 1\}$). A non-interactive zero-knowledge proof scheme NIZK [4] consists of a prover algorithm $\pi \leftarrow \mathsf{P}_{\mathsf{NIZK}}(x,w)$ and a verification algorithm $\{0,1\} \leftarrow \mathsf{V}_{\mathsf{NIZK}}(x,\pi)$. A NIZK scheme allows a prover to convince a verifier about the existence of a witness w for a statement x without revealing any information apart from the fact that it actually knows the witness w. We can model the security of a NIZK scheme using the following simple ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$: on input (sid, x, w) by the prover, check if $R(x,w) = 1$, and if this is the case send $(\mathsf{sid}, \mathsf{proof}, x)$ to the verifier.

Homomorphic Encryption. An additive homomorphic encryption scheme HE is composed of the algorithms $(\mathsf{KGen}_{\mathsf{HE}}, \mathsf{Enc}_{\mathsf{HE}}, \mathsf{Dec}_{\mathsf{HE}})$, where $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}_{\mathsf{HE}}()$, $c \leftarrow \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, m)$, and $m \leftarrow \mathsf{Dec}_{\mathsf{HE}}(\mathsf{sk}, c)$. In our construction, we rely on Paillier homomorphic encryption scheme [41]. It supports homomorphic operations over the ciphertexts of the form $\mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, m_1) \cdot \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, m_2) = \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, m_1 + m_2)$ and $\mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, m_1)^{m_2} = \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, m_1 \cdot m_2)$. For our Schnorr-based construction we use the assumption that Paillier is IND-CPA, whereas for our ECDSA-based construction we have to assume that Paillier is ecCPA as done in [33]. The reason for this distinction is that for the ECDSA case we rely on Lindell's two-party ECDSA protocol (as explained in the next paragraph), and that protocol requires the ecCPA assumption.

ECDSA Signature. Let $\mathbb{G}$ be an elliptic curve group of order q with a base point g, and let $H : \{0,1\}^* \to \mathbb{Z}_q$ be a collision resistant hash function. The ECDSA signature is composed of the algorithms $(\mathsf{KGen}_{\mathsf{ECDSA}}, \mathsf{Sig}_{\mathsf{ECDSA}}, \mathsf{Vf}_{\mathsf{ECDSA}})$, and is defined as follows (using the multiplicative notation): $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}_{\mathsf{ECDSA}}()$ samples a private key $\mathsf{sk} = x$ and computes the corresponding public key as $\mathsf{pk} = Q = g^x$. The signing algorithm $(r, s) \leftarrow \mathsf{Sig}_{\mathsf{ECDSA}}(\mathsf{sk}, m)$ samples a random $k \leftarrow_{\$} \mathbb{Z}_q$ and computes $e = H(m)$. Let $(r_x, r_y) := R \leftarrow g^k$, then the signing algorithm computes the signature as $r \leftarrow r_x \bmod q$ and $s \leftarrow k^{-1}(e + rx) \bmod q$. Lindell [33] proposed an interactive and efficient two-party protocol $\Pi^{\mathsf{ECDSA}}_{\mathsf{KGen}}$, which performs distributed key generation for ECDSA. One party receives (x_1, Q, sk), where sk is the Paillier secret key and $Q = g^{x_1 \cdot x_2}$. The other party receives $(x_2, Q, \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, x_1))$, where pk is the corresponding Paillier public key. An ideal functionality $\mathcal{F}^{\mathsf{ECDSA}}_{\mathsf{KGen}}$ that securely computes the tuples for both parties is given in Appendix B. After the distributed key generation is performed, the parties can go on to perform the distributed ECDSA signing, which is again detailed in Lindell's work [33]. A comparison of different threshold ECDSA schemes can be seen in Section VII.

Schnorr Signature. Let $\mathbb{G}$ be a group of prime order q with a generator g, and let $H : \{0,1\}^* \to \mathbb{Z}_q$ be a collision resistant hash function. The Schnorr signature is defined using the algorithms $(\mathsf{KGen}_{\mathsf{Schnorr}}, \mathsf{Sig}_{\mathsf{Schnorr}}, \mathsf{Vf}_{\mathsf{Schnorr}})$ as follows: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}_{\mathsf{Schnorr}}()$ samples a private key $\mathsf{sk} = x$ and computes the corresponding public key as $\mathsf{pk} = Q = g^x$. The signing algorithm $(e, s) \leftarrow \mathsf{Sig}_{\mathsf{Schnorr}}(\mathsf{sk}, m)$, samples a random $k \leftarrow_{\$} \mathbb{Z}_q$ and computes $e \leftarrow H(R \| Q \| m)$, where $R \leftarrow g^k$, and $s \leftarrow k - x \cdot e \bmod q$. Unlike ECDSA, Schnorr has a linear structure, hence, it is easier to produce a two-party protocol $\Pi^{\mathsf{Schnorr}}_{\mathsf{KGen}}$, which performs distributed key generation. One party receives (x_1, Q) and the other party receives (x_2, Q), where $Q = g^{x_1 + x_2}$. An ideal functionality $\mathcal{F}^{\mathsf{Schnorr}}_{\mathsf{KGen}}$ that securely computes the tuples for both parties is given in Appendix B. Due to its linear structure, it is obvious to see that one can also perform the distributed signing using the Schnorr signature.

C. Schnorr-based Construction

Let $\mathbb{G}$ be a 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 HE be a commitment scheme, a non-interactive zero-knowledge scheme, and a Paillier homomorphic encryption scheme, respectively, as defined in Section IV-B. The Schnorr-based promise and payment protocols are shown in Figure 3 and 4, respectively.

Each pair of parties (P_1, P_2) generates a shared Schnorr public key $\mathsf{pk} = g^{x_1+x_2}$ via the $\mathcal{F}_{\mathsf{KGen}}^{\mathsf{Schnorr}}$ ideal functionality, where we assume that $P_2 =$ Tumbler in both protocols, and $P_1 =$ Bob in the promise protocol whereas $P_1 =$ Alice in the payment protocol. The Schnorr-based distributed key generation functionality $\mathcal{F}_{\mathsf{KGen}}^{\mathsf{Schnorr}}$ is described in Appendix B.

The promise protocol is run between two parties (P_1, P_2) (Bob and Tumbler, respectively). They initially agree on a message which corresponds to a transaction that is supposed to transfer coins from Tumbler to Bob. Additionally, Tumbler chooses a secret value α, encrypts it under its own public key using Paillier homomorphic encryption, and sends the ciphertext to Bob. The parties then execute a coin tossing protocol to agree on a randomness $R' = k'_1 + k'_2 + \alpha$, where α is unknown to Bob. 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 Tumbler embedding α in the computed randomness. The computation of R' together with the corresponding consistency proof is piggybacked in the coin tossing. At this point, Tumbler computes its side of the two-party Schnorr signature, but does not include the secret α into the signature. Now, Bob is able to validate this partial signature that he receives from Tumbler, and also to compute an "almost valid" signature by performing his part of the two-party signature. This means that Bob computes a tuple $(e', s' := k'_1 + k'_2 - e' \cdot (x'_1 + x'_2))$, and that the complete signature is of the form $(e', s' + \alpha)$. However, Bob does not have α, so he cannot complete the signature. Nevertheless, Bob receives $c_a = \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_T, \alpha)$ and $A = g^\alpha$ from Tumbler at the beginning of the promise protocol, and at the end of the promise protocol Bob chooses a random value β, and re-randomizes the values as $c_{a'} = c_a \cdot \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_T, \beta) = \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_T, \alpha + \beta)$ and $A' = A \cdot g^\beta = g^{\alpha+\beta}$ using β. This is possible due to the homomorphic properties of Paillier. The promise protocol finishes with Bob sending these re-randomized values to Alice.

The payment protocol is executed between two parties (P_1, P_2) (Alice and Tumbler, respectively). At the beginning of the protocol, Alice chooses a random value τ, and re-randomizes the values she received from Bob, as $c_{a''} = c_{a'} \cdot \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_T, \tau) = \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_T, \alpha + \beta + \tau)$ and $A'' = A' \cdot g^\tau = g^{\alpha+\beta+\tau}$. Once this is done, Alice and Tumbler perform a coin tossing protocol similar to the one performed between Bob and Tumbler in the promise protocol, but additionally Alice sends $c_{a''}$ to Tumbler. At this point, Tumbler decrypts $c_{a''}$ to obtain the value $\gamma = \alpha + \beta + \tau$. The rest of the protocol continues similar to the promise protocol, where Tumbler and Alice compute a common randomness, and then perform a two-party Schnorr signature. This time, however, Tumbler incorporates the decrypted value γ as part of the randomness. After the two-party Schnorr signature completes and Tumbler publishes it (allowing Tumbler to receive the payment from Alice), Alice is able to extract the γ from the published signature. She removes her part of the re-randomization from γ as $\bar{\alpha} = \gamma - \tau$, and sends this value to Bob, who can also remove his side of the re-randomization and obtain the initial $\alpha = \bar{\alpha} - \beta$. Once Bob obtains α, he can use it to complete the "almost" signature that he computed at the end of the promise protocol, which allows him to claim the coins that were promised to him by Tumbler.

Security Analysis. The security of the Schnorr-based construction is established by the following theorem, which we formally prove in Appendix B.

Theorem 1. *Let* COM *be a secure commitment scheme and let* NIZK *be a non-interactive zero-knowledge scheme. If Schnorr signature is strongly existentially unforgeable and Paillier encryption is* IND-CPA *secure, then the construction in Figures 3, 4 and 5, UC-realizes the ideal functionality* $\mathcal{F}_{\mathsf{A^2L}}$ *in the* $(\mathcal{F}_{\mathsf{KGen}}^{\mathsf{Schnorr}}, \mathcal{F}_{\mathsf{anon}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{syn}})$*-hybrid model.*

D. ECDSA-based Construction

While the Schnorr-based construction can exploit the linear structure that the signature offers, this linearity is not present in ECDSA, which makes the design of our protocol more challenging.

Let $\mathbb{G}$ be an elliptic curve group of order q with a base point g, and let $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$ be a collision resistant hash function. Additionally, let $\mathsf{COM}, \mathsf{NIZK}$, and HE be a commitment scheme, a non-interactive zero-knowledge scheme, and a Paillier homomorphic encryption scheme, respectively, as defined in Section IV-B. The ECDSA-based promise and payment protocols are shown in Figure 6 and 7, respectively.

Our ECDSA-based instantiation shares similar ideas with our Schnorr-based instantiation. Hence, we only describe the differences compared to the Schnorr variant here. Each pair of parties (P_1, P_2) generates a shared ECDSA public key $\mathsf{pk} = g^{x_1 \cdot x_2}$ via the $\mathcal{F}_{\mathsf{KGen}}^{\mathsf{ECDSA}}$ ideal functionality, where, as before, $P_2 =$ Tumbler in both protocols, whereas $P_1 =$ Bob in the promise protocol and $P_1 =$ Alice in the payment protocol. Because ECDSA does not have the linear structure of Schnorr, the distributed key generation is also more complicated, and it requires additionally exchanging a Paillier encrypted secret key. More precisely, P_1 receives a Paillier secret key sk and its share x_1, whereas P_2 receives its share x_2 and the Paillier encryption c of x_1. The ECDSA-based distributed key generation functionality $\mathcal{F}_{\mathsf{KGen}}^{\mathsf{ECDSA}}$ is described in the full version [18].

The promise protocol runs similarly to the Schnorr-based promise protocol, expect 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 Bob. Once the randomness is computed, Tumbler performs its side of the two-party ECDSA signature using c'_{key} (the encryption of x'_1) and the homomorphic properties of Paillier. However, Tumbler does not include the inverse of α into the signature. Now, Bob is able to compute an "almost valid" signature by decrypting the ciphertext that it received from Tumbler and performing his part of the signature. This means that Bob 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 $(r', s' \cdot \alpha^{-1})$.

Public parameters: $\mathbb{G}, g, q$, message m', public key $Q' := g^{x'_1 + x'_2}$, and proof π_T that proves validity of (n, g)

$\mathsf{Promise}_T(\mathsf{sk}_T := x'_2, \mathsf{pk} := Q')$		$\mathsf{Promise}_B(\mathsf{sk}_B := x'_1, \mathsf{pk} := Q')$
$\alpha, k'_2 \leftarrow_{\$} \mathbb{Z}_q$ $c_a \leftarrow \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_T, \alpha); A \leftarrow g^{\alpha}$ $\pi_a \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists \alpha \mid A = g^{\alpha}\}, \alpha)$ $R'_2 \leftarrow g^{k'_2}; \pi'_2 \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists k'_2 \mid R'_2 = g^{k'_2}\}, k'_2)$ $(\mathsf{com}, \mathsf{decom}) \leftarrow \mathsf{P}_{\mathsf{COM}}((R'_2, \pi'_2))$		
	$\xrightarrow{\mathsf{com}, A, \pi_a, c_a}$	
		If $\mathsf{V}_{\mathsf{NIZK}}(\pi_a, A) \neq 1$ then abort $k'_1 \leftarrow_{\$} \mathbb{Z}_q; R'_1 \leftarrow g^{k'_1}$ $\pi'_1 \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists k'_1 \mid R'_1 = g^{k'_1}\}, k'_1)$
	$\xleftarrow{R'_1, \pi'_1}$	
If $\mathsf{V}_{\mathsf{NIZK}}(\pi'_1, R'_1) \neq 1$ then abort $R' \leftarrow R'_1 \cdot R'_2 \cdot A; e' \leftarrow H(R' \| Q' \| m')$ $s'_2 \leftarrow k'_2 - x'_2 \cdot e' \mod q$		
	$\xrightarrow{(\mathsf{decom}, R'_2, \pi'_2), s'_2}$	
		If $\mathsf{V}_{\mathsf{COM}}(\mathsf{com}, \mathsf{decom}, (R'_2, \pi'_2)) \neq 1$ then abort If $\mathsf{V}_{\mathsf{NIZK}}(\pi'_2, R'_2) \neq 1$ then abort $R' \leftarrow R'_1 \cdot R'_2 \cdot A; e' \leftarrow H(R' \| Q' \| m')$ If $g^{s'_2} \neq R'_2 \cdot (Q'/g^{x'_1})^{-e'}$ then abort $s'_1 \leftarrow k'_1 - x'_1 \cdot e' \mod q$ $s' \leftarrow s'_1 + s'_2 \mod q$ $\beta \leftarrow_{\$} \mathbb{Z}_q; A' \leftarrow A \cdot g^{\beta}$ $c_{a'} \leftarrow c_a \cdot \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_T, \beta)$
	$\xleftarrow{s'}$	
If $g^{s'} \neq R'_1 \cdot R'_2 \cdot Q^{-e'}$ then abort **return** $\sigma := (R', s' + \alpha)$		Send $\ell := (A', c_{a'})$ to Alice **return** $(\Pi := (\beta, (\mathsf{pk}, m', \sigma' := (R', s'))), \ell)$

Fig. 3: Promise protocol of Schnorr-based construction

Since Bob does not have α, he cannot complete the signature. However, similar to the Schnorr-based construction, Bob receives $c_a = \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_T, \alpha)$ and $A = g^{\alpha}$ from Tumbler at the beginning of the promise protocol, and at the end of the protocol Bob chooses a random value β and re-randomizes the values as $c_{a'} = c_a^{\beta}$ and $A' = A^{\beta}$ using β. The promise protocol finishes with Bob sending these re-randomized values to Alice.

At the beginning of the payment protocol, Alice chooses a random value τ and re-randomizes the values she received from Bob, as $c_{a''} = c_{a'}^{\tau}$ and $A'' = (A')^{\tau}$. The rest of the payment protocol continues similar to Schnorr-based payment protocol, though with Alice and Tumbler computing a two-party ECDSA signature. When Tumbler completes the signature and publishes it, Alice extracts the γ from the published signature. She removes her part of the re-randomization from γ as $\bar{\alpha} = \gamma \cdot (\tau)^{-1}$, and shares this value with Bob, who can also remove his side of the re-randomization and obtain the initial secret as $\alpha = \bar{\alpha} \cdot (\beta)^{-1}$. All that is left for Bob to claim the promised coins from Tumbler, is to invert α and use it to complete the "almost" signature that he computed at the end of the promise protocol.

Security Analysis. The security of the ECDSA-based construction is based on the ecCPA assumption and is established by Theorem 2, which we formally prove in Appendix B.

ecCPA Assumption and its Consequences. Our ECDSA-based construction relies on Lindell's two-party ECDSA protocol [33], whose security proof is based on the ecCPA assumption. Therefore, the security of our ECDSA-based construction is (partially) based on this assumption too. However, Lindell's original published work at CRYPTO 2017 included a minor flaw, and a revised version was later published on ePrint [33]. As a consequence of this revision, the oracle provided in the paper needs to stop working if there is an abort in the signing protocol. This implies that every time an abort occurs in the signing procedure between Tumbler and a party P, a new key between them need to be established to maintain security.

```
Public parameters: $\mathbb{G}, g, q$, message $m$, public key $Q := g^{x_1+x_2}$, and proof $\pi_T$ that proves validity of $(n, g)$

$\mathsf{Pay}_A(\mathsf{sk}_A := x_1, \mathsf{pk} := Q, \ell := (A', c_{a'}))$                      $\mathsf{Pay}_T(\mathsf{sk}_T := x_2, \mathsf{pk} := Q)$
                                                                      $k_2 \leftarrow_\$ \mathbb{Z}_q; R_2 \leftarrow g^{k_2}$
                                                                      $\pi_2 \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists k_2 \mid R_2 = g^{k_2}\}, k_2)$
                                                                      $(\mathsf{com}, \mathsf{decom}) \leftarrow \mathsf{P}_{\mathsf{COM}}((R_2, \pi_2))$
                                     $\xleftarrow{\mathsf{com}}$
$\tau, k_1 \leftarrow_\$ \mathbb{Z}_q; c_{a''} \leftarrow c_{a'} \cdot \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_T, \tau)$
$R_1 \leftarrow g^{k_1}; \pi_1 \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists k_1 \mid R_1 = g^{k_1}\}, k_1)$
                                     $\xrightarrow{c_{a''}, R_1, \pi_1}$
                                                                      If $\mathsf{V}_{\mathsf{NIZK}}(\pi_1, R_1) \neq 1$ then abort
                                                                      $\gamma \leftarrow \mathsf{Dec}_{\mathsf{HE}}(c_{a''}); A'' \leftarrow g^{\gamma}$
                                                                      $R \leftarrow R_1 \cdot R_2 \cdot A''; e \leftarrow H(R \| Q \| m)$
                                                                      $s_2 \leftarrow k_2 - x_2 \cdot e \bmod q$
                                     $\xleftarrow{(\mathsf{decom}, R_2, \pi_2), s_2, A''}$
If $\mathsf{V}_{\mathsf{COM}}(\mathsf{com}, \mathsf{decom}, (R_2, \pi_2)) \neq 1$ then abort
If $\mathsf{V}_{\mathsf{NIZK}}(\pi_2, R_2) \neq 1$ then abort
If $A' \cdot g^{\tau} \neq A''$ then abort
$R \leftarrow R_1 \cdot R_2 \cdot A''; e \leftarrow H(R \| Q \| m)$
If $g^{s_2} \neq R_2 \cdot (Q/g^{x_1})^{-e}$ then abort
$s_1 \leftarrow k_1 - x_1 \cdot e \bmod q$
$\bar{s} \leftarrow s_1 + s_2 \bmod q$
                                     $\xrightarrow{\bar{s}}$
                                                                      $s \leftarrow \bar{s} + \gamma$
                                                                      If verification of $(e, s)$ fails then abort
                                                                      Else publish signature $(e, s)$
$\gamma \leftarrow s - \bar{s}$
$\bar{\alpha} \leftarrow \gamma - \tau$
Send $\bar{\alpha}$ to Bob
return $\bar{\alpha}$                                                  return $\top$
```

Fig. 4: Payment protocol of Schnorr-based construction

```
$\mathsf{Open}(\Pi, \bar{\alpha})$
    Parse $\Pi$ as $(\beta, (\mathsf{pk}, m', \sigma' := (R', s')))$
    Set $\alpha \leftarrow \bar{\alpha} - \beta$
    Set $s \leftarrow s' + \alpha$
    return $(R', s)$
```

```
$\mathsf{Verify}(\Pi, \sigma)$
    Parse $\Pi$ as $(\beta, (\mathsf{pk}, m', \sigma'))$
    return $\mathsf{Verify}_{\mathsf{Schnorr}}(\mathsf{pk}, m', \sigma)$
```

Fig. 5: Open and verify algorithms of Schnorr-based construction.

Although this weakens the practicality of our ECDSA-based construction, we note that an abort only affects the parties involved in a channel where the abort occurred, and the rest of the parties in the system are not affected, hence, they do not require to re-generate the keys or their payment channels. Furthermore, such an abort can only happen when there is a misbehavior by one of the parties in a payment channel, and as such, the desired thing to do is to close the current channel with the gateway and possibly open a new one. A possible mitigation to this is to replace Lindell's two-party ECDSA protocol in our construction with one of the various existing threshold ECDSA protocols that support 2-of-2 signing, such as [34], [13], or [14]. We note that the drawback described here does not apply to our Schnorr-based construction as it uses the standard IND-CPA assumption for Paillier.

Theorem 2. *Let* COM *be a secure commitment scheme and let* NIZK *be a non-interactive zero-knowledge scheme. If ECDSA signature is strongly existentially unforgeable and Paillier encryption is* ecCPA *secure, then the construction in Figures 6, 7 and 8, UC-realizes the ideal functionality* $\mathcal{F}_{\mathsf{A^2L}}$ *in the* $(\mathcal{F}_{\mathsf{KGen}}^{\mathsf{ECDSA}}, \mathcal{F}_{\mathsf{anon}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{syn}})$*-hybrid model.*

Public parameters: $\mathbb{G}, g, q$, message m', public key $Q' := g^{x'_1 \cdot x'_2}$, and proof π_T that proves validity of (n, g)

$\mathsf{Promise}_T(\mathsf{sk}_T := x'_2, \mathsf{pk} := Q', c'_{key} = \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_B, x'_1))$		$\mathsf{Promise}_B(\mathsf{sk}_B := x'_1, \mathsf{pk} := Q')$
$\alpha, k'_2 \leftarrow_{\$} \mathbb{Z}_q; c_a \leftarrow \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_T, \alpha); A \leftarrow g^{\alpha}$ $\pi_a \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists \alpha \mid A = g^{\alpha}\}, \alpha)$ $R'_2 \leftarrow g^{k'_2}; \pi'_2 \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists k'_2 \mid R'_2 = g^{k'_2}\}, k'_2)$ $(\mathsf{com}, \mathsf{decom}) \leftarrow \mathsf{P}_{\mathsf{COM}}((R'_2, \pi'_2))$		
	$\xrightarrow{\mathsf{com}, A, \pi_a, c_a}$	
		If $\mathsf{V}_{\mathsf{NIZK}}(\pi_a, A) \neq 1$ then abort $k'_1 \leftarrow \mathbb{Z}_q$ $R'_1 \leftarrow g^{k'_1}; \pi'_1 \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists k'_1 \mid R'_1 = g^{k'_1}\}, k'_1)$
	$\xleftarrow{R'_1, \pi'_1}$	
If $\mathsf{V}_{\mathsf{NIZK}}(\pi'_1, R'_1) \neq 1$ then abort $R'_c \leftarrow (R'_2)^{\alpha}$ $\pi'_c \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists \alpha \mid R_c = (R'_2)^{\alpha}\}, \alpha)$ $\pi'_a \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists \alpha \mid A = g^{\alpha} \wedge R_c = (R'_2)^{\alpha}\}, \alpha)$ $R' \leftarrow (R'_1)^{k'_2 \cdot \alpha}; R' := (r'_x, r'_y)$; Set $r' \leftarrow r'_x \bmod q$ $m := (k'_2)^{-1} \cdot r' \cdot x'_1 \cdot x'_2 + (k'_2)^{-1} \cdot H(m') + \rho q$ $c' \leftarrow \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_B, m)$		
	$\xrightarrow{(\mathsf{decom}, R'_2, \pi'_2), c', R'_c, \pi'_c, \pi'_a}$	
		If $\mathsf{V}_{\mathsf{COM}}(\mathsf{com}, \mathsf{decom}, (R'_2, \pi'_2)) \neq 1$ then abort If $\mathsf{V}_{\mathsf{NIZK}}(\pi'_2, R'_2) \neq 1 \vee \mathsf{V}_{\mathsf{NIZK}}(\pi'_c, R'_c) \neq 1$ $\vee\, \mathsf{V}_{\mathsf{NIZK}}(\pi'_a, (A, R'_c)) \neq 1$ then abort $R' \leftarrow (R'_c)^{k'_1}; R' := (r'_x, r'_y)$ Set $r' \leftarrow r'_x \bmod q$ $s'_2 \leftarrow \mathsf{Dec}_{\mathsf{HE}}(\mathsf{sk}_B, c')$ If $(R'_2)^{s'_2 \bmod q} \neq (Q')^{r'} \cdot g^{H(m')}$ then abort $s' \leftarrow s'_2 \cdot (k'_1)^{-1} \bmod q; \beta \leftarrow_{\$} \mathbb{Z}_q$ $A' \leftarrow A^{\beta}; c_{a'} \leftarrow (c_a)^{\beta}$
	$\xleftarrow{s'}$	
If $(R'_1)^{k'_2 \cdot s'} \neq (Q')^{r'} \cdot g^{H(m')}$ then abort **return** $\sigma := (r', s' \cdot \alpha^{-1})$		Send $\ell := (A', c_{a'})$ to Alice **return** $(\Pi := (\beta, (\mathsf{pk}, m', \sigma' := (r', s'))), \ell)$

Fig. 6: Promise protocol of ECDSA-based construction

V. Performance Analysis

A. Implementation Details

We implemented our protocols and evaluated their performance. The implementation is done in C and it relies on the RELIC library [2] for the cryptographic operations. Both the ECDSA-based and Schnorr-based variants have been instantiated over the elliptic curve *secp256k1*, which is also used in Bitcoin. Paillier encryption is instantiated with a RSA group for 128-bit security level. Zero-knowledge proofs for discrete logarithm and Diffie-Hellman tuple have been implemented using Σ-protocols [12] and made non-interactive using the Fiat-Shamir heuristic [17]. Lastly, we have instantiated the commitment scheme using the hash function SHA-256.

We did not implement the distributed key generation and instead assigned random keys to every party. Key generation is usually carried out only once at setup time (e.g., opening a payment channel). We refer the interested reader to [33] and [19] for a detailed performance evaluation of distributed ECDSA and Schnorr key generation, respectively. In the following, we focus in the rest of operations as they are the ones defined for the first time in this work.

B. Evaluation

Testbed. We used three EC2 instances from Amazon AWS, where Tumbler was a m5a.2xlarge instance (2.50GHz AMD EPYC 7571 processor with 8 cores, 32GB RAM) located in

```
Public parameters: $\mathbb{G}, g, q$, message $m$, public key $Q := g^{x_1 \cdot x_2}$, and proof $\pi_T$ that proves validity of $(n, g)$

$\mathsf{Pay}_A(\mathsf{sk}_A := x_1, \mathsf{pk} := Q, \ell := (A', c_{a'}))$          $\mathsf{Pay}_T(\mathsf{sk}_T := x_2, \mathsf{pk} := Q, c_{key} = \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_A, x_1))$

                                                  $k_2 \leftarrow_\$ \mathbb{Z}_q; R_2 \leftarrow g^{k_2}$
                                                  $\pi_2 \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists k_2 \mid R_2 = g^{k_2}\}, k_2)$
                                                  $(\mathsf{com}, \mathsf{decom}) \leftarrow \mathsf{P}_{\mathsf{COM}}((R_2, \pi_2))$
                          $\xleftarrow{\mathsf{com}}$
$\tau, k_1 \leftarrow_\$ \mathbb{Z}_q; c_{a''} \leftarrow (c_{a'})^\tau; R_1 \leftarrow g^{k_1}$
$\pi_1 \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists k_1 \mid R_1 = g^{k_1}\}, k_1)$
                          $\xrightarrow{c_{a''}, R_1, \pi_1}$
                                                  If $\mathsf{V}_{\mathsf{NIZK}}(\pi_1, R_1) \neq 1$ then abort
                                                  $\gamma \leftarrow \mathsf{Dec}_{\mathsf{HE}}(\mathsf{sk}_T, c_{a''}); A'' \leftarrow g^\gamma; R_c \leftarrow (R_2)^\gamma$
                                                  $\pi_c \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists \gamma \mid R_c = (R_2)^\gamma\}, \gamma)$
                                                  $\pi_\gamma \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists \gamma \mid A'' = g^\gamma \wedge R_c = (R_2)^\gamma\}, \gamma)$
                                                  $R \leftarrow (R_1)^{k_2 \cdot \gamma}; R := (r_x, r_y);$ Set $r \leftarrow r_x \bmod q$
                                                  $m := (k_2)^{-1} \cdot r \cdot x_2 \cdot x_1 + (k_2)^{-1} \cdot H(m) + \rho q$
                                                  $c \leftarrow \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_A, m)$
                          $\xleftarrow{(\mathsf{decom}, R_2, \pi_2), c, A'', R_c, \pi_c, \pi_\gamma}$
If $\mathsf{V}_{\mathsf{COM}}(\mathsf{com}, \mathsf{decom}, (R_2, \pi_2)) \neq 1$ then abort
If $\mathsf{V}_{\mathsf{NIZK}}(\pi_c, R_c) \neq 1$ then abort
If $\mathsf{V}_{\mathsf{NIZK}}(\pi_\gamma, (A'', R_c)) \neq 1$ then abort
If $(A')^\tau \neq A''$ then abort
$R \leftarrow (R_c)^{k_1}; R := (r_x, r_y);$ Set $r \leftarrow r_x \bmod q$
$s_2 \leftarrow \mathsf{Dec}_{\mathsf{HE}}(\mathsf{sk}_A, c)$
If $(R_2)^{s_2 \bmod q} \neq Q^r \cdot g^{H(m)}$ then abort
$\bar{s} \leftarrow s_2 \cdot (k_1)^{-1} \bmod q$
                          $\xrightarrow{\bar{s}}$
                                                  $s \leftarrow (\gamma)^{-1} \cdot \bar{s}$
                                                  If verification of $(r, s)$ fails then abort
                                                  Else publish signature $(r, s)$
$\gamma \leftarrow (s \cdot (\bar{s})^{-1})^{-1}; \bar{\alpha} \leftarrow \gamma \cdot (\tau)^{-1}$
Send $\bar{\alpha}$ to Bob
return $\bar{\alpha}$                                  return $\top$
```

Fig. 7: Payment protocol of ECDSA-based construction

```
$\mathsf{Open}(\Pi, \bar{\alpha})$
Parse $\Pi$ as $(\beta, (\mathsf{pk}, m', \sigma' := (r', s')))$
Set $\alpha \leftarrow \bar{\alpha} \cdot \beta^{-1}$
Set $s \leftarrow s' \cdot \alpha^{-1}$
return $(r', s)$
```

```
$\mathsf{Verify}(\Pi, \sigma)$
Parse $\Pi$ as $(\beta, (\mathsf{pk}, m', \sigma'))$
return $\mathsf{Verify}_{\mathsf{ECDSA}}(\mathsf{pk}, m', \sigma)$
```

Fig. 8: Open and verify algorithms of ECDSA-based construction.

Frankfurt, whereas Alice and Bob were m5a.large instances (2.50GHz AMD EPYC 7571 processor with 2 cores, 8GB RAM) located in Singapore and Oregon, 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, and 32GB RAM. All the machines were running Ubuntu 18.04 LTS.

We measured the average runtimes over 100 runs each. The results of our performance evaluation are reported in Table II, where time is given in seconds.

TABLE II: Performance of ECDSA- and Schnorr-based construction. Time is shown in seconds.

	Payment Hub (Singapore-Frankfurt-Oregon)		LAN		Bandwidth	
	Schnorr	ECDSA	Schnorr	ECDSA	Schnorr	ECDSA
Promise	0.948	1.017	0.022	0.049	1.66KB	2.43KB
Payment	1.120	1.144	0.036	0.050	1.62KB	2.40KB
Open	1.136	1.139	0.010	0.010	0.16KB	0.16KB
Total	3.204	3.300	0.068	0.109	3.44KB	4.99KB

Computation Time. All our protocols complete in < 4 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 LAN setting. In that case our ECDSA-based construction finishes in ~ 110 milliseconds, whereas our Schnorr-based construction takes ~ 70 milliseconds. From these results we can observe that Schnorr-based construction is performing better than ECDSA-based construction. The reason for this is that ECDSA-based two-party signing has a more complex structure, requiring additional Paillier encryptions.

Next, we compare our constructions with the state-of-the-art payment hub TumbleBit [22]. In order to have more precise results, we performed the comparison in a LAN setting without any network latency. TumbleBit requires ~ 0.6 seconds to complete, hence, our ECDSA-based construction is more than 5x faster, whereas our Schnorr-based construction is more than 8x faster.

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. ECDSA-based construction has a higher communication overhead compared to the Schnorr-based construction. This is due to the fact that ECDSA-based two-party signing requires a Paillier ciphertext. Since we perform two-party ECDSA signing in both promise and payment protocols, this explains the additional bandwidth requirements of ECDSA-based construction.

TumbleBit requires 326KB of bandwidth, hence, our ECDSA-based and Schnorr-based constructions incur 65x and 95x less communication overhead, respectively.

In summary, we highlight two points. First, our constructions highly reduce both the communication and computation complexity compared to TumbleBit. Interestingly, while results in TumbleBit are shown for a security level of 80 bits, we run our experiments with a security parameter that provides 128 bits of security. Thus, our construction is more efficient even when providing a higher level of security.

Second, the reduction in computation and 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 and that implies that parties need to compute and exchange messages composed of $\binom{m+n}{m}$ elements, where m and n are the parameters for the cut-and-choose game. For instance, authors of TumbleBit used $m = 15$ and $n = 285$ to achieve 80 bits of security. Instead, A^{2}L requires to compute and exchange messages composed of constant number of elements.

VI. Payment Channel Hub Construction

We detail here how A^{2}L in combination with a blockchain $\mathcal{B}$ can be used to seamlessly realize a fully-fledged payment channel hub (PCH).

Assume that users have already carried out the key generation algorithm and set up the payment channels with Tumbler. Then, Alice can perform a payment to Bob through the Tumbler as follows.

First, Tumbler and Bob execute the Promise protocol and establish the following A^{2}L contract:

$\underline{A^2L\textbf{-Promise (Tumbler, Bob, }\Pi, x, t\textbf{):}}$

1) If Bob produces the promise fullfillment data Θ in such a manner that $\mathsf{Verify}(\Pi, \Theta) = 1$ before time t expires, Tumbler pays Bob x coins.
2) If timeout t expires, Tumbler gets back x coins.

Here, Π is the output (along with ℓ) of the Promise protocol in A^{2}L where the message is set as a transaction that sends x coins from Tumbler to Bob. t is an expiration time (validity period) of the promise, which is properly set to give Bob the time he needs to reveal the solution ϱ. In case this does not happen, then Tumbler 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 from either Schnorr or ECDSA digital signature scheme, and virtually all cryptocurrencies natively implement a time management system where time is measured as the number of blocks included in the blockchain.

Second, Bob sends the lock l (as output by the Promise protocol) to Alice. Then, Alice and Tumbler execute the Pay protocol and establish the following A^{2}L contract:

$\underline{A^2L\textbf{-Pay (Alice, Tumbler, }\ell, x\textbf{):}}$

1) If Tumbler sends Alice the solution ϱ to the cryptographic challenge encoded in ℓ, then Alice pays Tumbler x coins.
2) Otherwise, Alice gets back x coins.

Finally, Alice gets the solution ϱ to the cryptographic challenge encoded in the lock ℓ. Alice then sends ϱ to Bob who can then complete the A^2L-Promise contract with the promise fullfillment data $\Theta := \mathsf{Open}(\Pi, \varrho)$.

VII. Related Work

On-Chain Tumblers. Several prior works exist where a centralized tumbler assists users to mix their coins [1], [3], [5], [24], [37], [39], [44]–[47], [49], [50]. However, all these constructions heavily rely on on-chain transactions to operate, hindering thus the scalability of cryptocurrencies. A^2L instead is by definition operating with off-chain payments, aiding thus to the scalability of current blockchains. Moreover, while mentioned systems are restricted to one (or few) cryptocurrencies, A^2L rely only on widely deployed cryptographic primitives such as digital signatures schemes, paving the way to interoperable cross-chain applications.

Payment-Channel Networks. A scalability approach based on payment channels is payment-channel networks [42], where users performs payments through a path of opened channels between sender and receiver. Few research works have studied their security, privacy, and concurrency guarantees [35], [36]. Although interesting, we consider this research line orthogonal to our work. It is worth noting that Malavolta et al. [36] propose anonymous multi-hop locks (AMHL), a cryptographic construction to ensure the security and privacy of multi-hop locks also based on scriptless payments (i.e., payments where conditions are embedded in the signature itself). While interesting, this work is orthogonal to A^2L: A multi-hop payment inherenty requires to reveal the predecessor and successor nodes in the path to intermediaries, which is exactly the privacy notion in a PCH, where only one intermediary (tumbler) exists.

Threshold ECDSA Protocols. Subsequent to Lindell's work [33], Doerner et. al [13], [14] and Lindell et. al [34] provided a threshold variant of ECDSA signing, which can also be used in 2-of-2 signature setting that we require. However, [34] performs worse with respect to both communication and computation in 2-of-2 setting. On the other hand, although [13] and [14] perform better with respect to computation in 2-of-2 setting, they require more communication. Since we are in the Internet setting we want to minimize the communication, hence, they are not suitable for our scenario. Furthermore, since all these protocols target the threshold setting, they are more convoluted than Lindell's two-party ECDSA protocol. However, on a positive side, all these follow-up works remove the ad-hoc ecCPA assumption that Lindell's two-party ECDSA protocol introduced. Recently, Castagnos et. al. [10] proposed a generalized approach to Lindell's two-part ECDSA protocol [33] using hash proof systems, more precisely, homomorphically extended projective hash family. Their approach removes Lindell's ecCPA assumption, and additionally requires less communication compared to Lindell's protocol. Though, their computation requirements are higher. Nevertheless, this work is the closest direct replacement to Lindell's original two-party ECDSA protocol, hence, our approach works directly there without any modifications. This means that one can use [10] as a drop-in replacement to Lindell's two-party ECDSA in our ECDSA-based A^2L construction.

VIII. Conclusion

This paper presents A^2L, a new cryptographic primitive for realizing secure, privacy-preserving, interoperable, and fungibility-preserving PCHs. We develop two instantiations, based on ECDSA and Schnorr signatures, which makes our constructions compatible with the vast majority of today's cryptocurrencies. We defined and proved security and privacy for A^2L in the UC framework. We further demonstrated that A^2L is the most efficient Bitc oin-compatible PCH, showing that our ECDSA instantiation is 5x faster and requires 65x less bandwidth than the state-of-the-art TumbleBit protocol, even when providing a higher level of security.

As a future work, it would also be interesting to generalize our construction to multi-hop payment hubs and, ultimately, to interface PCHs with payment channel networks. Finally, we intend to explore techniques to achieve stronger value privacy guarantees and, possibly, the inherent trade-offs between interoperability and value privacy.

Acknowledgements

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 by FWF project W1255-N23.

References

[1] "CoinSwap: Transaction graph disjoint trustless trading," https://bitcointalk.org/index.php?topic=321228.0.

[2] D. F. Aranha and C. P. L. Gouvêa, "RELIC is an Efficient LIbrary for Cryptography," https://github.com/relic-toolkit/relic.

[3] G. Bissias, A. P. Ozisik, B. N. Levine, and M. Liberatore, "Sybil-Resistant Mixing for Bitcoin," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, ser. WPES '14. New York, NY, USA: ACM, 2014, pp. 149–158.

[4] M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge and its applications," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, ser. STOC '88. New York, NY, USA: ACM, 1988, pp. 103–112. [Online]. Available: http://doi.acm.org/10.1145/62212.62222

[5] 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*, ser. Lecture Notes in Computer Science, N. Christin and R. Safavi-Naini, Eds. Springer Berlin Heidelberg, 2014, pp. 486–504.

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

[7] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," 2000, https://eprint.iacr.org/2000/067.

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

[9] R. Canetti and T. Rabin, "Universal composition with joint state," in *Annual International Cryptology Conference*, 2003, pp. 265–281.

[10] 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*, A. Boldyreva and D. Micciancio, Eds. Cham: Springer International Publishing, 2019, pp. 191–221.

[11] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, D. Song, and R. Wattenhofer, "On scaling decentralized blockchains," in *Financial Cryptography and Data Security*, 2016.

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

[13] J. Doerner, Y. Kondi, E. Lee, and abhi shelat, "Secure multi-party threshold ecdsa from ecdsa assumptions," in *Oakland S&P'2018*, 2018.

[14] ——, "Threshold ecdsa from ecdsa assumptions: The multiparty case," in *Oakland S&P'2019*, 2019.

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

[16] S. Dziembowski, S. Faust, and K. Hostakova, "General state channel networks," in *CCS*, 2018.

[17] 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.

[18] A. for review, "A^{2}l project website," https://sites.google.com/view/a2l-website/home.

[19] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *Journal of Cryptology*, pp. 51–83, 2007.

[20] S. Goldberg, L. Reyzin, O. Sagga, and F. Baldimtsi, "Certifying rsa public keys with an efficient nizk," Cryptology ePrint Archive, Report 2018/057, 2018, https://eprint.iacr.org/2018/057.

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

[22] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "Tumblebit: An untrusted bitcoin-compatible anonymous payment hub," in *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/tumblebit-untrusted-bitcoin-compatible-anonymous-payment-hub/

[23] ——, "TumbleBit: An untrusted bitcoin-compatible anonymous payment hub," in *NDSS*, 2017.

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

[25] G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn, "An empirical analysis of anonymity in zcash," in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, 2018, pp. 463–477.

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

[27] R. Khalil, A. Gervais, and G. Felley, "Nocust - a securely scalable commit-chain," Cryptology ePrint Archive, Report 2018/642, 2018, https://eprint.iacr.org/2018/642.

[28] 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 Security Symposium*, 2016, pp. 279–296.

[29] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, 2018, pp. 583–598.

[30] 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.

[31] ——, "Teechain: Reducing storage costs on the blockchain with offline payment channels," in *Systems and Storage Conference*, 2018, p. 125.

[32] Y. Lindell, "Highly-efficient universally-composable commitments based on the ddh assumption," in *Proceedings of the 30th Annual International Conference on Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, ser. EUROCRYPT'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 446–466. [Online]. Available: http://dl.acm.org/citation.cfm?id=2008684.2008718

[33] ——, "Fast secure two-party ecdsa signing," Cryptology ePrint Archive, Report 2017/552, 2017, https://eprint.iacr.org/2017/552.

[34] Y. Lindell and A. Nof, "Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: ACM, 2018, pp. 1837–1854. [Online]. Available: http://doi.acm.org/10.1145/3243734.3243788

[35] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: ACM, 2017, pp. 455–471. [Online]. Available: http://doi.acm.org/10.1145/3133956.3134096

[36] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," Cryptology ePrint Archive, Report 2018/472, 2018, https://eprint.iacr.org/2018/472.

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

[38] P. Moreno-Sanchez, Randomrun, D. V. Le, S. Noether, B. Goodell, and A. Kate, "Dlsag: Non-interactive refund transactions for interoperable payment channels in monero," Cryptology ePrint Archive, Report 2019/595, 2019, https://eprint.iacr.org/2019/595.

[39] P. Moreno-Sanchez, T. Ruffing, and A. Kate, ""pathshuffle: Mixing credit paths for anonymous transactions in ripple"," http://crypsys.cs.purdue.edu/projects/internetOfValue/PathShuffle/.

[40] S. Noether and B. Goodel, "Dual linkable ring signatures," Monero Research Bulleting, https://web.getmonero.org/resources/research-lab/pubs/MRL-0008.pdf.

[41] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology — EUROCRYPT '99*, 1999, pp. 223–238.

[42] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," Technical Report, https://lightning.network/lightning-network-paper.pdf.

[43] G. Poupard and J. Stern, "Short proofs of knowledge for factoring," in *Public Key Cryptography*, 2000, pp. 147–166.

[44] T. Ruffing and P. Moreno-Sanchez, "ValueShuffle: Mixing Confidential Transactions for Comprehensive Transaction Privacy in Bitcoin," in *Financial Cryptography and Data Security - FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers*, ser. Lecture Notes in Computer Science, M. Brenner, K. Rohloff, J. Bonneau, A. Miller, P. Y. A. Ryan, V. Teague, A. Bracciali, M. Sala, F. Pintore, and M. Jakobsson, Eds., vol. 10323. Springer, 2017, pp. 133–154.

[45] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin," in *ESORICS'14*, ser. Lecture Notes in Computer Science, vol. 8713. Cham, Switzerland: Springer, 2014, pp. 345–364.

[46] ——, "P2P Mixing and Unlinkable Bitcoin Transactions," in *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017.

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

[48] 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.

[49] L. Valenta and B. Rowan, "Blindcoin: Blinded, Accountable Mixes for Bitcoin," in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, M. Brenner, N. Christin, B. Johnson, and K. Rohloff, Eds. Springer Berlin Heidelberg, 2015, pp. 112–126.

[50] J. H. Ziegeldorf, F. Grossmann, M. Henze, N. Inden, and K. Wehrle, "CoinParty: Secure Multi-Party Mixing of Bitcoins," in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '15. New York, NY, USA: ACM, 2015, pp. 75–86.

Appendix

A. A^2L Correctness

In this section, we define the notion of correctness for A^{2}Ls.

Definition 3 (Correctness of A^{2}Ls)**.** *Let $\mathbb{L}$ be an A^2L, $\lambda \in \mathbb{N}^+$ and $n \in \mathsf{poly}(\lambda)$. Let P_t be the intermediary, $(P_1, \dots, P_n) \in \mathbb{P}^n$ be a vector of parties, $(\mathsf{sk}_1, \dots, \mathsf{sk}_n, \mathsf{sk}_t)$ be a vector of secret keys, and $(\mathsf{pk}_{1,t}, \dots, \mathsf{pk}_{n,t})$ be a vector of public keys, such that for all $1 \le i \le n$, it holds that*

$$\{(\mathsf{sk}_i, \mathsf{pk}_{i,t}), (\mathsf{sk}_t, \mathsf{pk}_{i,t})\} \leftarrow \langle \mathsf{KGen}_{P_i}(1^\lambda), \mathsf{KGen}_{P_t}(1^\lambda)\rangle.$$

Furthermore, let $(\Pi_1, \dots, \Pi_n)$ be a vector of promises, $(\ell_1, \dots, \ell_n)$ be a vector of locks, and $(\varrho_1, \dots, \varrho_n)$ be a vector of opening information, such that for all $1 \le i, j \le n$, it holds that

$$\{\cdot, (\Pi_i, \ell_i)\} \leftarrow \langle \mathsf{Promise}_{P_t}(\mathsf{sk}_t, \mathsf{pk}_{i,t}), \mathsf{Promise}_{P_i}(\mathsf{sk}_i, \mathsf{pk}_{i,t})\rangle$$

and

$$\{\varrho_i, \cdot\} \leftarrow \langle \mathsf{Pay}_{P_j}(\mathsf{sk}_j, \mathsf{pk}_{j,t}, \ell_i), \mathsf{Pay}_{P_t}(\mathsf{sk}_t, \mathsf{pk}_{j,t})\rangle.$$

We say that $\mathbb{L}$ is correct if there exists a negligible function negl*, such that for all $1 \le i \le n$, the following holds*

$$\Pr[\mathsf{Verify}(\Pi_i, \mathsf{Open}(\Pi_i, \varrho_i)) = 1] \ge 1 - \mathsf{negl}(\lambda).$$

B. Security Analysis

Throughout this section we denote by $\mathsf{poly}(\lambda)$ any function that is bounded by a polynomial in λ, where λ 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 [7], and in the presence of *malicious adversaries* with *static corruptions*. Since both our promise and payment protocols are two party protocols, we are in the setting of no honest majority. As is standard in this setting, we consider security with abort, meaning that a corrupted party can learn output while the honest party does not.

Proof of Knowledge for Factoring. In our protocols we assume existence of a proof π_T, which is a non-interactive zero-knowledge proof that Paillier parameters (n, g) are valid, and a proof of knowledge of the associated Paillier secret key. This zero-knowledge proof of knowledge can be realized using the Poupard-Stern protocol [43] that proves knowledge of the factorization of the modulus n. Another alternative is to use the proof of [20], which certifies that RSA is a permutation by proving that $\gcd(N, \phi(N)) = 1$. This proof can be adapted to fit our needs, and this adaptation is explained in [34, Section 6.2.3].

Key Generation Functionalities. Our protocols build on key generation funtionalities for both Schnorr and ECDSA. The key generation functionalities below are taken from [35]. Ideal functionality for key generation of Schnorr signature $\mathcal{F}^{\mathsf{Schnorr}}_{\mathsf{KGen}}$ is defined below (it models a distributed key generation for discrete logarithm-based schemes).

$\mathsf{KeyGen}(\mathbb{G}, g, q)$

```
Upon invocation by both $P_1$ and $P_2$ on input $(\mathbb{G}, g, q)$ :
sample $x \leftarrow_{\$} \mathbb{Z}_q$ and compute $Q = g^x$
set $\mathsf{sk}_{P_1,P_2} = x$
sample $x_1, x_2 \leftarrow_{\$} \mathbb{Z}_q$ and a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$
send $(x_1, Q, H)$ to $P_1$ and $(x_2, Q, H)$ to $P_2$
ignore future calls by $(P_1, P_2)$
```

The ideal functionality for key generation of ECDSA signature $\mathcal{F}^{\mathsf{ECDSA}}_{\mathsf{KGen}}$ is defined as follows:

$\mathsf{KeyGen}(\mathbb{G}, g, q)$

```
Upon invocation by both $P_1$ and $P_2$ on input $(\mathbb{G}, g, q)$ :
sample $x \leftarrow_{\$} \mathbb{Z}_q$ and compute $Q = g^x$
sample $x_1, x_2 \leftarrow_{\$} \mathbb{Z}_q$ and a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$
sample a key pair $(\mathsf{sk}_{P_1,P_2}, \mathsf{pk}_{P_1,P_2}) \leftarrow \mathsf{KGen}_{\mathsf{HE}}()$
compute $c \leftarrow \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, \bar{r})$ for a random $\bar{r}$
send $(x_1, Q, H, \mathsf{sk})$ to $P_1$ and $(x_2, Q, H, c)$ to $P_2$
ignore future calls by $(P_1, P_2)$
```

We stress that the copies of these functionalities that are invoked as subroutines are fresh independent instances, and hence, the composition theorem [7] directly applies to our settings.

Schnorr-based Construction. Here we prove Theorem 1.

Proof: The proof is composed of a series of hybrids, where we gradually modify the initial experiment.

$\mathcal{H}_0$: Is identical to the construction as described in Section IV-C.

$\mathcal{H}_1$: All the calls to the commitment scheme COM are replaced with calls to the ideal functionality $\mathcal{F}_{\mathsf{COM-ZK}}$. which is defined for a relation R as described in [33].

$\mathsf{Commit}(\mathsf{sid}, x, w)$

```
Upon invocation by $P_i$, where $i \in \{1,2\}$, on input $(x, w)$ :
if some $(\mathsf{sid}, \cdot, \cdot)$ is already recorded or $(x, w) \notin R$
then ignore the message
else record $(\mathsf{sid}, i, x)$ and send $(\mathsf{com}, \mathsf{sid})$ to $P_{3-i}$
```

$\mathsf{Decommit}(\mathsf{sid})$

```
Upon invocation by $P_i$, where $i \in \{1,2\}$ :
if $(\mathsf{sid}, i, x)$ is recorded, then send $(\mathsf{decom}, \mathsf{sid}, x)$ to $P_{3-i}$
else ignore the message
```

We have to use the $\mathcal{F}_{\mathsf{COM-ZK}}$ functionality, while in our protocol the parties send commitments to non-interactive zero-knowledge proofs. This functionality is securely realized by having the prover commit to a non-interactive zero-knowledge proof using an ideal commitment functionality $\mathcal{F}_{\mathsf{COM}}$, such as the one from [32]. Instead of calling the commitment algorithm COM with (x, w), the parties send a message of the form $\mathsf{Commit}(\mathsf{sid}, x, w)$ to the ideal functionality $\mathcal{F}_{\mathsf{COM-ZK}}$. Similarly, the decommitment is replaced with a message of the form $\mathsf{Decommit}(\mathsf{sid})$. The verifying party records the messages from $\mathcal{F}_{\mathsf{COM-ZK}}$.

$\mathcal{H}_2$: All the 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 and is defined as follows.

$\mathsf{Prove}(\mathsf{sid}, x, w)$

```
Upon invocation by $P_i$, where $i \in \{1,2\}$, on input $(x, w)$ :
if $(x, w) \notin R$, then send $(\mathsf{proof}, \mathsf{sid}, x)$ to $P_{3-i}$
else ignore the message
```

Instead of calling the non-interactive zero-knowledge scheme NIZK with input (x, w), the proving party queries the ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$ with message Prove(sid, x, w). The verifier records the messages from $\mathcal{F}_{\mathsf{NIZK}}$.

$\mathcal{H}_3$: Consider the following ensemble of variables in the interaction with $\mathcal{A}$: key pairs $(\mathsf{sk}_A, \mathsf{pk}_{A,T})$ and $(\mathsf{sk}_B, \mathsf{pk}_{B,T})$, a pair $(\bar{\alpha}, (\Pi, \ell))$ such that

$$\{\cdot, (\Pi := (\beta, \cdot), \ell)\} \leftarrow \langle \mathsf{Promise}_B(\mathsf{sk}_B, \mathsf{pk}_{B,T}), \mathsf{Promise}_T(\mathsf{sk}_T, \mathsf{pk}_{B,T})\rangle$$

and

$$\{\bar{\alpha}, \cdot\} \leftarrow \langle \mathsf{Pay}_A(\mathsf{sk}_A, \mathsf{pk}_{A,T}, \ell), \mathsf{Pay}_T(\mathsf{sk}_T, \mathsf{pk}_{A,T})\rangle.$$

If for any set of these variables, the adversary returns some $\sigma := (R, s)$, such that $\mathsf{Verify}(\Pi, \sigma) = 1$, but $s \neq \mathsf{Open}(\Pi, \bar{\alpha})[s]$, then the experiment aborts.

$\mathcal{H}_4$: Consider the following ensemble of variables in the interaction with $\mathcal{A}$: key pairs $(\mathsf{sk}_A, \mathsf{pk}_{A,T})$ and $(\mathsf{sk}_B, \mathsf{pk}_{B,T})$, a pair $(\bar{\alpha}, (\Pi, \ell))$ such that

$$\{\cdot, (\Pi, \ell)\} \leftarrow \langle \mathsf{Promise}_B(\mathsf{sk}_B, \mathsf{pk}_{B,T}), \mathsf{Promise}_T(\mathsf{sk}_T, \mathsf{pk}_{B,T})\rangle$$

and

$$\{\bar{\alpha}, \cdot\} \leftarrow \langle \mathsf{Pay}_A(\mathsf{sk}_A, \mathsf{pk}_{A,T}, \ell), \mathsf{Pay}_T(\mathsf{sk}_T, \mathsf{pk}_{A,T})\rangle.$$

If for any set of these variables, the adversary returns some $\sigma := (R, s)$, such that $\mathsf{Verify}(\Pi, \sigma) = 1$, before Alice outputs $\bar{\alpha}$, such that $\mathsf{Verify}(\Pi, \mathsf{Open}(\Pi, \bar{\alpha})) = 1$ then the experiment aborts.

$\mathcal{S}$: The actions of the simulator $\mathcal{S}$ are dictated by interacting with $\mathcal{F}$. If $\mathcal{A}$ interacts with an honest user, then the simulator queries the corresponding interface of $\mathcal{F}$. More precisely, it is queried by $\mathcal{F}$ on the following set of inputs:

- Promise: The simulator initiates the promise procedure with the adversary and replies with $\perp$ if the execution is not successful, otherwise replies with a valid promise and lock.
- Pay: The simulator initiates the pay procedure with the adversary and replies with $\perp$ and if the execution is not successful, otherwise it releases the opening information of the corresponding lock.
- Open: The simulator returns the opened lock data.

Additionally, $\mathcal{S}$ obtains the pair $(n, g), (\lambda, \mu)$, by extracting them from the proof π_T, where (n, g) is the Paillier public key of Tumbler, and (λ, μ) is the corresponding secret key of Tumbler.

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

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

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

Proof: In order to show this claim, we introduce two intermediate hybrids.

$\mathcal{H}_2^*$: All the calls to the promise protocol are replaced with calls to the $\mathcal{F}_{\mathsf{Promise}}$ ideal functionality, which is defined as follows.

$\mathsf{PromiseSign}(\mathsf{sid}, m, \alpha)$

```
Upon invocation by Tumbler and Bob on input $(\mathsf{sid}, \mathsf{pk}, m, \alpha)$ :
if some $(\mathsf{sid}, \cdot, \cdot, \cdot)$ is already recorded, then ignore the message
else record $(\mathsf{sid}, \mathsf{pk}, m, \alpha)$
compute $(R, s) \leftarrow \mathsf{Sig}_{\mathsf{Schnorr}}(\mathsf{sk}_{B,T}, m)$
return $(R, s - \alpha)$
```

We note that the key $\mathsf{sk}_{B,T}$ refers to the previously established key between Bob and Tumbler in the call to the $\mathcal{F}_{\mathsf{KGen}}^{\mathsf{Schnorr}}$.

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

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

Proof: The proof consists of the description of the simulator for the interactive promise protocol. Since the promise protocol is executed between Tumbler and Bob, we describe two simulators depending on whether the adversary is playing the role of Tumbler or Bob.

1) Bob corrupted: After agreeing on a message m, the simulator $\mathcal{S}$ samples a random $\alpha^* \leftarrow_{\$} \mathbb{Z}_q$, and queries PromiseSign on input $(\mathsf{sid}, m, \alpha^*)$, for a random sid, and obtains $\sigma' := (R', s')$. $\mathcal{S}$ computes $A^* = g^{\alpha^*}$ and $c^* =$

$\mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_T, m)$, where pk_T is the Paillier public key of Tumbler. $\mathcal{S}$ sends $((\mathsf{com}, \mathsf{sid}), A^*, c^*, (\mathsf{proof}, \mathsf{sid}, \{\exists \alpha^* \mid A^* = g^{\alpha^*}\}))$ to $\mathcal{A}$. At some point of the execution $\mathcal{A}$ sends $(R'_1, (\mathsf{prove}, \{\exists k'_1 \mid R'_1 = g^{k'_1}\}, k'_1))$. $\mathcal{S}$ verifies that $R'_1 = g^{k'_1}$, and if this is not the case $\mathcal{S}$ simulates Tumbler aborting. $\mathcal{S}$ replies with

$$\left(\begin{array}{l} \mathsf{decom}, \mathsf{sid}, \left(\begin{array}{l} R^* = R'/(R'_1 \cdot A^*), \\ \mathsf{proof}, \mathsf{sid}, \\ \{\exists k^* \mid R^* = g^{k^*}\} \end{array} \right), \\ (s' - k'_1 + e \cdot x'_1) \end{array} \right)$$

where $e = H(\mathsf{pk} \| R^* \| m)$, and x'_1 is the value returned by the key generation to $\mathcal{A}$. The rest of the execution is unchanged.
The distribution induced by simulator is identical to the real execution except for the way c^* is computed (which corresponds to c in the real protocol). However, α is sample uniformly randomly from $\mathbb{Z}_q$ both in the real execution and the simulation. Hence, by the indistinguishability of Paillier the distributions are indistinguishable.

2) Tumbler corrupted: After agreeing on a message m, the simulator $\mathcal{S}$ is given

$$\left(\begin{array}{l} \mathsf{com}, \mathsf{sid}, \begin{array}{l} \left(R'_2, \begin{array}{l} \mathsf{prove}, \mathsf{sid}, \\ \{\exists k'_2 \mid R'_2 = g^{k'_2}\}, k'_2 \end{array} \right), \\ \left(A, \begin{array}{l} \mathsf{prove}, \mathsf{sid}, \\ \{\exists \alpha \mid A = g^{\alpha}\}, \alpha \end{array} \right), c \end{array} \end{array} \right)$$

by $\mathcal{A}$. $\mathcal{S}$ verifies that $R'_2 = g^{k'_2}$ and $A = g^{\alpha}$. If the verification fails, $\mathcal{S}$ simulates Bob aborting. $\mathcal{S}$ queries $\mathsf{PromiseSign}$ on input $(\mathsf{sid}, m, \alpha)$, and obtains $\sigma' := (R', s')$. $\mathcal{S}$ sends $(R^* = R'/(R'_2 \cdot A), (\mathsf{proof}, \mathsf{sid}, \{\exists k^* \mid R^* = g^{k^*}\}))$ to $\mathcal{A}$, and receives $((\mathsf{decom}, \mathsf{sid}), s'_2 = k'_2 - e' \cdot x'_2)$, where $e' = H(\mathsf{pk} \| R^* \| m)$, and x'_2 is the value returned by the key generation to $\mathcal{A}$. The rest of the execution is unchanged.
Simulator is efficient and the distribution induced by the simulated view is identical to the one of the original protocol.

■

Next, we define the second intermediate hybrid.

$\mathcal{H}_2^{\dagger}$: All the calls to the payment protocol are replaced with calls to the $\mathcal{F}_{\mathsf{Pay}}$ ideal functionality, which is defined as follows.

```
PaymentSign(sid, $m$, $\gamma$)
Upon invocation by Tumbler and Alice on input $(\mathsf{sid}, m, \gamma)$ :
if some $(\mathsf{sid}, \cdot, \cdot)$ is already recorded, then ignore the message
else record $(\mathsf{sid}, m, \gamma)$
and compute $(R, s) \leftarrow \mathsf{Sig}_{\mathsf{Schnorr}}(\mathsf{sk}_{A,T}, m)$
return $(R, s - \gamma)$
```

We note that $\mathsf{sk}_{A,T}$ refers to the previously established key between Alice and Tumbler in the call to the $\mathcal{F}^{\mathsf{Schnorr}}_{\mathsf{KGen}}$.

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

$$\mathsf{EXEC}_{\mathcal{H}_2^*, \mathcal{A}, \mathcal{E}} \approx \mathsf{EXEC}_{\mathcal{H}_2^{\dagger}, \mathcal{A}, \mathcal{E}}.$$

Proof: Similar to the proof of Lemma 4, we define two simulators. The payment protocol is run between Tumbler and Alice, hence, we define simulators for when one or the other is corrupted.

1) Alice corrupted: Prior to the interaction the simulator $\mathcal{S}$ is given π_T. After agreeing on a message m, $\mathcal{S}$ sends $(\mathsf{com}, \mathsf{sid})$ to $\mathcal{A}$, for a random sid. At some point of the execution $\mathcal{A}$ sends $(c'', R_1, (\mathsf{prove}, \{\exists k_1 \mid R_1 = g^{k_1}\}, k_1))$. If $R_1 \neq g^{k_1}$, then $\mathcal{S}$ simulates Tumbler aborting. $\mathcal{S}$ extracts the Paillier secret key sk_T of Tumbler from π_T, decrypts c'' to obtain $\gamma \leftarrow \mathsf{Dec}_{\mathsf{HE}}(\mathsf{sk}_T, c'')$, and computes $A^* = g^{\gamma}$. $\mathcal{S}$ queries $\mathsf{PaymentSign}$ on input $(\mathsf{sid}, m, \gamma)$, and receives $\sigma := (R, s)$. $\mathcal{S}$ sends

$$\left(\begin{array}{l} \mathsf{decom}, \mathsf{sid}, \left(\begin{array}{l} R^* = R/(R_1 \cdot A^*), \\ \mathsf{proof}, \mathsf{sid}, \\ \{\exists k^* \mid R^* = g^{k^*}\} \end{array} \right), \\ (s - k_1 + e \cdot x_1), A^* \end{array} \right)$$

to $\mathcal{A}$, where $e = H(\mathsf{pk} \| R^* \| m)$, and x_1 is the value returned by the key generation to $\mathcal{A}$. The rest of the execution is unchanged.
Simulator is efficient and the distribution induced by the simulated view is identical to the one of the original protocol.

2) Tumbler corrupted: After agreeing on a message m, the simulator $\mathcal{S}$ is given

$$\left(\mathsf{com}, \mathsf{sid}, \left(R_2, \begin{array}{l} \mathsf{prove}, \mathsf{sid}, \\ \{\exists k_2 \mid R_2 = g^{k_2}\}, k_2 \end{array} \right) \right)$$

by $\mathcal{A}$. If $R_2 \neq g^{k_2}$, then $\mathcal{S}$ simulates Bob aborting. $\mathcal{S}$ samples $\gamma^* \leftarrow\!\!\$\, \mathbb{Z}_q$, computes $A^* = g^{\gamma^*}$, encrypts γ^* as $c^* = \mathsf{Enc}_{\mathsf{HE}}(pk_T, \gamma^*)$, and it queries $\mathsf{PaymentSign}$ on input $(\mathsf{sid}, m, \gamma^*)$. The simulator receives $\sigma := (R, s)$, and sends $(c^*, R^* = R/(R_2 \cdot A^*), (\mathsf{proof}, \mathsf{sid}, \{\exists k^* \mid R^* = g^{k^*}\}))$ to $\mathcal{A}$. $\mathcal{S}$ receives $((\mathsf{decom}, \mathsf{sid}), s_2 = k_2 - e \cdot x_2, A^*)$, where $e' = H(\mathsf{pk} \| R^* \| m)$, and x_2 is the value returned by the key generation to $\mathcal{A}$. $\mathcal{S}$ replies with s. The rest of the execution is unchanged.
The distribution induced by simulator is identical to the real execution except for the way c^* is computed (which corresponds to c in the real protocol). However, the same argument about the indistinguishability from Lemma 4 applies here.

Both simulators are efficient and the distributions induced by the simulated views are identical to the ones of the original protocol. ■

Next, we continue with the proof of Lemma 3. Let cheat be the event that triggers an abort of the experiment in $\mathcal{H}_3$. Assume towards contradiction that $\Pr[\mathsf{cheat} \mid \mathcal{H}_2^{\dagger}] \geq \frac{1}{\mathsf{poly}(\lambda)}$, then we can construct the following reduction against the strong existential unforgeability of Schnorr signature. 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. Let Q be the key generated in the j-th interaction, the reduction sets $Q = \mathsf{pk}$. All the calls to the signing algorithm are redirected to the signing oracle. If the event cheat happens, the reduction returns the corresponding $(\mathsf{pk}^*, m^*, \sigma^* := (R^*, s^*))$, otherwise it aborts.

The reduction is clearly efficient. Assume that j is the index of the interaction where cheat happens. Note that in the case the guess of the reduction is correct we have that $\mathsf{pk}^* = \mathsf{pk}_{B,T}$. Since cheat happens we have that $\mathsf{Verify}_{\mathsf{Schnorr}}(\mathsf{pk}^*, m^*, \sigma^*) = 1$, but $s^* \neq \mathsf{Open}(\Pi, \bar{\alpha})[s]$, where Π and $\bar{\alpha}$ are returned from the promise and pay protocols, respectively. Recall that $\bar{\alpha} = \alpha + \beta$ and Open parses Π as (R', s'), where $s' = s_j - \alpha$, for some $\alpha \in \mathbb{Z}_q$, where s_j is the answer of the oracle on the j-th session on input m_j. Substituting we get

$$\begin{aligned} s^* &\neq \mathsf{Open}(\Pi, \bar{\alpha})[s] \\ &\neq s' + (\bar{\alpha} - \beta) \\ &\neq s_j - \alpha + \alpha + \beta - \beta \\ &\neq s_j \end{aligned}$$

as expected. Since each message uniquely identifies a session, this implies that $(\mathsf{pk}^*, m^*, \sigma^*)$ 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 $\Pr[\mathsf{cheat} \mid \mathcal{H}_2^{\dagger}] \leq \mathsf{negl}(\lambda)$. ■

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

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

Proof: Let $q \in \mathsf{poly}(\lambda)$ be a bound on the total number of interactions. Let cheat denote an event that triggers an abort in $\mathcal{H}_4$, but not in $\mathcal{H}_3$. We prove the indistinguishability of $\mathcal{H}_3$ and $\mathcal{H}_4$ by showing that $\Pr[\mathsf{cheat} \mid \mathcal{H}_3] \leq \mathsf{negl}(\lambda)$. Assume that the converse is true, then we can construct the following reduction against the discrete logarithm problem: On input some $A^* \in \mathbb{G}$ and a public key pk, the reduction guesses a session $j \in [1, q]$. The reduction replaces A from the first message of the promise protocol with A^*. If Alice is requested to call the payment protocol, the reduction aborts. At some point of the execution $\mathcal{A}$ outputs some $(\mathsf{pk}^*, m^*, \sigma^* := (R^*, s^*))$. The reduction returns $g^{s^* - s'}$, where s' is part of the output of the promise protocol.

The reduction is clearly efficient, and whenever j is guessed correctly, the reduction does not abort, and we also have that $\mathsf{pk}^* = \mathsf{pk}_{B,T}$. The event cheat happens only in the case where $\mathsf{Verify}_{\mathsf{Schnorr}}(\mathsf{pk}^*, m^*, \sigma^*) = 1$, but payment protocol has not been executed. Recall that $s' = s_j - \alpha$ and $A = g^{\alpha}$, for some $\alpha \in \mathbb{Z}_q$, where s_j is the answer of the oracle on the j-th session on input m_j. We note that we replaced A with the input A^* of the reduction, hence $A = A^*$ in this case. As argued in the proof of Lemma 3, if $s^* \neq s_j$, then we have an attacker against the strong unforgeability of the signature scheme. Hence, it follows that $s^* = s_j$ with all but negligible probability. Substituting we have

$$\begin{aligned} g^{s^* - s'} &= g^{s^* - (s_j - \alpha)} \\ &= g^{\alpha} \\ &= A \end{aligned}$$

as expected. Since, by assumption this happens with probability at least $\frac{1}{q \cdot n \cdot \mathsf{poly}(\lambda)}$, we have a successful attacker against the discrete logarithm problem. This proves our lemma. ■

This concludes the proof. ■

ECDSA-based Construction. Here we prove Theorem 2.

Proof: The sequence of hybrids that we need are identical to the ones used in the proof of the Schnorr-based construction. Hence, here we only prove the indistinguishability of the neighboring experiments which require modifications in the argument. If the argument is the same, then the proof is omitted.

Next, we prove the indistinguishability of the neighboring hybrids.

Lemma 7. *For all* PPT *distinguisher $\mathcal{E}$ it holds that*

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

Proof: Similar to the proof of the Schnorr-based construction, we defined two intermediate hybrids.

$\mathcal{H}_2^*$: The promise protocol is substituted with the $\mathcal{F}_{\mathsf{Promise}}$ ideal functionality, defined as follows.

```
PromiseSign$(\mathsf{sid}, m, \alpha)$
Upon invocation by Tumbler and Bob on input $(\mathsf{sid}, \mathsf{pk}, m, \alpha)$ :
if some $(\mathsf{sid}, \cdot, \cdot, \cdot)$ is already recorded, then ignore the message
else record $(\mathsf{sid}, \mathsf{pk}, m, \alpha)$
compute $(r, s) \leftarrow \mathsf{Sig}_{\mathsf{ECDSA}}(\mathsf{sk}_{B,T}, m)$
return $(r, \min(s \cdot \alpha, -s \cdot \alpha))$
```

Recall that the key $\mathsf{sk}_{B,T}$ refers to the key established between Bob and Tumbler in the call to the $\mathcal{F}_{\mathsf{KGen}}^{\mathsf{ECDSA}}$ functionality.

Lemma 8. *For all* PPT *distinguisher $\mathcal{E}$ it holds that*

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

Proof: We define two simulators since the promise protocol is executed between two parties (namely, Tumbler and Bob).

1) Bob corrupted: After agreeing on a message m, the simulator $\mathcal{S}$ samples a random $\alpha^* \leftarrow_{\$} \mathbb{Z}_q$, and queries PromiseSign on input $(\mathsf{sid}, m, \alpha^*)$, for a random sid, obtains $\sigma' := (r', s')$ and sets $R' = g^{H(m) \cdot (s')^{-1}} \cdot Q^{r' \cdot (s')^{-1}}$. $\mathcal{S}$ computes $A^* = g^{\alpha^*}$ and $c^* = \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_T, m)$, where pk_T is the Paillier public key of Tumbler. $\mathcal{S}$ sends $((\mathsf{com}, \mathsf{sid}), A^*, c^*, (\mathsf{proof}, \mathsf{sid}, \{\exists \alpha \mid A^* = g^{\alpha^*}\}))$ to $\mathcal{A}$. At some point of the execution $\mathcal{A}$ sends $(R_1', (\mathsf{prove}, \{\exists k_1' \mid R_1' = g^{k_1'}\}, k_1'))$. $\mathcal{S}$ verifies that $R_1' = g^{k_1'}$, and if this is not the case $\mathcal{S}$ simulates Tumbler aborting. $\mathcal{S}$ samples a random $\rho \leftarrow_{\$} \mathbb{Z}_{q^2}$ and computes $c' \leftarrow \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_T, k_1' \cdot s' + \rho q)$. $\mathcal{S}$ provides the attacker with

$$\left(\mathsf{decom}, sid, \left(\begin{array}{l} R^* = (R')^{(k_1')^{-1}}, \\ R_2 = (R^*)^{(\alpha^*)^{-1}}, \\ (\mathsf{proof}, \mathsf{sid}, \\ \{\exists k^* \mid R_2 = g^{k^*}\}), \\ (\mathsf{proof}, \mathsf{sid}, \\ \{\exists \alpha^* \mid R^* = (R_2)^{\alpha^*}\}), \\ (\mathsf{proof}, \mathsf{sid}, \\ \ \{\exists \alpha^* \mid A^* = g^{\alpha^*} \wedge \\ \ R^* = (R_2)^{\alpha^*}\}) \end{array} \right), c' \right).$$

The rest of the execution is unchanged.

The distribution induced by the simulator is identical to the real execution except for the way c^* and c' are computed. The same argument from the proof of Lemma 4 apply about the distribution of c^*. Whereas, for the distribution of c' we can prove the statistical proximity using the following lemma (proved in [33]):

Lemma 9. *[33] For all $(k, s, t) \in \mathbb{Z}_q$ and for a random $\rho \in \mathbb{Z}_{q^2}$, the distributions $\mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, k \cdot s \bmod q + tq + \rho q)$ and $\mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, k \cdot s \bmod q + \rho q)$ are statistically close.*

In the real world c' is computed as $\mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, k \cdot s \bmod q + tq + \rho q)$, for some t that is bound by q. The reason t is bound between 0 and q is that the only operations performed without modular reduction are one multiplication and one addition, which cannot increase the result more than q^2. Since the distributions are identical, the indistinguishability follows.

2) Tumbler corrupted: After agreeing on a message m, the simulator $\mathcal{S}$ is given
$$\left(\mathsf{com}, sid, \begin{matrix} \left(R_2', \begin{matrix}\mathsf{prove}, \mathsf{sid}, \\ \{\exists k_2' \mid R_2' = g^{k_2'}\}, k_2'\end{matrix}\right), \\ \left(A, \begin{matrix}\mathsf{prove}, \mathsf{sid}, \\ \{\exists \alpha \mid A = g^{\alpha}\}, \alpha\end{matrix}\right), c\end{matrix}\right)$$
by $\mathcal{A}$. $\mathcal{S}$ verifies that $R_2' = g^{k_2'}$ and $A = g^{\alpha}$. If the verification fails, $\mathcal{S}$ simulates Bob aborting. $\mathcal{S}$ queries $\mathsf{PromiseSign}$ on input $(\mathsf{sid}, m, \alpha)$, obtains $\sigma' := (r', s')$ and sets $R' = g^{H(m)\cdot(s')^{-1}} \cdot Q^{r' \cdot (s')^{-1}}$. $\mathcal{S}$ sends $(R^* = (R')^{(k_2')^{-1} \cdot \alpha^{-1}}, (\mathsf{proof}, \mathsf{sid}, \{\exists k^* \mid R^* = g^{k^*}\}))$ to $\mathcal{A}$, and receives
$$\left(\mathsf{decom}, sid, \begin{matrix} \left(R_c', \begin{matrix}\mathsf{prove}, \mathsf{sid}, \\ \{\exists \alpha \mid R_c' = (R_2')^{\alpha}\}, \alpha\end{matrix}\right), \\ \left(A, R_c', \begin{matrix}\mathsf{prove}, \mathsf{sid}, \\ \{\exists \alpha \mid A = g^{\alpha} \wedge \\ R_c' = (R_2')^{\alpha}\}, \alpha\end{matrix}\right), \\ c' \end{matrix}\right).$$
$\mathcal{S}$ verifies that $R_c' = (R_2')^{\alpha}$ and $A = g^{\alpha}$. If the verification fails $\mathcal{S}$ simulates Bob aborting. $\mathcal{S}$ checks
$$\mathsf{Dec}_{\mathsf{HE}}(\mathsf{sk}, c') = \bar{r} \cdot r' \cdot (k_2')^{-1} + H(m) \cdot (k_2')^{-1} \bmod q,$$
where $\bar{r}$ was sampled in the key generation algorithm. If the check holds, then the rest of the execution proceeds unchanged, else $\mathcal{S}$ simulates Bob aborting.

The distribution induced by the simulator is identical to the real execution except for the way c' is computed. However, we can show indistinguishability using the a modified simulator, which is given the oracle $\mathcal{O}(c', a, b)$ as is defined in the following security experiment of the Paillier encryption scheme [33]:

```
$\mathsf{Exp\text{-}ecCPA}_{\mathsf{HE}}^{\mathcal{A}}(\lambda)$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}_{\mathsf{HE}}(1^{\lambda})$
$(w_0, w_1) \leftarrow_{\$} \mathbb{Z}_q$
$Q = g^{w_0}$
$b \leftarrow_{\$} \{0, 1\}$
$c \leftarrow \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, w_b)$
$b' \leftarrow \mathcal{A}(\mathsf{pk}, c, Q)^{\mathcal{O}(\cdot,\cdot,\cdot)}$
where $\mathcal{O}(c', a, b)$ returns 1 iff $\mathsf{Dec}_{\mathsf{HE}}(\mathsf{sk}, c') = a + b \cdot w_b \bmod q$
and $\mathcal{O}$ halts after the first time it returns 0
return 1 iff $b = b'$
```

The modified simulator queries the oracle on input $(c', a = H(m) \cdot (k_2')^{-1}, b = r' \cdot (k_2')^{-1})$. It is apparent that the modified simulator accepts only if the original simulator accepts. Assume towards contradiction that the modified simulator can be efficiently distinguished form the real world experiment. Then, we can give the following reduction to the security of Paillier encryption scheme: On input (pk, c, Q), the reduction simulates the inputs of $\mathcal{A}$ as described in the modified simulator using the input pk, Q, and c as the corresponding variables. The reduction is clearly efficient. We note that if $b = 0$, then $c = \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, w_0)$ and $Q = g^{w_0}$, which is identical to the real world execution by setting $w_0 = x_1$. In contrast, if $b = 1$, then we have that $c = \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, w_1)$ and $Q = g^{w_0}$, where w_1 is uniformly distributed in $\mathbb{Z}_q$, which is identical to the modified simulated experiment. This means that the modified simulation is computationally indistinguishable from the real world experiment. This concludes the proof of Lemma 8, while the modified simulation and the original simulation are identical to the eyes of the adversary. ■

Next, we define the second intermediate hybrid.

$\mathcal{H}_2^{\dagger}$: The payment protocol is substituted with the $\mathcal{F}_{\mathsf{Pay}}$ ideal functionality, which is defined as follows.

```
$\mathsf{PaymentSign}(\mathsf{sid}, m, \gamma)$
Upon invocation by Tumbler and Alice on input $(\mathsf{sid}, m, \gamma)$ :
if some $(\mathsf{sid}, \cdot, \cdot)$ is already recorded, then ignore the message
else record $(\mathsf{sid}, m, \gamma)$
and compute $(R, s) \leftarrow \mathsf{Sig}_{\mathsf{ECDSA}}(\mathsf{sk}_{A,T}, m)$
return $(r, \min(s \cdot \gamma, -s \cdot \gamma))$
```

We note that $\mathsf{sk}_{A,T}$ refers to the previously established key between Alice and Tumbler in the call to the $\mathcal{F}_{\mathsf{KGen}}^{\mathsf{Schnorr}}$.

Lemma 10. *For all* PPT *distinguisher $\mathcal{E}$ it holds that*
$$\mathsf{EXEC}_{\mathcal{H}_2^*, \mathcal{A}, \mathcal{E}} \approx \mathsf{EXEC}_{\mathcal{H}_2^{\dagger}, \mathcal{A}, \mathcal{E}}.$$

Proof: We define two simulators, one when Alice is corrupted, and the other one when Tumbler is corrupted.

1) Alice corrupted: Prior to the interaction the simulator $\mathcal{S}$ is given π_T. After agreeing on a message m, $\mathcal{S}$ sends $(\mathsf{com}, \mathsf{sid})$ to $\mathcal{A}$, for a random sid. At some point of the execution $\mathcal{A}$ sends $(c'', R_1, (\mathsf{prove}, \{\exists k_1 \mid R_1 = g^{k_1}\}, k_1))$. If $R_1 \neq g^{k_1}$, then $\mathcal{S}$ simulates Tumbler aborting. $\mathcal{S}$ extracts the Paillier secret key sk_T of Tumbler from π_T, decrypts c'' to obtain $\gamma \leftarrow \mathsf{Dec}_{\mathsf{HE}}(\mathsf{sk}_T, c'')$, and computes $A^* = g^{\gamma}$. $\mathcal{S}$ queries $\mathsf{PaymentSign}$ on input $(\mathsf{sid}, m, \gamma)$, receives $\sigma := (r, s)$, and sets $R =$

$g^{H(m)\cdot s^{-1}} \cdot Q^{r\cdot s^{-1}}$. $\mathcal{S}$ samples a random $\rho \leftarrow_\$ \mathbb{Z}_{q^2}$, computes $c \leftarrow \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_T, k_1 \cdot s + \rho q)$, and sends

$$\left(\mathsf{decom}, sid, \left(\begin{array}{l} R_c = R^{(k_1)^{-1}}, \\ R_2 = (R_c)^{\alpha^{-1}}, \\ (\mathsf{proof}, \mathsf{sid}, \\ \{\exists k^* \mid R_2 = g^{k^*}\}), \\ (\mathsf{proof}, \mathsf{sid}, \\ \{\exists \alpha \mid R_c = (R_2)^\alpha\}), \\ (\mathsf{proof}, \mathsf{sid}, \\ \ \{\exists \alpha \mid A = g^\alpha \wedge \\ \ R_c = (R_2)^\alpha\}) \end{array}\right), c\right).$$

to $\mathcal{A}$. The rest of the execution is unchanged.

The distribution induced by the simulator is identical to the real execution except for the way c is computed. However, the same argument about the statistical proximity as is given in Lemma 9 applies here too.

2) Tumbler corrupted: After agreeing on a message m, the simulator $\mathcal{S}$ is given

$$\left(\mathsf{com}, \mathsf{sid}, \left(R_2, \begin{array}{l}\mathsf{prove}, \mathsf{sid}, \\ \{\exists k_2 \mid R_2 = g^{k_2}\}, k_2\end{array}\right)\right)$$

by $\mathcal{A}$. If $R_2 \neq g^{k_2}$, then $\mathcal{S}$ simulates Bob aborting. $\mathcal{S}$ samples $\gamma^* \leftarrow_\$ \mathbb{Z}_q$, computes $A^* = g^{\gamma^*}$, encrypts γ^* as $c^* = \mathsf{Enc}_{\mathsf{HE}}(pk_T, \gamma^*)$, and it queries $\mathsf{PaymentSign}$ on input $(\mathsf{sid}, m, \gamma^*)$. $\mathcal{S}$ receives $\sigma := (r, s)$, and sets $R = g^{H(m)\cdot s^{-1}} \cdot Q^{r\cdot s^{-1}}$. $\mathcal{S}$ sends $(c^*, R^* = R^{(k_2)^{-1}\cdot(\gamma^*)^{-1}}, (\mathsf{proof}, \mathsf{sid}, \{\exists k^* \mid R^* = g^{k^*}\}))$ to $\mathcal{A}$. $\mathcal{S}$ receives

$$\left(\begin{array}{ll} & \left(R_c, \begin{array}{l}\mathsf{prove}, \mathsf{sid}, \\ \{\exists \gamma \mid R_c = (R_2)^{\gamma^*}\}, \gamma^*\end{array}\right), \\ \mathsf{decom}, \mathsf{sid}, & \left(A^*, R_c, \begin{array}{l}\mathsf{prove}, \mathsf{sid}, \\ \{\exists \gamma^* \mid A^* = g^\gamma \wedge \\ R_c = (R_2)^{\gamma^*}\}, \gamma^*\end{array}\right), \\ c & \end{array}\right).$$

The rest of the execution is unchanged.

The distribution induced by the simulator is identical to the real execution except for the way c and c^* are computed. However, the indistinguishability argument from the proof of Lemma 8 applies here for c, and the argument from the proof of Lemma 5 applies for c^*. This concludes the proof of Lemma 10. ∎

Next, we continue with the proof of Lemma 7. Let cheat be the event that triggers an abort of the experiment in $\mathcal{H}_3$. Assume towards contradiction that $\Pr[\mathsf{cheat} \mid \mathcal{H}_2^\dagger] \geq \frac{1}{\mathsf{poly}(\lambda)}$, then we can construct the following reduction against the strong existential unforgeability of ECDSA signature. 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. Let Q be the key generated in the j-th interaction, the reduction sets $Q = \mathsf{pk}$. All the calls to the signing algorithm are redirected to the signing oracle. If the event cheat happens, the reduction returns the corresponding $(\mathsf{pk}^*, m^*, \sigma^* := (r^*, s^*))$, otherwise it aborts.

The reduction is clearly runs in polynomial time. Assume that j is the index of the interaction where cheat happens. Note that in the case the guess of the reduction is correct we have that $\mathsf{pk}^* = \mathsf{pk}_{B,T}$. Since cheat happens we have that $\mathsf{Verify}_{\mathsf{ECDSA}}(\mathsf{pk}^*, m^*, \sigma^*) = 1$, but $s^* \neq \mathsf{Open}(\Pi, \bar{\alpha})[s]$, where Π and $\bar{\alpha}$ are returned from the promise and pay protocols, respectively. Recall that $\bar{\alpha} = \alpha \cdot \beta$ and Open parses Π as (r', s'), where $s' = s_j \cdot \alpha$, for some $\alpha \in \mathbb{Z}_q$, where s_j is the answer of the oracle on the j-th session on input m_j.

Substituting we get

$$\begin{aligned} s^* &\neq \mathsf{Open}(\Pi, \bar{\alpha})[s] \\ &\neq s' \cdot (\bar{\alpha} \cdot \beta^{-1})^{-1} \\ &\neq s_j \cdot \alpha \cdot (\alpha \cdot \beta \cdot \beta^{-1})^{-1} \\ &\neq s_j \cdot \alpha \cdot \alpha^{-1} \cdot \beta^{-1} \cdot \beta \\ &\neq s_j \end{aligned}$$

as expected. Since each message uniquely identifies a session, this implies that $(\mathsf{pk}^*, m^*, \sigma^*)$ 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 $\Pr[\mathsf{cheat} \mid \mathcal{H}_2^\dagger] \leq \mathsf{negl}(\lambda)$. ∎

Lemma 11. *For all* PPT *distinguisher* $\mathcal{E}$ *it holds that*

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

Proof: Let $q \in \mathsf{poly}(\lambda)$ be a bound on the total number of interactions. Let cheat denote an event that triggers an abort in $\mathcal{H}_4$, but not in $\mathcal{H}_3$. We prove the indistinguishability of $\mathcal{H}_3$ and $\mathcal{H}_4$ by showing that $\Pr[\mathsf{cheat} \mid \mathcal{H}_3] \leq \mathsf{negl}(\lambda)$. Assume that the converse is true, then we can construct the following reduction against the discrete logarithm problem: On input some $A^* \in \mathbb{G}$ and a public key pk, the reduction guesses a session $j \in [1, q]$. The reduction replaces A from the first message of the promise protocol with A^*. If Alice is requested to call the payment protocol, the reduction aborts. At some point of the execution $\mathcal{A}$ outputs some $(\mathsf{pk}^*, m^*, \sigma^* := (R^*, s^*))$. The reduction returns $g^{(s^*)^{-1}\cdot s'}$, where s' is part of the output of the promise protocol.

The reduction is clearly efficient, and whenever j is guessed correctly, the reduction does not abort, and we also have that $\mathsf{pk}^* = \mathsf{pk}_{B,T}$. The event cheat happens only in the case where $\mathsf{Verify}_{\mathsf{ECDSA}}(\mathsf{pk}^*, m^*, \sigma^*) = 1$, but payment protocol has not been executed. Recall that $s' = s_j \cdot \alpha$ and $A = g^\alpha$, for some $\alpha \in \mathbb{Z}_q$, where s_j is the answer of the oracle on the j-th session on input m_j. We note that we replaced A with the input A^* of the reduction, hence $A = A^*$ in this case. As argued in the proof of Lemma 7, if $s^* \neq s_j$, then we have an attacker against the strong unforgeability of the signature scheme. Hence, it follows that $s^* = s_j$ with all but negligible probability. Substituting we have

$$\begin{aligned} g^{s^* - s'} &= g^{(s^*)^{-1}\cdot(s_j\cdot\alpha)} \\ &= g^\alpha \\ &= A \end{aligned}$$

as expected. Since, by assumption this happens with probability at least $\frac{1}{q\cdot n\cdot\mathsf{poly}(\lambda)}$, we have a successful attacker against the discrete logarithm problem. This proves our lemma. ∎

This concludes the proof. ∎

C. PCH from Anonymous Atomic Locks

In this section, we show that A^{2}L are sufficient to construct a fully-fledged PCH. In order to do that, we first define the ideal functionality for PCH. We then detail the PCH construction sketched in Section VI. Finally, we analyze the security of the PCH construction.

1) Ideal Functionalities: We require the ideal functionality for anonymous atomic locks $\mathcal{F}_{\mathsf{A^2L}}$ as described in Figure 2. That is, all parties have oracle access to $\mathcal{F}_{\mathsf{A^2L}}$ through the specified interfaces.

Furthermore, we require the existence of a blockchain $\mathcal{B}$ modeled as a trusted append-only bulletin board. The corresponding ideal functionality $\mathcal{F}_{\mathsf{B}}$, as defined in [15], is used to store and update the balance of every party. It is defined in the global UC (GUC) model [8], since it provides values that should be globally accessible, and it can be updated by multiple instances of our ideal functionality or by other protocols simultaneously. In order to update the balance of a party P, $\mathcal{F}_{\mathsf{B}}$ processes the messages (add, P, x) and (remove, P, x), which allow to add/remove x coins to/from a party P's account, respectively. For readability we write the balance of a party P in $\mathcal{B}$ as $\mathcal{B}[P]$. The state of $\mathcal{F}_{\mathsf{B}}$ is available to all parties, that is, at any point in the execution, a party P can send a distinguished message read to $\mathcal{F}_{\mathsf{B}}$, which sends the whole transcript of $\mathcal{B}$ to P. Moreover, we denote the number of entries in $\mathcal{B}$ as $|\mathcal{B}|$, and we model time as the number of entries of the blockchain $\mathcal{B}$ (i.e., time $\Delta = |\mathcal{B}|$). Note that it is possible to elapse time by adding dummy entries to $\mathcal{B}$ and that the time is available to all parties by simply reading $\mathcal{B}$. Lastly, for readability, we assume that users can specify arbitrary *contracts*, that is, validity of transactions from users can be associated with arbitrary conditions that must be satisfied in order to make the transaction effective. $\mathcal{F}_{\mathsf{B}}$ is then assumed to enforce that contract clauses are fulfilled before the transaction is added to $\mathcal{B}$.

As defined in Section III, here we assume synchronous communication between users, modeled by the functionality $\mathcal{F}_{\mathsf{syn}}$, and secure message transmission channels between users, modeled by $\mathcal{F}_{\mathsf{smt}}$.

Multi-session Extension. Composition theorem requires that each call of every ideal functionality spawns an independent instance of the corresponding functionality. However, our $\mathcal{F}_{\mathsf{A^2L}}$ functionality formally requires a joint state between sessions. More precisely, the KGen protocols that are used for establishing pairwise links are shared between multiple promise/payment instances, which might potentially result in shared keys between the different instances of A^{2}L that realize payment channels. Therefore, we need to rely on composition with joint state (as discussed in [9]), where the authors state a stronger version of the composition theorem, called JUC, which accounts for joint state and randomness across protocol sessions.

In order to satisfy the conditions for the JUC theorem to apply, we must first argue that our protocol realizes a stronger ideal functionality $\tilde{\mathcal{F}}_{\mathsf{A^2L}}$, that makes only independent calls to the underlying interfaces. More precisely, we need to argue for each of the previously presented concrete realizations of $\mathcal{F}_{\mathsf{A^2L}}$ that a parallel composition of those protocols realizes the functionality $\tilde{\mathcal{F}}_{\mathsf{A^2L}}$ (with all instances of the protocols sharing the same KGen, but running independently otherwise). We show this in the following lemmas.

Lemma 12. *Let* COM *be a secure commitment scheme, let* NIZK *be a non-interactive zero-knowledge scheme, and let* $\widehat{\mathbb{L}_{\mathsf{Schnorr}}}^{\mathsf{KGen}}$ *be the multi-session extension of the protocol described in Figures 3, 4 and 5, using a shared* KGen *algorithm that realizes* $\mathcal{F}_{\mathsf{KGen}}^{\mathsf{Schnorr}}$*. If Schnorr signatures are strongly existentially unforgeable and Paillier encryption is* IND-CPA *secure, then* $\widehat{\mathbb{L}_{\mathsf{Schnorr}}}^{\mathsf{KGen}}$*, UC-realizes the ideal functionality* $\tilde{\mathcal{F}}_{\mathsf{A^2L}}$ *in the* $(\mathcal{F}_{\mathsf{KGen}}^{\mathsf{Schnorr}}, \mathcal{F}_{\mathsf{anon}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{syn}})$*-hybrid model.*

Proof: It is trivial to see that the $\mathcal{F}_{\mathsf{KGen}}^{\mathsf{Schnorr}}$ functionality itself is stateless, and therefore, consecutive invocations of $\mathcal{F}_{\mathsf{KGen}}^{\mathsf{Schnorr}}$ are indistinguishable from the invocations of fresh instances of the functionality. Thus, for multiple protocols, it is identical to query the same $\mathcal{F}_{\mathsf{KGen}}^{\mathsf{Schnorr}}$ instance or to work on independent copies (note that the same property carries over to protocols realizing this functionality). Consequently, $\widehat{\mathbb{L}_{\mathsf{Schnorr}}}^{\mathsf{KGen}}$ is indistinguishable from the multi-session extension of $\mathbb{L}_{\mathsf{Schnorr}}$ using independent KGen copies that realize $\mathcal{F}_{\mathsf{KGen}}^{\mathsf{Schnorr}}$. Hence, the claim follows from the composition theorem [7] and Theorem 1. ■

Lemma 13. *Let* COM *be a secure commitment scheme, let* NIZK *be a non-interactive zero-knowledge scheme, and let* $\widehat{\mathbb{L}_{\mathsf{ECDSA}}}^{\mathsf{KGen}}$ *be the multi-session extension of the protocol described in Figures 6, 7 and 8, using a shared* KGen *algorithm that realizes* $\mathcal{F}_{\mathsf{KGen}}^{\mathsf{ECDSA}}$*. If ECDSA signatures are strongly existentially unforgeable and Paillier encryption is* ecCPA *secure, then* $\widehat{\mathbb{L}_{\mathsf{ECDSA}}}^{\mathsf{KGen}}$*, UC-realizes the ideal functionality* $\tilde{\mathcal{F}}_{\mathsf{A^2L}}$ *in the* $(\mathcal{F}_{\mathsf{KGen}}^{\mathsf{ECDSA}}, \mathcal{F}_{\mathsf{anon}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{syn}})$*-hybrid model.*

Proof: $\mathcal{F}_{\mathsf{KGen}}^{\mathsf{ECDSA}}$ satisfies the same independence properties as $\mathcal{F}_{\mathsf{KGen}}^{\mathsf{Schnorr}}$, hence, the same argument as for Lemma 12 applies. ■

2) PCH Ideal Functionality: Next, we define an ideal functionality for a PCH, called $\mathcal{F}_{\mathsf{PCH}}$, which can be seen in Figure 9. For simplicity, we do not consider any transaction fees, however, our construction can be trivially extended to include fees.

Data Structures. In order to simplify the exposition, we define a few data structures. Additionally, to ease the notation, we use attributes to access the values of tuples. For example, γ.cid denotes the cid attribute of the tuple γ. The data structures that we require are the following:

- List of promises $\mathcal{P}$, which keeps track of the currently existing promises. The entries in the list have the format $(\mathsf{pid}, \mathsf{lid},$
  $\mathsf{cid}, \nu, P_i)$, where pid is a promise identifier, lid is a lock identifier, cid is the channel identifier, ν is a validity period (expiration time) of the promise, and P_i is the party to whom the promise and lock are given. We note that pid and lid are unique identifiers, and each promise has a validity period defined as $\nu = \Delta + \upsilon$ for a constant value υ.
- List of open channels $\mathcal{C}$, which keeps track of the currently open channels. A channel ς is a tuple defined as

(cid, P_1, P_2), where cid is the channel identifier, and P_1 and P_2 are the parties between whom the channel is established. We consider bidirectional payment channels and, for simplicity, we assume that at any moment there can only be a single open channel between the two parties P_1 and P_2, and one of these parties is always the Tumbler. Hence, we do not consider the payment channels for which Tumbler is not one of the parties involved. This is a natural assumption as PCH involves an intermediary. Apart from the actual tuple values, the channel additionally has the following attributes (as defined in [15]): $\varsigma.\mathsf{parties} = \{\varsigma.P_1, \varsigma.P_2\}$, which defines the two endpoints (parties) of the channel, $\varsigma.\mathsf{balance} : \varsigma.\mathsf{parties} \to \mathbb{R}_{\geq 0}$, which returns the balance of the specified party within the channel, and $\varsigma.\mathsf{other{-}party} : \varsigma.\mathsf{parties} \to \varsigma.\mathsf{parties}$, which is defined as $\varsigma.\mathsf{other{-}party}(\varsigma.P_1) = \varsigma.P_2$ and $\varsigma.\mathsf{other{-}party}(\varsigma.P_2) = \varsigma.P_1$.

3) Discussion: We define here the security and privacy notions of interest for payment hubs.

Balance Security. The system should not be exploited to print new money or steal existing money, even when parties collude. This property was defined in [22]. $\mathcal{F}_{\mathsf{PCH}}$ provides balance security as the only place where the balances are updated is inside the payment operation, and it makes sure that either all the balances are updated or none. Additionally, it assures that the balances are updated only if the correct opening information for a lock is provided by the Tumbler. The atomicity and correctness properties are enough to ensure balance security.

Unlinkability. The intermediary should not learn information that allows it to associate the sender and the receiver of a payment. This is the same property that was previously defined in Section III-C. $\mathcal{F}_{\mathsf{PCH}}$ achieves unlinkability while it uses constant amounts and random but unique identifiers locks, which gets rerandomized before reaching the Tumbler.

D. Trilero: Our System

In the following, we describe the four operations (open channel, close channel, promise and payment) that constitute the core of our system for PCH, which can be seen in Figure 10. Although, we describe open channel and close channel operations here, we do not formally call them in Figure 10, and instead assume that the parties have already established payment channels between themselves before the start of the protocol

Open Channel. The open channel operation generates a new payment channel between the Tumbler and another party P (in our case P is either Alice or Bob). The parties create an initial blockchain deposit with the amount they want to invest for the channel. If the parties have sufficient balance in the blockchain, and the channel opening is mutually authorized, then the operation successfully creates a new payment channel, adds it to a list of open channels, and returns the channel information ς to both parties. Otherwise, it returns $\perp$.

Close Channel. The close channel operation is run by parties that share an open payment channel. The operation checks whether the specified channel is still open, and whether there are still unexpired promises tied to this channel. In case such promises exist, it removes them from the list of currently valid promises. Next, it updates the blockchain balance of each party according to their channel balance, and sends $\top$ to both parties.

Promise. The promise operation returns a promise Π from Tumbler to Bob, conditioned that Tumbler and Bob share an open payment channel, and Tumbler has sufficient balance to fulfill the promise. If the conditions are not satisfied it returns $\perp$.

Payment. The payment operation transfers amt coins from Tumbler to Bob, and from a party Alice to Tumbler. The operation makes sure that the promise has not expired, the parties have enough balance to fulfill the transactions, and that Tumbler provides a valid opening to the lock corresponding to the given promise. If all these conditions are satisfied, then it updates the balances of the parties, and returns $\top$. Otherwise, the balances are not modified, and it returns $\perp$.

1) Security Analysis: In the following we argue that the system as described in Figure 10, UC-realizes the functionality $\mathcal{F}_{\mathsf{PCH}}$ as defined in Figure 9.

Theorem 3. *The system described in Figure 10, UC-realizes $\mathcal{F}_{\mathsf{PCH}}$ (as defined in Figure 9) in the $(\mathcal{F}_{\mathsf{A^2L}}, \mathcal{F}_{\mathsf{B}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{syn}})$-hybrid model.*

Proof: The proof consists of the observation that the ideal functionality $\mathcal{F}_{\mathsf{A^2L}}$ enforces balance security and unlinkability properties of a PCH (as defined in Section C3). Balance security is guaranteed due to the atomicity of $\mathcal{F}_{\mathsf{A^2L}}$, meaning either all the balances are updated or none of them. This ensures that no party loses or gains more than it should. As was discussed in Section III-C, $\mathcal{F}_{\mathsf{A^2L}}$ satisfies the unlinkability property, hence, the same argument for unlinkability applies here too. Also, note that the only information that is sent outside of $\mathcal{F}_{\mathsf{A^2L}}$ consists of amounts and timeouts, and these values are chosen exactly as described in $\mathcal{F}_{\mathsf{PCH}}$. Furthermore, it is sufficient to argue about the individual copies of $\mathcal{F}_{\mathsf{A^2L}}$ in isolation by the JUC theorem [9]. As was shown in Lemmas 12 and 13, the multi-session extended ideal functionality $\tilde{\mathcal{F}}_{\mathsf{A^2L}}$ is realized by our instantiations, and therefore, the JUC theorem allows us to complete the analysis assuming independent copies of $\mathcal{F}_{\mathsf{A^2L}}$ running in parallel. ■

E. A^2L for Monero

In the following, we present how to construct $\mathrm{A^2L}$ for Monero. Our construction makes use of the recent ring signature scheme, called DLSAG, introduced by Moreno-Sanchez et al. [38]. The promise and payment protocols can be seen in Figure 11 and Figure 12. The values inside brackets denote the shares. For instance, $[s_0']_B$ and $[s_0']_T$ denote the share of Alice and Tumbler, respectively, for the value s_0. We consider two hash functions: (i) H_s takes as input a bitstring and outputs a scalar (i.e., $H_s \colon \{0,1\}^* \to \mathbb{Z}_q$); (ii) H_p takes as input a bitstring and outputs an element of $\mathbb{G}$ (i.e., $H_p \colon \{0,1\}^* \to \mathbb{G}$). k and k' are bitstrings chosen uniformly at random from $\{0,1\}^\lambda$. Q and Q' denote parts of the dual key created when opening payment channels between Bob/Tumbler and Alice/Tumbler, respectively. For more information about how to open payment channels and make conditional payments in Monero we refer the reader to [38].

For brevity we write $\mathcal{F}$ for $\mathcal{F}_{\mathsf{PCH}}$, and denote Tumbler as T. We assume that the channel and promise identifiers are unique and generated at random by the ideal functionality. Additionally, there exists a lock randomizer function rand, and all the promises use a constant amount (amt).

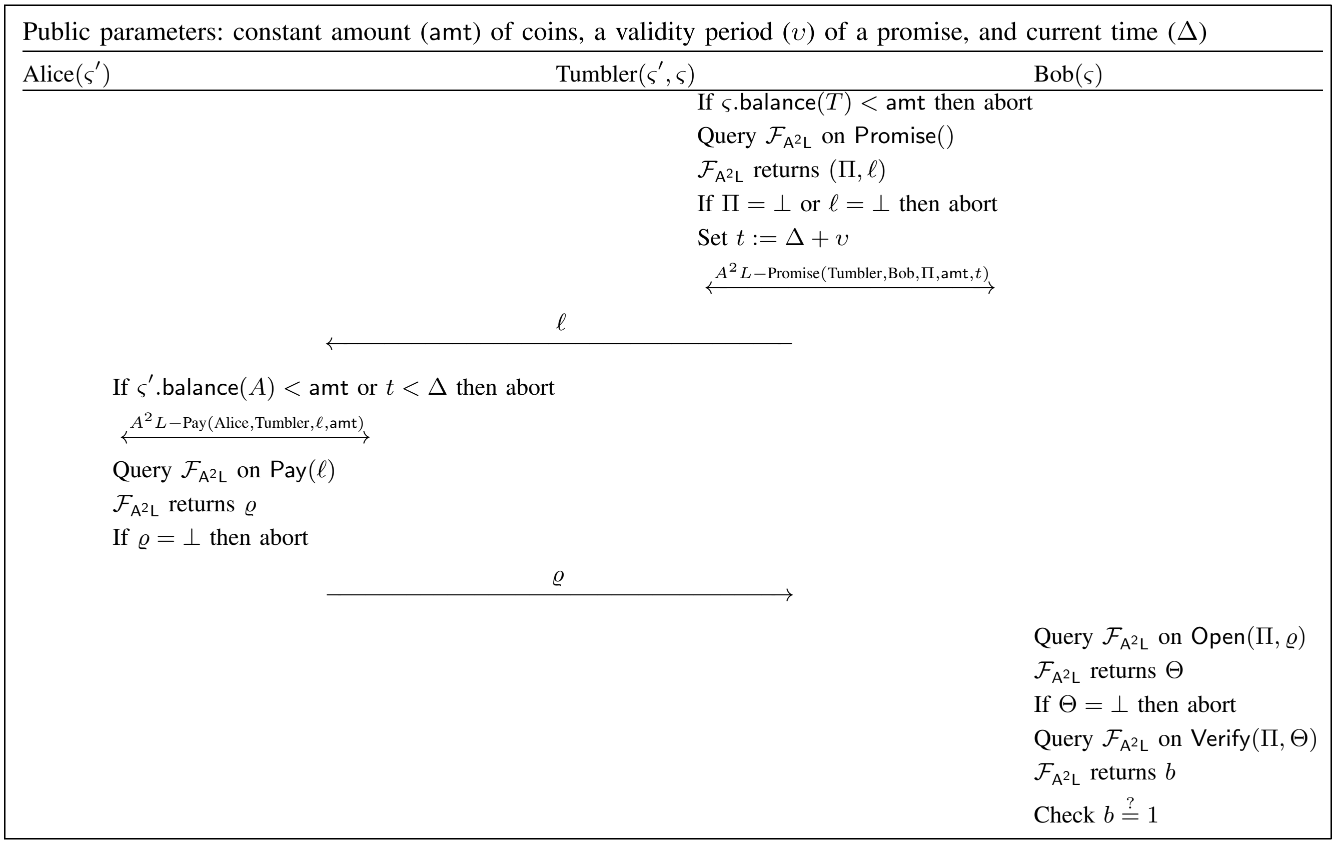
Open Channel: On input $(\mathsf{pc{-}open}, \mathsf{sid}, \varsigma)$, from a party P with $\varsigma.\mathsf{balance}(P)$ coins, where ς is the channel, $P \in \{A, B\}$, $P \in \varsigma.\mathsf{parties}$, and $\varsigma.\mathsf{other{-}party}(P) = T$, $\mathcal{F}$ checks whether $(\mathsf{sid}, \varsigma)$ is present in $\mathcal{C}$. If it is present, then $\mathcal{F}$ sends $(\mathsf{pc{-}exists}, \mathsf{sid}, \perp)$ to P, otherwise it sends $(\mathsf{pc{-}request}, \mathsf{sid}, \varsigma)$ to T, who can either abort or authorize the operation. In the latter case, $\mathcal{F}$ receives $(\mathsf{pc{-}open}, \mathsf{sid}, \varsigma')$ from T with $\varsigma'.\mathsf{balance}(T)$ coins, and checks whether $\varsigma = \varsigma'$, and $\forall P' \in \varsigma.\mathsf{parties}$, such that $\mathcal{B}[P'] \geq \varsigma.\mathsf{balance}(P')$ using $\mathcal{F}_{\mathsf{B}}$. If the checks pass, $\mathcal{F}$ sends $(\mathsf{remove}, \varsigma.P_1, \varsigma.\mathsf{balance}(\varsigma.P_1))$ and $(\mathsf{remove}, \varsigma.P_2, \varsigma.\mathsf{balance}(\varsigma.P_2))$ to $\mathcal{F}_{\mathsf{B}}$. Lastly, $\mathcal{F}$ sends $(\mathsf{pc{-}opened}, \mathsf{sid}, \varsigma)$ to $\varsigma.P_1$ and $\varsigma.P_2$. Otherwise, channel opening fails and $\mathcal{F}$ sends $(\mathsf{pc{-}failed}, \mathsf{sid}, \perp)$ to parties in $\varsigma.\mathsf{parties}$.

Promise: On input $(\mathsf{promise{-}request}, \mathsf{sid}, \varsigma)$ from a party P, such that $P \in \varsigma.\mathsf{parties}$ and $\varsigma.\mathsf{other{-}party}(P) = T$, $\mathcal{F}$ sends $(\mathsf{create{-}promise}, \mathsf{sid}, \varsigma)$ to T, who can either abort or authorize the operation. In the former case, $\mathcal{F}$ receives $(\mathsf{promise}, \mathsf{sid}, \perp)$ from T, and sends $(\mathsf{promise{-}failed}, \mathsf{sid}, \perp)$ to P. In the latter case, $\mathcal{F}$ receives $(\mathsf{promise}, \mathsf{sid}, \top)$ from T, and checks whether $\varsigma.\mathsf{balance}(T) \geq \mathsf{amt}$. If the condition is not satisfied it sends $(\mathsf{promise{-}failed}, \mathsf{sid}, \perp)$ to both parties in $\varsigma.\mathsf{parties}$. Otherwise, it stores $\Pi = (\mathsf{pid}, \mathsf{lid}, \mathsf{cid}, \nu, P)$ in $\mathcal{P}$, for a random but unique $\Pi.\mathsf{pid}$ and $\Pi.\mathsf{lid}$, a channel identifier $\Pi.\mathsf{cid} = \varsigma.\mathsf{cid}$, a validity period $\Pi.\nu$, and sends $(\mathsf{promise{-}created}, \mathsf{sid}, \Pi)$ to P.

Payment: On input $(\mathsf{pay{-}request}, \mathsf{sid}', \mathsf{lid}, \varsigma')$ from P', such that $P' \in \varsigma'.\mathsf{parties}$, and $\varsigma'.\mathsf{other{-}party}(P') = T$, $\mathcal{F}$ sends $(\mathsf{receive{-}payment}, \mathsf{sid}', P', \mathsf{rand}(\mathsf{lid}), \varsigma')$ to T, who can either abort or authorize the operation. In the former case, $\mathcal{F}$ receives $(\mathsf{pay}, \mathsf{sid}', \perp)$ from T and sends $(\mathsf{pay{-}failed}, \mathsf{sid}', \perp)$ to P'. In the latter case, $\mathcal{F}$ receives $(\mathsf{pay}, \mathsf{sid}', \varrho)$ from T. At this point, $\mathcal{F}$ checks the following conditions: 1) there is an entry $\Pi \in \mathcal{P}$, such that $\Pi.\mathsf{lid} = \mathsf{lid}$ and $\Pi.\nu \geq \Delta$ (i.e., promise has not expired), 2) ϱ is a valid opening of $\Pi.\mathsf{lid}$, and 3) $\varsigma'.(P') \geq \mathsf{amt}$. If the conditions are satisfied, then $\mathcal{F}$ updates the balances of P' and T in channel ς' as $\varsigma'.(P') \mathrel{-}= \mathsf{amt}$ and $\varsigma'.(T) \mathrel{+}= \mathsf{amt}$, respectively. Also, updates the balance of $\Pi.P$ and T in channel ς as $\varsigma.(P) \mathrel{+}= \mathsf{amt}$ and $\varsigma.(T) \mathrel{-}= \mathsf{amt}$, respectively, where $\varsigma.\mathsf{cid} = \Pi.\mathsf{cid}$. Lastly, $\mathcal{F}$ removes the entry Π from $\mathcal{P}$, and sends $(\mathsf{paid}, \mathsf{sid}', \top)$ to P'. Otherwise, if any of the conditions fails, then $\mathcal{F}$ sends $(\mathsf{pay{-}failed}, \mathsf{sid}', \perp)$ to parties in $\varsigma'.\mathsf{parties}$.

Close Channel: On input $(\mathsf{pc{-}close}, \mathsf{sid}, \mathsf{cid}')$ from a party P, $\mathcal{F}$ checks whether there exists a payment channel $\varsigma \in \mathcal{C}$, such that $\varsigma.\mathsf{cid} = \mathsf{cid}'$ and $P \in \varsigma.\mathsf{parties}$. If no such channel exists, $\mathcal{F}$ ignores the message. Otherwise, $\mathcal{F}$ checks whether there exists a $\Pi \in \mathcal{P}$, such that $\Pi.\mathsf{cid} = \varsigma.\mathsf{cid}$ and $\Pi.\nu \geq \Delta$ (i.e., a promise has not expired). If such a Π exists, then $\mathcal{F}$ removes Π from $\mathcal{P}$. Then, $\mathcal{F}$ sends $(\mathsf{add}, \varsigma.P_1, \varsigma.\mathsf{balance}(P_1))$ and $(\mathsf{add}, \varsigma.P_2, \varsigma.\mathsf{balance}(P_2))$ to $\mathcal{F}_{\mathsf{B}}$. Lastly, $\mathcal{F}$ removes ς from $\mathcal{C}$, and sends $(\mathsf{pc{-}closed}, \top)$ to parties in $\varsigma.\mathsf{parties}$.

Fig. 9: Ideal functionality $\mathcal{F}_{\mathsf{PCH}}$ in the $(\mathcal{F}_{\mathsf{B}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{syn}})$-hybrid model



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

Alice(ς') Tumbler(ς', ς) Bob(ς)

If $\varsigma.\mathsf{balance}(T) < \mathsf{amt}$ then abort
Query $\mathcal{F}_{\mathsf{A^2L}}$ on $\mathsf{Promise}()$
$\mathcal{F}_{\mathsf{A^2L}}$ returns (Π, ℓ)
If $\Pi = \perp$ or $\ell = \perp$ then abort
Set $t := \Delta + \upsilon$

$\xleftrightarrow{A^2L-\text{Promise}(\text{Tumbler},\text{Bob},\Pi,\mathsf{amt},t)}$

$\xleftarrow{\ \ell\ }$

If $\varsigma'.\mathsf{balance}(A) < \mathsf{amt}$ or $t < \Delta$ then abort

$\xleftrightarrow{A^2L-\text{Pay}(\text{Alice},\text{Tumbler},\ell,\mathsf{amt})}$

Query $\mathcal{F}_{\mathsf{A^2L}}$ on $\mathsf{Pay}(\ell)$
$\mathcal{F}_{\mathsf{A^2L}}$ returns ϱ
If $\varrho = \perp$ then abort

$\xrightarrow{\ \varrho\ }$

Query $\mathcal{F}_{\mathsf{A^2L}}$ on $\mathsf{Open}(\Pi, \varrho)$
$\mathcal{F}_{\mathsf{A^2L}}$ returns Θ
If $\Theta = \perp$ then abort
Query $\mathcal{F}_{\mathsf{A^2L}}$ on $\mathsf{Verify}(\Pi, \Theta)$
$\mathcal{F}_{\mathsf{A^2L}}$ returns b
Check $b \overset{?}{=} 1$

Fig. 10: Trilero protocol in the $(\mathcal{F}_{\mathsf{A^2L}}, \mathcal{F}_{\mathsf{B}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{syn}})$-hybrid model.

Public parameters: $\mathbb{G}, g, q$, a ring $\vec{\mathsf{pk}} := ((\mathsf{pk}_{1,0}, \mathsf{pk}_{1,1}), (\mathsf{pk}_{n-1,0}, \mathsf{pk}_{n-1,1}), (\mathsf{pk}_{TB,0}, \mathsf{pk}_{TB,1}))$, a message m'

$\mathsf{Promise}_T(\mathsf{sk}_T, [\mathsf{pk}_{TB}]_T)$ — $\mathsf{Promise}_B(\mathsf{sk}_B, [\mathsf{pk}_{TB}]_B)$

```
$\vec{s} := ([s'_0]_T, s_1, \ldots, s_{n-1}) \leftarrow_\$ \mathbb{Z}_q$
$\mathcal{J}_T \leftarrow (Q')^{[\mathsf{pk}_{TB}]_T \cdot k'}; \hat{\mathcal{J}}_T \leftarrow (Q')^{[s'_0]_T \cdot k'}$
$R_T \leftarrow g^{[s'_0]_T}; \alpha \leftarrow_\$ \mathbb{Z}_q; c_a \leftarrow \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_T, \alpha)$
$A \leftarrow g^{\alpha}; A^* \leftarrow \mathsf{pk}_{TB,1}^{\alpha \cdot k'}$
$\pi_a \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists \alpha \mid A = g^{\alpha} \wedge A^* = (\mathsf{pk}_{TB,1}^{k'})^{\alpha}, \alpha\})$
$\pi_T \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists [s'_0]_T \mid R_T = g^{[s'_0]_T}$
        $\wedge \hat{\mathcal{J}}_T = (Q'^{k'})^{[s'_0]_T}\}, [s'_0]_T)$
```

$$\xrightarrow{\vec{s}, \mathcal{J}_T, \hat{\mathcal{J}}_T, R_T, \pi_T, \pi_a, c_a, A, A^*}$$

```
If $\mathsf{V}_{\mathsf{NIZK}}(\pi_T, (R_T, \hat{\mathcal{J}}_T)) \neq 1$
 $\vee \mathsf{V}_{\mathsf{NIZK}}(\pi_a, (A, A^*)) \neq 1$ then abort
$[s'_0]_B \leftarrow_\$ \mathbb{Z}_q; R_B \leftarrow g^{[s'_0]_B}$
$\mathcal{J}_B \leftarrow (Q')^{[\mathsf{pk}_{TB}]_B \cdot k'}; \hat{\mathcal{J}}_B \leftarrow (Q')^{[s'_0]_B \cdot k'}$
$\mathcal{J} \leftarrow \mathcal{J}_T \cdot \mathcal{J}_B$
$\pi_B \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists [s'_0]_B \mid R_B = g^{[s'_0]_B}$
        $\wedge \hat{\mathcal{J}}_B = (Q'^{k'})^{[s'_0]_B}\}, [s'_0]_B)$
```

$$\xleftarrow{\mathcal{J}_B, \hat{\mathcal{J}}_B, R_B, \pi_B}$$

```
If $\mathsf{V}_{\mathsf{NIZK}}(\pi_B, (R_B, \hat{\mathcal{J}}_B)) \neq 1$ then abort
$\mathcal{J} \leftarrow \mathcal{J}_T \cdot \mathcal{J}_B$
$h_0 \leftarrow H_s(m' \| g^{[s'_0]_T} \cdot g^{[s'_0]_B} \cdot A \| \hat{\mathcal{J}}_T \cdot \hat{\mathcal{J}}_B \cdot A^*)$
$\forall i \in \{1, \ldots, n-1\}$ :
    $L_i \leftarrow g^{s_i} \cdot (\mathsf{pk}_i)^{h_{i-1}}$
    $R_i \leftarrow H_p(\mathsf{pk}_i)^{s_i} \cdot \mathcal{J}^{h_{i-1}}$
    $h_i \leftarrow H_s(m' \| L_i \| R_i)$
$[s_0]_T \leftarrow [s'_0]_T - h_{n-1} \cdot \mathsf{sk}_T$
$[\sigma]_T := ([s_0]_T, s_1, \ldots, s_{n-1}, h_0, \mathcal{J}_T)$
```

$$\xrightarrow{[\sigma]_T}$$

```
$h_0 \leftarrow H_s(m' \| g^{[s'_0]_T} \cdot g^{[s'_0]_B} \cdot A \| \hat{\mathcal{J}}_T \cdot \hat{\mathcal{J}}_B \cdot A^*)$
$\forall i \in \{1, \ldots, n-1\}$ :
    $L_i \leftarrow g^{s_i} \cdot (\mathsf{pk}_i)^{h_{i-1}}$
    $R_i \leftarrow H_p(\mathsf{pk}_i)^{s_i} \cdot \mathcal{J}^{h_{i-1}}$
    $h_i \leftarrow H_s(m' \| L_i \| R_i)$
$[s_0]_B \leftarrow [s'_0]_B - h_{n-1} \cdot \mathsf{sk}_B$
$[\sigma]_B := ([s_0]_B, s_1, \ldots, s_{n-1}, h_0, \mathcal{J}_B)$
$\beta \leftarrow_\$ \mathbb{Z}_q; c_{a'} \leftarrow c_a \cdot \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_T, \beta)$
$A' \leftarrow A \cdot g^{\beta}$
```

$$\xleftarrow{[\sigma]_B}$$

```
Send $\ell := (A', c_{a'})$ to Alice
$\sigma' := ([s_0]_T + [s_0]_B, s_1, \ldots, s_{n-1}, h_0, \mathcal{J})$
```

return $\sigma := ([s_0]_T + [s_0]_B + \alpha, s_1, \ldots, s_{n-1}, h_0, \mathcal{J})$ — **return** $(\Pi := (\beta, (\vec{\mathsf{pk}}, m', \sigma'), \ell)$

Fig. 11: Promise protocol of A^{2}L for Monero

```
Public parameters: $\mathbb{G}, g, q$, a ring $\vec{\mathsf{pk}} := ((\mathsf{pk}_{1,0}, \mathsf{pk}_{1,1}), (\mathsf{pk}_{n-1,0}, \mathsf{pk}_{n-1,1}), (\mathsf{pk}_{AT,0}, \mathsf{pk}_{AT,1}))$, a message $m$

$\mathsf{Pay}_A(\mathsf{sk}_A, [\mathsf{pk}_{AT}]_A, \ell := (A', c'_a))$        $\mathsf{Pay}_T(\mathsf{sk}_T, [\mathsf{pk}_{AT}]_T)$

                                                        $\vec{s} := ([s'_0]_T, s_1, \ldots, s_{n-1}) \leftarrow\!\!\$\, \mathbb{Z}_q$
                                                        $\mathcal{J}_T \leftarrow Q^{[\mathsf{pk}_{AT}]_T \cdot k}; \hat{\mathcal{J}}_T \leftarrow Q^{[s'_0]_T \cdot k}$
                                                        $R_T \leftarrow g^{[s'_0]_T}$
                                                        $\pi_T \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists [s'_0]_T \mid R_T = g^{[s'_0]_T}$
                                                            $\wedge\, \hat{\mathcal{J}}_T = (Q^k)^{[s'_0]_T}\}, [s'_0]_T)$

                            $\xleftarrow{\vec{s}, \mathcal{J}_T, \hat{\mathcal{J}}_T, R_T, \pi_T}$

If $\mathsf{V}_{\mathsf{NIZK}}(\pi_T, (R_T, \hat{\mathcal{J}}_T)) \neq 1$ then abort
$[s'_0]_A \leftarrow\!\!\$\, \mathbb{Z}_q; R_A \leftarrow g^{[s'_0]_A}$
$\mathcal{J}_A \leftarrow Q^{[\mathsf{pk}_{AT}]_A \cdot k}; \hat{\mathcal{J}}_A \leftarrow Q^{[s'_0]_A \cdot k}$
$\pi_A \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists [s'_0]_A \mid R_A = g^{[s'_0]_A}$
    $\wedge\, \hat{\mathcal{J}}_A = (Q^k)^{[s'_0]_A}\}, [s'_0]_A)$
$\tau \leftarrow\!\!\$\, \mathbb{Z}_q; c_{a''} \leftarrow c_{a'} \cdot \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}_T, \tau)$

                            $\xrightarrow{\mathcal{J}_A, \hat{\mathcal{J}}_A, R_A, \pi_A, c''_a}$

                                                        If $\mathsf{V}_{\mathsf{NIZK}}(\pi_A, (R_A, \hat{\mathcal{J}}_A)) \neq 1$ then abort
                                                        $\gamma \leftarrow \mathsf{Dec}_{\mathsf{HE}}(c''_a); A'' \leftarrow g^{\gamma}; (A^*)'' \leftarrow pk_{AT,1}^{\gamma \cdot k}$
                                                        $\mathcal{J} \leftarrow \mathcal{J}_A \cdot \mathcal{J}_T$
                                                        $h_0 \leftarrow H_s(m \| g^{[s'_0]_T} \cdot g^{[s'_0]_A} \cdot A'' \| \hat{\mathcal{J}}_T \cdot \hat{\mathcal{J}}_A \cdot (A^*)'')$
                                                        $\forall i \in \{1, \ldots, n-1\}:$
                                                          $L_i \leftarrow g^{s_i} \cdot (\mathsf{pk}_i)^{h_{i-1}}$
                                                          $R_i \leftarrow H_p(\mathsf{pk}_i)^{s_i} \cdot \mathcal{J}^{h_{i-1}}$
                                                          $h_i \leftarrow H_s(m \| L_i \| R_i)$
                                                        $[s_0]_T \leftarrow [s'_0]_T - h_{n-1} \cdot \mathsf{sk}_T$
                                                        $[\sigma]_T := ([s_0]_T, s_1, \ldots, s_{n-1}, h_0, \mathcal{J}_T)$

                            $\xleftarrow{A'', [\sigma]_T}$

If $(A') \cdot g^{\tau} \neq A''$ then abort
$h_0 \leftarrow H_s(m \| g^{[s'_0]_T} \cdot g^{[s'_0]_A} \cdot A'' \| \hat{\mathcal{J}}_T \cdot \hat{\mathcal{J}}_A \cdot (A^*)'')$
$\mathcal{J} \leftarrow \mathcal{J}_A \cdot \mathcal{J}_T$
$\forall i \in \{1, \ldots, n-1\}:$
  $L_i \leftarrow g^{s_i} \cdot (\mathsf{pk}_i)^{h_{i-1}}$
  $R_i \leftarrow H_p(\mathsf{pk}_i)^{s_i} \cdot \mathcal{J}^{h_{i-1}}$
  $h_i \leftarrow H_s(m \| L_i \| R_i)$
$[s_0]_A \leftarrow [s'_0]_A - h_{n-1} \cdot \mathsf{sk}_A$
$[\sigma]_A := ([s_0]_A, s_1, \ldots, s_{n-1}, h_0, \mathcal{J}_A)$

                            $\xrightarrow{[\sigma]_A}$

                                                        $\sigma := (s_0 = [s_0]_A + [s_0]_T + \gamma, s_1, \ldots, s_{n-1}, h_0, \mathcal{J})$
                                                        If verification of $\sigma$ fails then abort
                                                        Else publish signature $\sigma$

$\gamma \leftarrow s_0 - ([s_0]_A + [s_0]_T); \bar{\alpha} \leftarrow \gamma - \tau$
Send $\bar{\alpha}$ to Bob
**return** $\bar{\alpha}$                                   **return** $\top$
```

Fig. 12: Payment protocol of A^{2}L for Monero