Universal Atomic Swaps: Secure Exchange of Coins Across All Blockchains

Sri AravindaKrishnan Thyagarajan
Carnegie Mellon University
t.srikrishnan@gmail.com

Giulio Malavolta
Max Planck Institute for Security and Privacy
giulio.malavolta@hotmail.it

Pedro Moreno-Sanchez
IMDEA Software Institute
pedro.moreno@imdea.org

***Abstract*—Trading goods lies at the backbone of the modern economy and the recent advent of cryptocurrencies has opened the door for trading decentralized (digital) assets: A large fraction of the value of cryptocurrencies comes from the inter-currency exchange and trading, which has been arguably the most successful application of decentralized money. The security issues observed with centralized, custodial cryptocurrency exchanges have motivated the design of *atomic swaps*, a protocol for coin exchanges between any two users. Yet, somewhat surprisingly, no atomic swap protocol exists that simultaneously satisfies the following simple but desired properties: (i) *non-custodial*, departing from a third party trusted holding the coins from users during the exchange; (ii) *universal*, that is, compatible with all (current and future) cryptocurrencies; (iii) *multi-asset*, supporting the exchange of multiple coins in a single atomic swap.**

From a theoretical standpoint, in this work we show a generic protocol to securely swap n coins from any (possible multiple) currencies for $\tilde{n}$ coins of any other currencies, for any n and $\tilde{n}$. We do not require *any* custom scripting language supported by the corresponding blockchains, besides the bare minimum ability to verify signatures on transactions. For the special case when the blockchains use ECDSA or Schnorr signatures, we design a practically efficient protocol based on adaptor signatures and time-lock puzzles. As a byproduct of our approach, atomic swaps transactions no longer include custom scripts and are identical to standard one-to-one transactions. We also show that our protocol naturally generalizes to any cycle of users, i.e., atomic swaps with more than two participants. To demonstrate the practicality of our approach, we have evaluated a prototypical implementation of our protocol for Schnorr/ECDSA signatures and observed that an atomic swap requires below one second on commodity machines. Even on blockchains with expressive smart contract support (e.g., Ethereum), our approach reduces the on-chain cost both in terms of transaction size and gas cost.

***Index Terms*—Atomic Swaps, Adaptor Signatures, Blockchain**

I. Introduction

Blockchains coexisting today are no longer isolated siloes, but their value rather comes from the exchange of assets across them. Cross-chain communication [1] is thus found a critical component on cryptocurrency transfers and exchanges. In fact, trading is arguably among the top applications in the cryptocurrency landscape and numerous competing interoperability projects, attempting to unite otherwise independent blockchains, have been deployed in practice creating a multi-billion dollar industry [2]–[5].

On a technical level, a cross-chain swap involves two ledgers $\mathbb{B}_A$, $\mathbb{B}_B$ and two users Alice and Bob. Alice holds α assets on $\mathbb{B}_A$ while Bob holds β assets on $\mathbb{B}_B$. The *atomic swap* problem consists in ensuring that Alice transfers α to Bob in $\mathbb{B}_A$ if and only if Bob transfers β to Alice in $\mathbb{B}_B$. We then say that a solution is *atomic* if it ends in either of the following outcomes: (i) Bob owns α in $\mathbb{B}_A$ and Alice owns β in $\mathbb{B}_B$ (i.e., asset swap); or (ii) Alice owns α in $\mathbb{B}_A$ and Bob owns β in $\mathbb{B}_B$ (i.e., asset refund). Moreover, an atomic swap must preserve *fungibility*, that is, an observer (other than Alice and Bob) of the ledger should not distinguish a transfer executed as part of an atomic swap from a standard asset transfer in such ledger.

Despite being a fundamental problem in the cryptocurrencies landscape, the state of the art for atomic swaps protocols is rather chaotic: Atomic swaps protocols are typically tailored to the characteristics offered by a restricted set of currencies (e.g., Turing-complete scripting languages) [6]–[9], require the existence of a third ledger to coordinate the swap [5], [10], require additional trust assumptions such as trusted hardware [11], or require the presence of a trusted third party like an online exchange. As a result, adding a new cryptocurrency in the market most likely requires one to design a new ad-hoc atomic swap protocol. Even among existing cryptocurrencies, atomic swaps (and consequently secure exchange) are limited to a handful of combinations of cryptocurrencies, or require one to accept strong trust assumptions.

On a conceptual level, such atomic swap of coins across currencies enables the most basic form of inter-chain connectivity, irrespective of the application. In this work, we ask whether one can obtain a *universal* solution for such atomic swaps: a *Swiss Army knife* protocol that works for all cryptocurrencies with minimal support from the blockchain. In order to answer this question, we first analyze existing approaches and argue why they fall short of our quest for a universal solution.

A. Where Existing Approaches Fall Short

We provide a summary of the comparison in Table I and we elaborate on each approach individually in the following.

Third Ledger. Migration protocols like cryptocurrency-backed assets or side-chains, mimic the atomic swap functionality by requiring that Alice and Bob migrate their assets from (perhaps language-restricted) ledgers $\mathbb{B}_A$ and $\mathbb{B}_B$ into $\mathbb{B}_C$ with a more expressive scripting language (like Ethereum) [10]. Once the funds are in $\mathbb{B}_C$, they are swapped and sent back to the initial ledgers $\mathbb{B}_A$ and $\mathbb{B}_B$. Such an approach has the following drawbacks: (i) the swap protocol imposes transaction and cost overhead not only to $\mathbb{B}_A$ and $\mathbb{B}_B$ but also $\mathbb{B}_C$; and (ii) it is not an universal solution as it potentially requires a different asset migration as well as atomic swap protocols for each ledger that plays the role of $\mathbb{B}_C$ in the aforementioned example. Even in the unlikely case that everybody agrees to use $\mathbb{B}_C$ for their atomic swaps, we need to enhance every blockchain with the capability of migrating assets to/from $\mathbb{B}_C$.

Use of Trusted Hardware. An alternative approach would be to defer the atomic swap functionality to a trusted execution environment (TEE) [11]. Besides the fact that this solution requires all users to have such a TEE (which is unrealistic), recent works have shown serious TEE vulnerabilities [12], [13].

Hash Timelock Contracts. The closest solution to a universal protocol for atomic swaps (which is implemented in the large majority of trustless exchange protocols) relies on hash-time lock contract (HTLC), an excerpt of script that implements the following logic: On input a tuple (α, h, t, A, B), where α denotes the assets to be transferred, h is a hash value, t denotes a certain timeout, A and B represent two users in the ledger, the HTLC contract transfers α to B if it is invoked before time t on input a value r such that $h = \mathsf{H}(r)$, where $\mathsf{H}(\cdot)$ is a cryptographic hash function. If the HTLC contract is invoked after time t, it transfers the α assets to A unconditionally.

Using HTLCs as building block, an atomic cross-chain swap where Alice exchanges α assets in $\mathbb{B}_A$ for β assets from Bob in $\mathbb{B}_B$, is then realized as follows: Alice chooses r, computes $h = \mathsf{H}(r)$, transfers α into an HTLC(α, h, t, A, B) in $\mathbb{B}_A$ and sends h, t to Bob. Note that at this point Bob cannot claim the HTLC because r is only known to Alice. Instead, Bob finishes the setup of the exchange by choosing a time $t' < t$ and transferring his β assets into an HTLC(β, h, t', B, A) in $\mathbb{B}_B$. The atomicity of the swap is enforced by the logic of the HTLC: (Case i) Alice claims the HTLC in $\mathbb{B}_B$, effectively revealing r to Bob (and anyone observing $\mathbb{B}_B$) before t'. Bob can then use r to claim the HTLC in $\mathbb{B}_A$; (Case ii) Alice does not claim the HTLC in $\mathbb{B}_B$ before time t', then she does not reveal r, ensuring that Bob cannot claim the HTLC in $\mathbb{B}_A$.

HTLC-based cross-chain swaps are deployed in practice [14], [15] and have a wide range of applications [18]–[20], However, they incur high execution costs (e.g., gas in Ethereum), as well as large transaction sizes (due to large scripts), and they have inherent challenges that reduce their utility, which we summarize next.

(1) Compatibility of the Hash Function. Both ledgers must support compatible hash functions within their scripting language. In fact, they both should support *the same* hash function and each ledger must use the same number of bits to represent it, otherwise atomicity does no longer hold [21] since one ledger may not allow pre-images of a large enough size. Apart from atomicity, the use of the same h value at two ledgers raises a privacy concern, as an observer can link both HTLC as part of the same swap. Finally, a perhaps more fundamental issue, is that there exist several cryptocurrencies such as Monero [22], Mimblewimble [23], Ripple [24], Stellar [25] or Zcash [26] (shielded addresses) that do not support the computation of the HTLC contract in their scripting language.

TABLE I: Comparison among different approaches

Approach	**Required Functionality**	**Fungibility**	n-to-$\tilde{n}$ **support**
TEE [11]	Any digital signature	Yes	No
HTLC [14], [15]	Hashing and Timelock	No	No
Smart contracts [10], [16], [17]	Expressive script	No	Yes
Generic (**This work**)	Any digital signature	Yes	Yes
Tailored (**This work**)	ECDSA/Schnorr	Yes	Yes

(2) Presence of the Timelock. Another issue with the HTLC approach is that both ledgers must support the timelock functionality in their scripting language, in other words, the possibility to lock the spending of coins by a certain user until a certain time (e.g., defined as block height) is reached. Adding this timelock functionality is at odds with privacy because (i) if implemented naively, it makes time-locked transfers easier to distinguish from those transfers without time restrictions [27]; (ii) including it conflicts with other privacy operations already available at the ledger [28]; (iii) if possible to include and implemented in a privacy-preserving manner, it adds a non-trivial overhead to the computation and storage overhead of the ledger [27], [29]. In this state of affairs, there exist cryptocurrencies that have been created with privacy by design and avoid the use of timelocked assets as a design principle [30].

(3) Single-Asset Swap. The swap is restricted to only two HTLC, one per ledger, and thus to the exchange of α assets in $\mathbb{B}_A$ and β assets in $\mathbb{B}_B$ (e.g., α bitcoins by β ethers). However, given the huge differences in value in current cryptocurrencies (e.g., 1 bitcoin is worth 100× more than 1 ether at the time of writing), current atomic swaps are restricted to small values of α (or β) to be able to match the exchange offer. However, in practice there exist users (e.g., market makers or exchanges) that have a varied portfolio and hold assets at different ledgers, who could potentially use several of their assets to match a swap offer, if multi-asset swaps were available. For instance, multi-asset swaps for the first time would allow Alice to use coins she owns at Ethereum, Monero and Ripple to match an exchange offer from Bob of 1 Bitcoin through a single atomic swap operation.

B. *Our contributions*

Our goal is to design a universal atomic swap protocol, that does not make any assumption on any specific features of the blockchain, or scripting functionality of the currencies and only assumes the (arguably minimal) ability to verify signatures on transactions. Besides establishing an important feasibility result, such a protocol immediately enables secure exchange protocols across *all combination of cryptocurrencies*, excluded by current ad-hoc solutions. As a byproduct of such a generality, such atomic swap transactions are *identical* to standard one-to-one transactions, thereby increasing the fungibility of the swapped coins. The contributions of this work can be summarized as follows.

(1) Universal Swaps. We establish the theoretical foundations of universal atomic swaps by presenting the first protocol to securely exchange *any* n coins (possibly) from different currencies for *any* $\tilde{n}$ coins from another (possibly any disjoint) set of currencies. Specifically, the protocol runs in polynomial time for any polynomial n and $\tilde{n}$ and can handle *any* currency that only offers signature verification script for authenticating transactions, and therefore is *universal*. We assume the existence of a (UC-secure) general-purpose 2-party computation (2PC) protocol [31] and the existence of verifiable timed signatures [32]–[34]. We overview our construction in Section II and give the formal construction in Appendix H.

We stress that we present this protocol only as a conceptual contribution. Using general purpose cryptographic tools (such as 2PC) would likely result in an impractical protocol. Yet, such protocol will serve as the general outline to construct more efficient schemes tailored to specific signature schemes, as we show with our next contribution.

(2) Efficient Protocol for Schnorr/ECDSA. For the special case of Schnorr/ECDSA signatures, we design a special-purpose n-to-$\tilde{n}$ atomic swap protocol (Section V), which is optimized in several aspects to achieve high practical efficiency. Our protocol supports any (crypto)currency that uses Schnorr or ECSDA signatures to sign transactions, *regardless* of the *elliptic curve* used to implement such signature schemes. This captures many existing cryptocurrencies, including those with the highest market capitalization such as Bitcoin, Ethereum, Ripple, or Stellar. Our techniques can also be efficiently extended to the transaction scheme of Monero [22], [35], [36], the largest privacy preserving currency.

(3) Cyclic Swaps. We show that our protocols naturally lends themselves to interesting extensions, such as supporting *cyclic swaps*, i.e. atomic swaps involving more than two users. As an example, consider the scenario where Alice wants to exchange some ether for some bitcoins with Carol, who accepts only credit in Ripple. This can be done with the help of Bob, who is willing to exchange ethers for ripples. This translates into

$$\text{Alice} \xrightarrow{\text{1 ETH}} \text{Bob} \xrightarrow{\text{1 XRP}} \text{Carol} \xrightarrow{\text{1 BTC}} \text{Alice}$$

We show that our protocols can be adapted to securely implement swaps among user cycles (for any cycle length) without the need to place additional trust assumptions. We defer the details to Appendix G.

(4) Fungibility. One appealing aspect of our approach is that signed transactions resulting from atomic swaps are identical to standard one-to-one transactions. To the best of our knowledge, this is the first protocol that achieves this form of privacy without requiring additional trust assumptions such as use of trusted hardware or a trusted third party. This has the potential to improve the fungibility of the coins and the scalability of the currency, since it decreases the impact of atomic swaps on the size of the blockchain.

As an amusing exercise for the reader, we have carried out the Bitcoin testnet transaction that corresponds to the atomic swap of a certain amount of bitcoin for ether. We let the reader identify such atomic swap transaction among the five transactions in the Bitcoin testnet [37]–[41] (the solution can be found in Appendix A).

(5) Implementation and Optimizations. We have implemented a prototype of our Schnorr/ECDSA based protocol (Section VI) and evaluated it showing that one instance of universal swap can be executed in less than one second. In order to achieve such performance, we not only take advantage of the parallelization possible in our protocol as operations for different coins are independent of each other, but also describe implementation-level optimizations that greatly improve the performance in practice. Our evaluation also shows that our protocol reduces the on-chain gas cost between 2-6 times and the transaction size when compared to Hash TimeLock Contract (HTLC) contract, demonstrating the best suitability of our protocol for any blockchain (including those with expressive scripting language support).

II. Solution Overview

In this outline, we mostly focus on our generic protocol, which is compatible with any blockchain, assuming the minimal ability to verify signatures on transactions (for any signature scheme). This lays out the main ideas of our approach and it is the basis for our efficient protocol for the special case of Schnorr/ECDSA signatures.

A. Outline of Our Generic Solution

Assume a setting where a party P_0 owns the coin $v^{(0)}$ at ledger $\mathbb{B}_0$ and P_1 owns the coin $v^{(1)}$ at ledger $\mathbb{B}_1$, which they want to securely swap. We assume that the parties have a bootstrapping mechanism (e.g., a forum where they can match their orders and find each other for swapping their coins). A detailed study of this assumption is out of the scope of this paper. The party P_0 could naively transfer $v^{(0)}$ to P_1 in $\mathbb{B}_0$ with the hope that afterwards P_1 transfers $v^{(1)}$ to P_0 in $\mathbb{B}_1$. However, the success of such a swap crucially relies on the honesty of the users: Should P_1 not forward the coins, P_0 would incur a loss.

The central challenge that our protocol needs to address is in ensuring *atomicity* of the swap even in the presence of malicious parties, which is guaranteed by the HTLC-based protocols. Drawing inspiration from that approach, an immediate barrier that we encounter is that the absence of scripting language does not allow us to set "time-outs" on transactions. To avoid users being stuck in deadlocks, we resort to different techniques.

1) Simulating Transaction Timeouts: A timeout t for a transaction tx means that the transaction is accepted by the nodes in the network, only after time t has expired. Typically, this is implemented by expressing t in terms of a block number and leveraging a `timelock` script, that is explicitly included in the transaction and checks whether the block number expressed in t has already been reached. That is, even if the transaction has a valid signature σ but time t has not expired, the `timelock` script prevents the transaction from being processed. Our objective will be to simulate such a functionality without using any on-chain script. Our main leverage to achieve this will be verifiable timed signatures (VTS) [34]. A VTS lets a user (or a committer) generate a timed commitment C of a signature σ on a message m under a public key pk. The commitment

1. Setup	3. Complete
$P_0 : tx^{(0)}_{\mathsf{rfnd}}, \mathsf{VTS}_{\mathbf{T}_0}$	$P_0 : tx^{(0)}_{\mathsf{swp}}, \sigma^{(0)}_{\mathsf{swp}}$ on $\mathbb{B}$
$P_1 : tx^{(1)}_{\mathsf{rfnd}}, \mathsf{VTS}_{\mathbf{T}_1}$	$P_1 : tx^{(1)}_{\mathsf{swp}}, \sigma^{(1)}_{\mathsf{swp}}$ on $\mathbb{B}$
2. Lock	**4. Timeout**
$P_1 : tx^{(1)}_{\mathsf{swp}}, \ell k$	$P_1 : tx^{(1)}_{\mathsf{rfnd}}, \sigma^{(1)}_{\mathsf{rfnd}}$ at $\mathbf{T}_1$
$P_0 : tx^{(0)}_{\mathsf{swp}}, \sigma^{(0)}_{\mathsf{swp}}$	$P_0 : tx^{(1)}_{\mathsf{rfnd}}, \sigma^{(1)}_{\mathsf{rfnd}}$ at $\mathbf{T}_0$

Fig. 1: Results of different phases for parties P_0 and P_1 in a 1-to-1 swap. In each phase, the first party to receive the output in the phase is written at the top followed by the second party to receive the output in the phase.

C must hide the signature σ for time $\mathbf{T}$ (which can be chosen arbitrarily by the committer). At the same time, the committer also generates a proof π that proves that the commitment C contains a valid signature σ: This guarantees that σ can be publicly recovered in time $\mathbf{T}$ by anyone who solves the computational puzzle.

To build some intuition on how to use this tool to simulate a transaction timelock, consider the case of two users (Alice and Bob) sharing an address pk_{AB} (where each party owns a share of the corresponding secret key). Before sending the funds to pk_{AB}, Alice and Bob jointly sign a "refund" transaction tx_{rfnd} that transfers all funds from pk_{AB} back to the address of Alice, in such a way that only Bob learns the signature. Bob then generates a VTS (using time parameter $\mathbf{T}$) on this refund signature and provides Alice with the resulting commitment C and proof π. Note that, if after time $\mathbf{T}$ some funds in pk_{AB} remain unspent, then Alice can immediately refund them by posting the transaction tx_{rfnd} together with the valid signature that she learned from C.

2) One-to-One Atomic Swaps: Equipped with a VTS scheme, we first present a simple single-currency one-to-one atomic swap protocol. The protocol consists of four phases, that we describe in the following. Figure 1 shows the parties' outputs in each phase. For convention, keys, transactions and signatures with (01) are involved in the payment from P_0 to P_1 and (10) are involved in the payment from P_1 to P_0.

Swap Setup Phase. In the setup phase, the parties transfer their coins to new joint addresses $pk^{(01)}$ and $pk^{(10)}$ (one for each coin) that both parties together control. More concretely, we have that

$$sk^{(01)} = sk_0^{(01)} \oplus sk_1^{(01)} \text{ and } sk^{(10)} = sk_0^{(10)} \oplus sk_1^{(10)}$$

where each party possesses one share of each signing key. However, before transferring the coins to the joint keys, the parties need to ensure that the coins will not be locked forever in the joint address, in case one party goes offline.

As briefly discussed above, this is done by generating two refund transactions[1] of the form

$$tx^{(0)}_{\mathsf{rfnd}} : pk^{(01)} \xrightarrow{v^{(0)}} P_0 \qquad tx^{(1)}_{\mathsf{rfnd}} : pk^{(10)} \xrightarrow{v^{(1)}} P_1.$$

Then P_1 generates a VTS for the signature on the former refund transaction with time parameter $\mathbf{T}_0$ and P_0 generates a VTS for the signature on the latter refund transaction with time parameter $\mathbf{T}_1$. The time parameters are set such that $\mathbf{T}_0 = \mathbf{T}_1 + \Delta$, where Δ is a conservatively chosen delay parameter. This gap Δ prevents race conditions and ensures that a adversarial P_0 *cannot* wait until the last moment to retrieve the coin $v^{(1)}$, in the hope that the transaction swapping $v^{(0)}$ expires in the meantime, effectively stealing coins from P_1. Once both VTS are verified, the parties proceed by transferring the coins to the shared addresses by signing the freeze transactions of the form

$$tx^{(0)}_{\mathsf{frz}} : P_0 \xrightarrow{v^{(0)}} pk^{(01)} \qquad tx^{(1)}_{\mathsf{frz}} : P_1 \xrightarrow{v^{(1)}} pk^{(10)}.$$

Swap Lock Phase. After a successful setup phase, parties generate payment "locks" on transactions that spend from the joint keys. Specifically, they define the following swap transactions

$$tx^{(1)}_{\mathsf{swp}} : pk^{(01)} \xrightarrow{v^{(0)}} P_1 \qquad tx^{(0)}_{\mathsf{swp}} : pk^{(10)} \xrightarrow{v^{(1)}} P_0.$$

Since the secret keys of the joint addresses are secret shared among the parties, a natural idea is to compute the respective signatures using a secure 2-party computation (2PC) protocol. However, this naive attempt leads to an *insecure* scheme: The 2PC protocol does not guarantee any form of guaranteed output delivery,[2] so nothing prevents P_0 from going offline after receiving a valid signature on $tx^{(0)}_{\mathsf{swp}}$. Instead, the parties first compute via a 2PC, a "locked" version of a signature on $tx^{(1)}_{\mathsf{swp}}$, i.e.,

$$\ell k := \sigma^{(1)}_{\mathsf{swp}} \oplus H\left(\sigma^{(0)}_{\mathsf{swp}}\right)$$

where H is a hash function (modelled as a random oracle[3]). Observe that the $\oplus$ operation mimics the one-time pad encryption with $H\left(\sigma^{(0)}_{\mathsf{swp}}\right)$ as the encryption key and $\sigma^{(1)}_{\mathsf{swp}}$ as the encrypted message. Note that at this point, neither party knows the valid signature $\sigma^{(1)}_{\mathsf{swp}}$ since it is masked by the output of H. However, we now know that if P_0 were to somehow publish the signature on $tx^{(0)}_{\mathsf{swp}}$, then P_1 would immediately learn a valid signature on $tx^{(1)}_{\mathsf{swp}}$, by recomputing[4] and removing the mask. Only after this "lock" step is successfully completed, both parties engage in a 2PC to jointly compute a signature on $tx^{(0)}_{\mathsf{swp}}$, which is now safe to reveal.

Swap Complete Phase. After a successful lock phase, each party can post the respective transaction-signature pair on the

[1] We assume the parties come to an agreement on the transaction fee for all swap related transactions.

[2] The standard security notion for 2PC is *security with aborts*, which allows an adversarial party to learn the output of the computation while preventing honest parties to do so.

[3] A standard model instantiation is possible if we let H be a sufficiently stretching leakage resilient pseudorandom generator that can be constructed from Extremely Lossy Functions [42]

[4] We assume the signing algorithm is deterministic.

blockchain. As mentioned before, while P_0 can simply read the signature in plain, P_1 recovers his signature by computing

$$\sigma_{\mathsf{swp}}^{(1)} = \ell k \oplus H\left(\sigma_{\mathsf{swp}}^{(0)}\right).$$

This guarantees the atomicity of the swap: P_0 cannot obtain $v^{(1)}$ without P_1 transferring $v^{(0)}$, and vice versa.

Swap Timeout Phase. If at any point in the setup or lock phase either of the party goes offline, then party P_b can recover her coins from the joint address using the time-locked signature $\sigma_{\mathsf{swp}}^{(b)}$, which she eventually learns by opening the VTS. Here, as shown in Figure 1, party P_1 can recover her coins first as $\mathbf{T}_1 < \mathbf{T}_0$. Thus, the parties are guaranteed to not lose their coins if any participant goes offline for extended periods of time.

3) Multi-Asset Atomic Swaps: The most general (and realistic) setting that we consider is when P_0 holds n coins (possibly from different currencies) and P_1 holds $\tilde{n}$ coins (from a possibly disjoint set of currencies), which they want to swap atomically. This situation is presented in Figure 2.

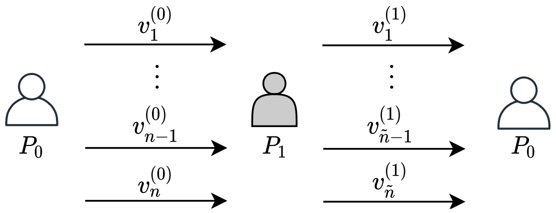

Fig. 2: Setting for a n-to-$\tilde{n}$ swap.

Before delving into the details on how to modify our vanilla protocol to support multi-asset swaps, let us pause and discuss what kind of security we (intuitively) expect from such a protocol. On P_0 side, we want to ensure that P_1 cannot claim (or transfer) any of P_0's coin before P_0 *holds signatures* on transactions for *all* of P_1's coin. P_0 can obtain P_1's coins by posting the transactions and the corresponding signatures. The coins are then considered transacted to P_0. Conversely, we want to guarantee that if *any* of P_1's coin is *transacted* to P_0, then P_1 can simultaneously learn signatures on *all transactions* that spend P_0's coins to P_1.

To do this, we proceed by viewing the swap as a collection of $\tilde{n}$ separate n-to-1 atomic swaps. Intuitively, this forces P_0 to wait until the end of all $\tilde{n}$ iteration before publishing any signature, as otherwise P_1 would be able to claim all of P_0's coins. To implement a single n-to-1 atomic swap we need to be able to generate a *single* ℓk that simultaneously locks n signatures $(\sigma_1, \dots, \sigma_n)$ and condition their release on a single signature $\tilde{\sigma}$. We realize this extending our one-to-one locking mechanism and computing ℓk as

$$\ell k = (\sigma_1 \oplus H(1||\tilde{\sigma}), \dots, \sigma_n \oplus H(n||\tilde{\sigma})).$$

Since we model H as a random oracle, this allows us to stretch the randomness extracted from $\tilde{\sigma}$ as much as we need. Note that once $\tilde{\sigma}$ is revealed, all of the other signatures are simultaneously unmasked. Running $\tilde{n}$ copies of this modified protocol once for each of P_0's coins, allows us to achieve a secure n-to-$\tilde{n}$ multi-asset swap.

Shielded Addresses of Zcash. Our approach can also be extended to atomic swaps of shielded addresses of Zcash [26], with the aid of 2PC protocols for generating SNARK proofs [43]. Zcash provides two types of addresses: shielded (that retain privacy) and unshielded (that are trasnparent like Bitcoin addresses), and prior works only support atomic swap for the unshielded addresses of Zcash [44].

Trade-off with HTLC based Solutions. Note that in HTLC based solutions (described in Section I-A) we implicitly assume the adversary cannot force parties to go offline for too long. In our VTS based solution, we implicitly assume that an adversary cannot force a party not to perform local computation for too long, for example by cutting off the computation power of the party for extended periods of time. An advantage of our approach is that a party only have to spend computational effort to open their VTS if the other party does not respond quickly. In other words, opening of VTS is only a deterrent mechanism in case the parties do not instantaneously complete the swap. On the other hand, in the HTLC approach, parties have to pay the transaction fee/gas cost associated with the HTLC invariably.

B. The Case of Schnorr/ECDSA Signatures

While our generic protocol satisfies all desired properties simultaneously (non-custodial, universal, multi-asset), it falls short in achieving concrete computational efficiency: The usage of generic tools (such as general-purpose 2PC) might make the cost of running such a protocol prohibitive for some users. However, we can use the general blueprint established by the approach to develop efficient protocols for specific signature schemes.

In this regard, we revisit our general framework to dramatically improve its efficiency for the case where the signature scheme verified by the ledgers is either Schnorr or ECDSA. While this is a downgrade for the generality of our approach, we remark that virtually all major cryptocurrencies rely on Schnorr or ECDSA signatures, which allows us to remain compatible with the vast majority of coins. In the following, we give a cursory outlook at the aspects that we improve and we refer the reader to Section V for a precise description of the protocol.

More Efficient VTS. Time-lock encryption [45] in principle lets us encrypt messages to the future, however there is no known practically efficient instantiation of their proposal. Practically efficient VTS constructions tailored for Schnorr/ECDSA signatures were presented in [34]. However, we observe that we can further improve the efficiency by committing to the whole signature key corresponding to $pk^{(01)}$ and $pk^{(10)}$ (instead of the signature), since the public keys are used only once. Recall that in Schnorr/ECDSA signatures, secret keys are integers x and public keys are of the form (G, G^x), where G is the generator of some cyclic group of prime order. Thus, instead of VTS, we can generate commitments to a secret key x via a verifiable timed discrete logarithm (VTD) scheme [34]. Concretely, the

VTD construction from [34] is far more efficient than the VTS constructions for Schnorr/ECDSA signatures in terms of commitment generation and verification, as the committer no longer needs to prove that the committed signature is valid. Instead, the committer needs to prove that the committed value is a valid discrete logarithm of some known element, which is a simpler algebraic statement. This allows us to significantly boost the efficiency of our setup phase. However, in our protocol we need to ensure that the parties generate valid VTD commitments to their shares of the secret key x as neither party has access to the whole secret key at the beginning of the protocol. We defer the details to Section V-D.

Avoiding General-Purpose 2PC. Instead of relying on a general-purpose 2PC protocol to compute a "locked" signature, we leverage atomic multi-hop locks [46] originally introduced in the context of payment channel networks.[5] On a high level, their 2PC protocol lets party P_0 and P_1 jointly compute a pre-signature $\tilde{\sigma}$ on a message m under the joint public key pk, with respect to an instance Y of a hard relation R. The property of interest here is that $\tilde{\sigma}$ is not a valid signature on the message m, but can be adapted into a valid signature σ with the knowledge of a witness y, such that $(Y, y) \in R$. Additionally, given the pre-signature $\tilde{\sigma}$ and the valid signature σ, one can efficiently extract the witness y for the instance Y. A first approach would be to use their protocol instead of a 2PC, however it turns out that we can do even better by tweaking the composition of different protocol instances.

Reducing the Number of Iterations. We show how to reduce the number of iterations for the execution of lock protocol from $\tilde{n} \cdot (n+1)$ to an additive $n + \tilde{n}$. The overall idea to do this, is to let the parties engage in the 2PC protocol from [46] for each swap transaction ($n + \tilde{n}$ in total) exactly once, generating a pre-signature on the transaction under the corresponding joint public key. Importantly, all the pre-signatures are generated with respect to the same instance Y of a hard relation R, where one of the two parties (in our case P_0) knows the corresponding witness y.

Care must be taken to ensure an ordering such that party P_1 first obtains all the pre-signatures on the transactions from P_0 to P_1, before party P_0 obtains *any* pre-signature on any transaction from P_1 to P_0. This is to prevent P_0 from completing any pre-signature, since he knows the witness y. Once all the $n+\tilde{n}$ pre-signatures are generated and available with both parties, the swap can be completed by P_0, using the knowledge of y. On the other hand, P_1 can extract the witness y from any of the signatures posted by P_0 and therefore turn his pre-signatures into valid ones. This completes the swap.

Optimisations. We have two possible optimisations to reduce computational work for both parties in terms of opening the VTD commitments. In the first optimisation, party P_0 instead of computing on n VTD commitments, can homomorphically combine those commitments and work on opening only a single VTD commitment. Similar optimisation is possible for party P_1 also. The technique is called *batching VTD commitments* [48] and we discuss this in more detail in Appendix G.

In Section VI we implement the second optimization where instead of $n+\tilde{n}$ VTD commitments, *only* 2 VTD commitments are generated. Now the parties solve *one commitment each* instead of n and $\tilde{n}$ like before. To do this, we exploit the key structure in Schnorr/ECDSA signature schemes, where opening one VTD commitment helps P_0 learn n secret keys and vice versa for P_1. The parties additionally need to execute a joint coin tossing protocol $n+\tilde{n}$ times which is significantly cheaper than computing on $n+\tilde{n}$ VTD commitments.

Optimising Number of Swaps. With the above optimisation, the parties are still required to perform persistent computation to open their respective VTD commitments. This could limit the number of coins (n and $\tilde{n}$) that the parties may want to swap simultaneously with many other parties. However, the persistent computation of opening a VTD commitment can be securely outsourced to a decentralized service [49] at a market determined cost. This relieves both parties of any potentially heavy computation related to VTD opening. As a consequence, provided the parties have enough funds to outsource VTD openings, the number of coin swaps of a party is no longer limited by its own computational power.

Optimistic Efficiency. More importantly, in the optimistic case (i.e., where both parties P_0 and P_1 are honest and remain online until the end of the protocol) our protocol terminates instantly, without either party having to invest computational resources into opening the VTD commitments. In practice, we expect that the presence of VTD commitments will mostly function as a deterrent for people not to misbehave and the parties will not have to open them, except for rare cases. Therefore, in the pessimistic case where parties need to use the deterrent mechanism, parties need to dedicate one CPU core for solving a certain number of batched VTD commitments.

Extensions. Finally, we show (Section V-E) how to handle a mixture of Schnorr/ECDSA signatures even when implemented over different curves (that define groups of different orders).

III. Preliminaries

We denote by $\lambda \in \mathbb{N}$ the security parameter and by $x \leftarrow \mathcal{A}(\mathsf{in}; r)$ the output of the algorithm $\mathcal{A}$ on input in using $r \leftarrow \{0,1\}^*$ as its randomness. We often omit this randomness and only mention it explicitly when required. The notation $[n]$ denotes a set $\{1, \ldots, n\}$ and $[i, j]$ denotes the set $\{i, i+1, \ldots, j\}$. We consider *probabilistic polynomial time* (PPT) machines as efficient algorithms.

Universal Composability. We model security in the *universal composability* framework from Canetti [50] extended to support a global setup [51], which lets us model concurrent executions. We refer the reader to [50] for a comprehensive discussion. We consider *static* corruptions, where the adversary announces at the beginning which parties he corrupts. We denote the environment by $\mathcal{E}$. For a real protocol Π and an adversary $\mathcal{A}$ we write $EXEC_{\tau,\mathcal{A},\mathcal{E}}$ to denote the ensemble corresponding to the protocol execution. For an ideal functionality $\mathcal{F}$ and an

[5] This functionality can be abstracted into what is referred to as *adaptor signature* [47].

adversary $\mathcal{S}$ we write $EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$ to denote the distribution ensemble of the ideal world execution.

Definition 1 (Universal Composability): A protocol τ 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 $EXEC_{\tau,\mathcal{A},\mathcal{E}}$ and $EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are computationally indistinguishable.

Digital Signatures. A digital signature scheme DS, formally, has a key generation algorithm $\mathsf{KGen}(1^\lambda)$ that takes the security parameter 1^λ and outputs the public/secret key pair (pk, sk), a signing algorithm $\mathsf{Sign}(sk, m)$ inputs a secret key and a message $m \in \{0,1\}^*$ and outputs a signature σ, and a verification algorithm $\mathsf{Vf}(pk, m, \sigma)$ outputs 1 if σ is a valid signature on m under the public key pk, and outputs 0 otherwise. We require the standard notion unforgeability for the signature scheme [52]. A stronger notion of strong unforgeability for the signature scheme was shown to be equivalent to the UC formulation of security [53].

Hard Relations. We recall the notion of a hard relation R with statement/witness pairs (Y, y). We denote by $\mathcal{L}_R$ the associated language defined as $\mathcal{L}_R := \{Y | \exists y \ s.t. \ (Y, y) \in R\}$. The relation is called a hard relation if the following holds: (i) There exists a PPT sampling algorithm $\mathsf{GenR}(1^\lambda)$ that outputs a statement/witness pair $(Y, y) \in R$; (ii) The relation is poly-time decidable; (iii) For all PPT adversaries $\mathcal{A}$ the probability of $\mathcal{A}$ on input Y outputting a witness y is negligible.

2-Party Computation. The aim of a secure 2-party computation (2PC) protocol is for the two participating users P_0 and P_1 to securely compute some function f over their private inputs x_0 and x_1, respectively. Apart from correctness of output, we require *privacy* that states that the only information learned by the parties in the computation is that specified by the function output. Note that we require the standard *security with aborts*, where the adversary can decide whether the honest party will receive the output of the computation or not. I.e., we do not assume any form of fairness or guaranteed output delivery. For a comprehensive treatment of the formal UC definition we refer the reader to [31]. As standard in the UC settings, we work in the *static* corruption model, where the adversary declares which party will be corrupted ahead of time.

Synchrony and Communication. We assume synchronous communication between users, where the execution of the protocol happens in rounds. We model this via an ideal functionality $\mathcal{F}_{\mathsf{clock}}$ [54], [55], where all honest parties are required to indicate that are ready to proceed to the next round before the clock proceeds. The clock functionality that we consider is fully described in [51]. This means that all entities are always aware of the given round. Users can abort a session at any given round by sending a distinguished message (abort). We also assume secure message transmission channels between users modelled by the ideal functionality $\mathcal{F}_{\mathsf{smt}}$.

Blockchain. We assume the existence of an ideal ledger (blockchain) functionality $\mathbb{B}$ (just as in [9], [46], [56]) that maintains the list of coins currently associated with each address and that we model as a trusted append-only bulletin board. For simplicity of notation, we make use of $\mathbb{B}$ for the chain of all the currencies involved in the swap. The corresponding ideal functionality $\mathcal{F}_{\mathbb{B}}$ maintains the ledger $\mathbb{B}$ locally and updates it according to the transactions between users. More precisely, it offers an interface $\mathtt{Post}(\mathsf{id}, A, B, v)$ to transfer v coins during a session with identifier id, from address A (with associated verification key pk_A), to address B (with associated verification key pk_B), if provided with sk_A. Users may use the interface to transact among themselves. An additional interface $\mathtt{Register}$, is for users to register their address A along with the associated verification key pk_A and a value v, which is stored in the ledger.

At any point in the execution, any user U can send a distinguished message $\mathtt{Read}$ to $\mathcal{F}_{\mathbb{B}}$, who sends the whole transcript of $\mathbb{B}$ to U. We refer the reader to [56] for a formal definition of this functionality.

IV. Definitions for Atomic Swaps

In the following we motivate and present the security definition of atomic swaps in the form of an ideal functionality. The main property the ideal functionality must guarantee is *atomicity*: Either both users interested in the swap successfully swap their coins, or the swap fails and everyone is back to their initial holdings. For the more general case of n-to-$\tilde{n}$ swap between users U_0 and U_1, the atomicity notion we set out to achieve is: if at least one of the $\tilde{n}$ coins is moved to U_0, it is possible for U_1 (if has not aborted) to swap *all of the* n coins. This notion ensures if U_0 initiates the swap, U_1 does not lose coins, and if U_0 aborts the swap before initiation, the swap is aborted and she does not lose coins.

Discussion. Before delving into the description of the ideal functionality, we discuss why existing definitions fall short in capturing the security of universal atomic swaps.

The notion of cryptographic fairness (the analogue of output atomicity in multi-party computation) has been studied extensively in the literature, even in the context of blockchains [57], [58]. The UC modelling of such a functionality guarantees that honest participants in the computation receive their output if the adversary also receives its. This notion intuitively seems to capture the security for an atomic swap, i.e., if the adversary gets an honest party's digital good, the honest party gets the adversary's digital good. However, there are subtle aspects that make it insufficient to model security in our setting.

To exemplify the problem, consider the scenario where the adversary has a coin v_a at address A and the honest party has a coin v_h at address H that they want to swap. Both parties register the addresses A and H with the ideal functionality. To complete the swap, the coin v_a must be transferred from A to an address for which the honest party knows the authentication key, and vice versa for the coin v_h. A naive idea to implement this using the fair exchange functionality would be for the parties to simply exchange the secret keys of their addresses. Unfortunately, this attempt turns out to lead to a completely insecure protocol: During the execution of the fair exchange, the attacker might quickly move all the coins from A to some other address, before the execution of the protocol is completed. In the end, the honest party will end up with the secret key of the empty address A, whereas the attacker will learn the secret key for H, effectively stealing coins.

Prior works [6] address this issue by leveraging a special *coin freezing* functionality of the blockchain, thus preventing the above described attack. However, the modelling of [6] implicitly relies on special scripting capabilities of the blockchain to implement the coin freezing function. Since we are interested in settings, where the blockchain may not offer such scripts, we have to work around this issue. Our functionality (largely inspired by [6]) will implement coin freezing only by using the standard interfaces offered by any blockchain (i.e., address registration and transaction posting).

A. Ideal Functionality

We now formally describe our ideal functionality $\mathcal{F}_{\mathsf{swap}}$ (Figure 3) for atomic swaps of coins that circumvents the above problem. $\mathcal{F}_{\mathsf{swap}}$ interacts with the two users U_0, U_1 and the simulator $\mathcal{S}$. In the first round, user U_1 initiates a swap with the message $(\mathtt{swap}_1, \mathsf{id}, V, \tilde{V}, PK, \tilde{PK}, \tilde{SK})$ that specifies the coins $V := (v_1, \ldots, v_n)$ (owned by U_0) and $\tilde{V} := (\tilde{v}_1, \ldots, \tilde{v}_{\tilde{n}})$ (owned by U_1) that are to be swapped. User U_1 also gives his authentication keys $\tilde{\mathsf{SK}}$ for the addresses $\tilde{\mathsf{PK}}$ corresponding to the coins in $\tilde{V}$.

In the second round, user U_0 also acknowledges the swap initiation by giving his authentication keys SK for the addresses PK corresponding to the coins V. The ideal functionality uses the `Freeze` subroutine to transfer each of those coins to an id-specific address that is controlled by the functionality. The functionality can do this because it knows the authentication keys of each of those coins and can interact with $\mathcal{F}_{\mathbb{B}}$ to transfer coins to these functionality controlled keys. If some transaction fails, then the functionality returns the coins frozen so far to the original owner.

In the third round, if user U_0 aborts, all the coins are unfrozen by the ideal functionality and transferred back to the respective users. On the other hand, if the functionality receives buy for some index set $J \subseteq [\tilde{n}]$ of coins from U_0, then it uses the `Transfer` subroutine to transfer the coins $\tilde{v}_j$ for $j \in J$ from its control to user U_0.

In the last round, if user U_1 aborts, all the frozen coins of user U_0 (from V) are unfrozen by the ideal functionality and transferred back to user U_0. On the other hand, all coins of U_1 that remained frozen (coins not in J) are unfrozen and transferred back to U_1. Otherwise, if user U_1 responds with buy for some index $I \subseteq [n]$, the functionality uses the `Transfer` subroutine to transfer the coins v_i for $i \in I$ to user U_1. Finally, all coins still controlled by the functionality are unfrozen and transferred to their initial owners.

Atomicity. The functionality ensures that if user U_0 in round 3 aborts the swap, all the coins are refunded to both parties. Instead if U_0 initiates a swap for any subset of $\tilde{n}$ coins in round 3, user U_1 is allowed to complete the swap in round 4 for all n coins. Since no party can unilaterally transfer a coin (as it is locked with functionality), the functionality guarantees atomicity for the n-to-$\tilde{n}$ swap.

The ideal functionality $\mathcal{F}_{\mathsf{swap}}$ (in session id) interacts with users U_0 and U_1 and the ideal adversary $\mathcal{S}$.

(Round 1) Upon receiving $(\mathtt{swap}_1, \mathsf{id}, V, \tilde{V}, PK, \tilde{PK}, \tilde{SK})$ from U_1, where $V := (v_1, \ldots, v_n) \in \mathbb{N}$, $\tilde{V} := (\tilde{v}_1, \ldots, \tilde{v}_{\tilde{n}}) \in \mathbb{N}$, $PK := (pk_1, \ldots, pk_n)$, $\tilde{PK} := (\tilde{pk}_1, \ldots, \tilde{pk}_{\tilde{n}})$, $\tilde{SK} := (\tilde{sk}_1, \ldots, \tilde{sk}_{\tilde{n}})$. Send $(\mathtt{swap}_1, \mathsf{id}, V, \tilde{V}, PK, \tilde{PK}, U_1)$ to $\mathcal{S}$ and store the input tuple.

(Round 2) Upon receiving $(\mathtt{swap}_2, \mathsf{id}, V, \tilde{V}, PK, \tilde{PK}, SK)$ from user U_0, where $\mathsf{SK} := (sk_1, \ldots, sk_n)$, call the subroutines $\mathtt{Freeze}(\mathsf{id}, v_i, pk_i, sk_i, i)$ and $\mathtt{Freeze}(\mathsf{id}, \tilde{v}_j, \tilde{pk}_j, \tilde{sk}_j, j)$ for all $i \in [n]$ and $j \in [\tilde{n}]$. If all of the invocations are successful, send $(\mathtt{swap}_2, \mathsf{id}, V, \tilde{V}, PK, \tilde{PK}, U_0)$ to $\mathcal{S}$, store the input tuple and proceed to round 3. Otherwise, revert the coin freezing by invoking the corresponding subroutines $\mathtt{UnFreeze}(\mathsf{id}, v_i, pk_i, i)$ and $\mathtt{UnFreeze}(\mathsf{id}, \tilde{v}_j, \tilde{pk}_j, j)$.

(Round 3) Upon receiving $(\mathtt{abort}, \mathsf{id})$ from the user U_0, revert the freezing by calling $\mathtt{UnFreeze}(\mathsf{id}, v_i, pk_i, i)$ and $\mathtt{UnFreeze}(\mathsf{id}, \tilde{v}_j, \tilde{pk}_j, j)$ for all $i \in [n]$ and $j \in [\tilde{n}]$. Otherwise, upon receiving $(\mathtt{buy}, \mathsf{id}, J, pk)$, for some $J \subseteq [\tilde{n}]$, call the subroutine $\mathtt{Transfer}(\mathsf{id}, J, pk)$. Store J and proceed to round 4.

(Round 4) Upon receiving $(\mathtt{abort}, \mathsf{id})$ from the user U_1, call $\mathtt{UnFreeze}(\mathsf{id}, v_i, pk_i, i)$ for all $i \in [n]$ and $\mathtt{UnFreeze}(\mathsf{id}, \tilde{v}_j, \tilde{pk}_j, j)$ for all $j \notin J$ and terminate. Otherwise, upon receiver $(\mathtt{buy}, \mathsf{id}, I, \tilde{pk})$ from U_1 do the following:

- If $I \neq \emptyset$ call $\mathtt{Transfer}(\mathsf{id}, I, \tilde{pk})$, then $\mathtt{UnFreeze}(\mathsf{id}, \tilde{v}_j, \tilde{pk}_j, j)$ for all $j \notin J$ and $\mathtt{UnFreeze}(\mathsf{id}, v_i, pk_i, i)$ for all $i \notin I$.
- If $I = \emptyset$ call $\mathtt{UnFreeze}(\mathsf{id}, v_i, pk_i, i)$ and $\mathtt{UnFreeze}(\mathsf{id}, \tilde{v}_j, \tilde{pk}_j, j)$ for all $i \in [n]$ and $j \notin J$.

Notify $\mathcal{S}$ of the outcome of the operation.

Description of the subroutines

Freeze: On input a tuple $(\mathsf{id}, v, pk, sk, i)$, transfer v coins from the address specified by pk (using sk) to some id-specific address pk_{id} controlled by the ideal functionality via $\mathtt{Post}(\mathsf{id}, pk, pk_{\mathsf{id}}, v)$. The function is successful if the transaction is accepted by $\mathcal{F}_{\mathbb{B}}$.

UnFreeze: On input a tuple (id, v, pk, i) transfer back the v frozen coins to the corresponding public key pk, via $\mathtt{Post}(\mathsf{id}, pk_{\mathsf{id}}, pk, v)$.

Transfer: On input a tuple (id, I, pk), transfer all frozen coins corresponding to the index set I to the public key pk, via $\mathtt{Post}(\mathsf{id}, pk_{\mathsf{id}}, pk, \sum_{i \in I} v_i)$.

Fig. 3: Ideal functionality $\mathcal{F}^{\mathbb{B}}_{\mathsf{swap}}$ for fair swap of coins

Fungibility. Notice that the functionality only makes standard `Transfer` calls to the blockchain $\mathbb{B}$. No additional information is present, besides verification keys and transacted values. Thus these calls are syntactically identical to regular transactions.

V. Atomic Swaps from Adaptor Signatures

Here we present an efficient atomic swap protocol for n-to-$\tilde{n}$ swap of coins between two users P_0 and P_1, under the condition that transactions are signed using Schnorr or ECDSA signatures. The fundamental building blocks of our protocol are adaptor signatures and verifiable timed dlog (VTD), both of which are defined below.

A. Adaptor Signature

Adaptor signatures [47] let users generate a pre-signature on a message m which by itself is not a valid signature, but can later be adapted into a valid signature if the user knows some secret value. The formal definition of adaptor signatures is given below.

Definition 2 (Adaptor Signatures): An adaptor signature scheme Π_{AS} w.r.t. a hard relation R and a signature scheme $\Pi_{\mathsf{DS}} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ consists of algorithms $(\mathsf{pSign}, \mathsf{Adapt}, \mathsf{pVf}, \mathsf{Ext})$ defined as:

$\hat{\sigma} \leftarrow \mathsf{pSign}(sk, m, Y)$: The pre-sign algorithm takes as input a secret key sk, message $m \in \{0,1\}^*$ and statement $Y \in L_R$, outputs a pre-signature $\hat{\sigma}$.

$0/1 \leftarrow \mathsf{pVf}(pk, m, Y, \hat{\sigma})$: The pre-verify algorithm takes as input a public key pk, message $m \in \{0,1\}^*$, statement $Y \in L_R$ and pre-signature $\hat{\sigma}$, outputs a bit b.

$\sigma \leftarrow \mathsf{Adapt}(\hat{\sigma}, y)$: The adapt algorithm takes as input a pre-signature $\hat{\sigma}$ and witness y, outputs a signature σ.

$y \leftarrow \mathsf{Ext}(\sigma, \hat{\sigma}, Y)$: The extract algorithm takes as input a signature σ, pre-signature $\hat{\sigma}$ and statement $Y \in L_R$, outputs a witness y such that $(Y, y) \in R$, or $\perp$.

In terms of security, we want (i) *unforgeability* that is similar to the unforgeability of standard signature schemes, except that we additionally want that producing a forgery σ for some message m is hard even given a pre-signature $\tilde{\sigma}$ on m, with respect to some uniformly sampled instance $Y \in \mathcal{L}_R$. We then want (ii) *witness extractability* that guarantees that given a valid signature σ and a pre-signature $\tilde{\sigma}$ for a message m and an instance Y, one can efficiently extract a witness y for Y. Finally, we want (iii) *pre-signature adaptability* that guarantees that any valid pre-signature $\tilde{\sigma}$ generated with respect to an instance Y, can be adapted into a valid signature σ if given the witness y for the instance Y. For formal definitions we refer the reader to Appendix C.

In this work, we consider the constructions proposed in [47] of adaptor signatures where the signature scheme Π_{DS} is of Schnorr or ECDSA. The hard relation R used in their constructions is that of the discrete log relation, where the language is defined as: $\mathcal{L}_R := \{H : \exists x \in \mathbb{Z}_q^*,\ s.t.\ H = G^x\}$, where $(\mathbb{G}, G, q)$ are the group description, its generator, and its order, respectively.

B. Verifiable Timed DLog

A Verifiable timed dlog [34] is defined with respect to a group $\mathbb{G}$ of order q and a generator G. Here, a committer generates a timed commitment of timing hardness $\mathbf{T}$ of a value $x \in \mathbb{Z}_q^*$ such that $H = G^x$ where H is public and x is referred to as the dlog. (discrete logarithm) of H (wrt. G). The verifier checks the well-formedness of the timed commitment and can learn the value x by force opening the commitment in time $\mathbf{T}$. The formal definition is as follows.

Definition 3 (Verifiable Timed Dlog): A VTD for the group $\mathbb{G}$ with generator G and order q is a tuple of four algorithms $(\mathsf{Commit}, \mathsf{Verify}, \mathsf{Open}, \mathsf{ForceOp})$ where:

$(C, \pi) \leftarrow \mathsf{Commit}(x, \mathbf{T})$: the commit algorithm (randomized) takes as input a discrete log value $x \in \mathbb{Z}_q$ (generated using $\mathsf{KGen}(1^\lambda)$) and a hiding time $\mathbf{T}$ and outputs a commitment C and a proof π.

$0/1 \leftarrow \mathsf{Verify}(H, C, \pi)$: the verify algorithm takes as input a group element H, a commitment C of hardness $\mathbf{T}$ and a proof π and outputs 1 if and only if, the value x embedded in C satisfies $H = G^x$. Otherwise it outputs 0.

$(x, r) \leftarrow \mathsf{Open}(C)$: the open algorithm (run by committer) takes as input a commitment C and outputs the committed value x and the randomness r used in generating C.

$x \leftarrow \mathsf{ForceOp}(C)$: the force open algorithm takes as input the commitment C and outputs a discrete log value x.

The security requirements for a VTD scheme are that of (i) *soundness* where the user is convinced that, given C, the $\mathsf{ForceOp}$ algorithm will produce the committed dlog. value x in time $\mathbf{T}$ and (ii) *privacy*, where all PRAM algorithms[6] whose running time is at most t (where $t < \mathbf{T}$) succeed in extracting x from the commitment C and π with at most negligible probability. For formal definitions we refer the reader to Appendix E.

The construction of VTD from [34] makes use of time-lock puzzles [59], a primitive that lets a committer embed a secret inside a puzzle with the guarantee that the puzzle can be opened only after time $\mathbf{T}$. Specifically, the committer embeds the dlog. value x inside a time-lock puzzle and uses a special NIZK proof to prove its validity. Specifically, the NIZK proves that the time-lock puzzle can be solved in time $\mathbf{T}$ and whats embedded inside satisfies the equation $H = G^x$. They construct such an efficient NIZK proof using cut-and-choose techniques, Shamir secret sharing [60] and homomorphic time-lock puzzles [61]. For completeness, we recall their construction in Figure 9.

C. 2-Party Protocols

Joint Key Generation. We require that two parties P_0 and P_1 jointly generate (ECDSA and Schnorr) keys. We denote this interactive protocol by $\Gamma^{\mathsf{SIG}}_{\mathsf{JKGen}}$, where $\mathsf{SIG} = \mathsf{Schnorr}$ or $\mathsf{SIG} = \mathsf{ECDSA}$. The joint key generation protocol $\Gamma^{\mathsf{SIG}}_{\mathsf{JKGen}}$ takes as input the security parameter 1^λ and the group description $(\mathbb{G}, G, q)$ as input from both parties. The protocol outputs the public key pk to both parties and outputs the secret key share sk_0 to P_0 and sk_1 to P_1. We efficiently instantiate $\Gamma^{\mathsf{Schnorr}}_{\mathsf{JKGen}}$ with the interactive protocol from [62], where in the end we have $pk = G^{sk}$ (sk is the secret key), and $sk = (sk_0 + sk_1)$. On the other hand, we instantiate $\Gamma^{\mathsf{ECDSA}}_{\mathsf{JKGen}}$ with the protocol

[6] PRAM algorithms are polynomial time algorithms that have polynomially bounded amount of parallel computing power.

from [63] where in the end we have $pk = G^{sk}$ (sk is the secret key), and $sk = (sk_0 \cdot sk_1)$.

Joint Adaptor Signing. We require the parties P_0 and P_1 to jointly generate adaptor signatures on messages with respect to an instance Y of a hard (dlog) relation R, for a joint public key pk. We denote this interactive protocol by $\Gamma^{\mathsf{SIG}}_{\mathsf{AdpSg}}$ which takes as common input a message m, and an instance Y of the hard relation R. As private input, party P_0 and P_1 input their secret key share sk_0 and sk_1, respectively. Here sk_0 and sk_1 are shares of the secret key sk whose corresponding public key is pk. The output of the protocol for both parties is the pre-signature $\tilde{\sigma}$, such that it holds that $\Pi^{\mathsf{SIG}}_{\mathsf{AS}}.\mathsf{pVf}(pk, m, Y; \tilde{\sigma}) = 1$. We efficiently instantiate $\Gamma^{\mathsf{SIG}}_{\mathsf{AdpSg}}$ for $\mathsf{SIG} = \mathsf{Schnorr}$ and $\mathsf{SIG} = \mathsf{ECDSA}$ with the protocols from [46].

D. *Our Protocol*

We now describe our atomic swap protocol for transaction schemes based on Schnorr and ECDSA signature schemes. In Figure 5 and Figure 6, we present the protocol for n-to-$\tilde{n}$ atomic swaps. That is, coins $v_i^{(0)}$ for $i \in [n]$ of party P_0 are swapped with coins $v_k^{(1)}$ for $k \in [\tilde{n}]$ belonging to party P_1.

On Choosing the Timing Hardness $\mathbf{T}_0$ and $\mathbf{T}_1$. The parties shall make conservative estimates of each other's computational power in force opening the VTD commitments. This is to prevent scenarios where P_0 or P_1 with a powerful machine force opens its VTD commitments earlier than expected in terms of real time. The party could potentially steal the coins of the other party during the swap lock or swap complete phase. In particular, the parties must ensure that Δ (such that $\mathbf{T}_0 = \mathbf{T}_1 + \Delta$) is large enough such that it tolerates the time differences to open the VTD commitments.

Security Analysis. In the below theorem, we precisely state the security of the above atomic swap protocol. For a formal proof, we refer the reader to Appendix F.

Theorem 5.1: Let $\mathsf{SIG} \in \{\mathsf{Schnorr}, \mathsf{ECDSA}\}$ and let $\Pi^{\mathsf{SIG}}_{\mathsf{AS}} := (\mathsf{pSign}, \mathsf{pVf}, \mathsf{Adapt}, \mathsf{Ext})$ be a secure adaptor signature scheme with respect to a strongly unforgeable signature scheme $\Pi^{\mathsf{SIG}}_{\mathsf{DS}} := (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ and a hard dlog relation R. Let $\Gamma^{\mathsf{SIG}}_{\mathsf{JKGen}}, \Gamma_{\mathsf{AdpSg}}$ be UC secure 2PC protocols for computing JKGen and pSign, respectively. Let Π_{VTD} be a timed private and sound verifiable timed dlog scheme. Then the atomic swap protocol described in Figure 5 with access to $(\mathcal{F}_{\mathbb{B}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{clock}})$, UC-realizes the functionality $\mathcal{F}_{\mathsf{swap}}$.

E. *Cross-Curve Swaps*

The protocol described in Figure 5 assumes that the Schnorr/ECDSA signatures used for spending different coins are implemented on the same curve. We can easily extend our protocol to a case where all of the $n + \tilde{n}$ coins are from transaction schemes using Schnorr/ECDSA signatures but are implemented on different curves, possibly over groups of different orders. In such a scenario we have for $i \in [n]$, coin $v_i^{(0)}$ is in a currency whose transaction scheme uses $\mathsf{SIG} \in \{\mathsf{Schnorr}, \mathsf{ECDSA}\}$ implemented in the group $\mathbb{G}_i$ with generator G_i and order q_i. Similarly, for $k \in [\tilde{n}]$, coin $v_k^{(1)}$ is in a currency whose transaction scheme uses $\mathsf{SIG} \in \{\mathsf{Schnorr}, \mathsf{ECDSA}\}$ implemented in the group $\tilde{\mathbb{G}}_k$ with generator $\tilde{G}_k$ and order $\tilde{q}_k$.

It is not hard to see that the only bridge that connects the different signature schemes is the hard instance Y and as soon as the signature schemes are defined over different groups, it becomes unclear how to sample Y. The natural solution for this problem is to define a different Y_i for each signature as $Y_i = G_i^y$ (or $\tilde{Y}_k = \tilde{G}_k^y$, respectively), where G_i is the generator of the i-th group. At this point, we can compute the i-th pre-signature with respect to the corresponding instance Y_i, instead of using a fixed Y. Since the witness y (i.e., the discrete logarithm) is identical for all Y_i, correctness is preserved. However, nothing prevents P_0 from diverging from the protocol, which is the reason why we also need to include a Non-Interactive Zero Knowledge (NIZK) [64] proof to certify that all instances $(Y_1, \ldots, Y_n, \tilde{Y}_1, \ldots, \tilde{Y}_{\tilde{n}})$ have the same discrete logarithm. Fortunately, this NIZK can be efficiently instantiated extending the construction from [65] to support proving of equality of discrete logarithm in $n + \tilde{n}$ groups. The rest of the protocol is the same as in Figure 5.

We now argue the security of the protocol with this modification in place. The analysis largely follows along the same lines of Theorem 5.1, with the exception that we can no longer reduce to the standard dlog problem, since we are now given access to many instances (across different groups) with the same discrete logarithm. While this problem has been (implicitly) used before, to the best of our knowledge, it has never been formalized. We call this problem the *cross-group short discrete logarithm* problem and define it below.

Definition 4 (n-Cross-Group Short Discrete Logarithm): Consider $\{\mathbb{G}_i, G_i, q_i\}_{i \in [n]}$ be n uniformly sampled groups of λ-bits orders q_i. Let $q_{i^*} = \mathsf{min}(q_1, \ldots, q_n)$. Let $y \leftarrow\$ \mathbb{Z}_{q_{i^*}}$. Then for all PPT adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr\left[\mathcal{A}(\{\mathbb{G}_i, G_i, q_i, G_i^y\}_{i \in [n]}) = y\right] = \mathsf{negl}(\lambda).$$

Observe that y cannot be sampled to be perfectly uniform over all groups $(\mathbb{Z}_{q_1}, \ldots, \mathbb{Z}_{q_n})$, since the groups may have different orders, therefore we sample it uniformly from $\mathbb{Z}_{q_{i^*}}$, where q_{i^*} is the smallest prime. The work of Corrigan-Gibbs and Kogan [66] shows that solving the short exponent discrete logarithm problem is as hard as solving the standard discrete logarithm problem (in a group of order q_{i^*}) in the generic group model. That is, if $y \leftarrow\$ \mathbb{Z}_{q_{i^*}}$ then the running time of the best known generic algorithm to compute y given G_i^y (for some $i \neq i^*$) is $O(\sqrt{q_{i^*}})$. We conjecture that this equivalence is still true even when given all $(G_1^y, \ldots, G_n^y)$. This generalization is a natural analogue of the computational Diffie-Hellman [67] problem across groups, where different group elements are replaced with different generators of different groups. This assumption was implicit in prior works [68].

VI. Performance Analysis

We first evaluate each building block of our protocol separately, followed by the atomic swap protocol (Section V-D).

Global input: ($\mathbb{G}$, G, q, v, pk, $\mathbf{T}$), *Party U_0's input*: sk

Parties U_0 and U_1 do the following:

1) Execute the 2PC protocol $\Gamma^{\mathsf{SIG}}_{\mathsf{JKGen}}$ with with common input $(\mathbb{G}, G, q)$. Party U_0 receives $(\mathsf{tsk}_0, \mathsf{tpk})$ as output while U_1 receives $(\mathsf{tsk}_1, \mathsf{tpk})$.
2) Party U_1 generates $(C, \pi) \leftarrow \Pi_{\mathsf{VTD}}.\mathsf{Commit}(\mathsf{tsk}_1, \mathbf{T})$ and sends (C, π) to U_0.
3) Party U_0 does the following:
   - Checks if $\mathsf{SIG} = \mathsf{Schnorr}$. If so it further checks if $\Pi_{\mathsf{VTD}}.\mathsf{Verify}\left(\mathsf{tpk}/\left(G^{\mathsf{tsk}_0}\right), C, \pi\right) = 1$, and aborts otherwise.
   - Instead if $\mathsf{SIG} = \mathsf{ECDSA}$, it further checks if $\Pi_{\mathsf{VTD}}.\mathsf{Verify}\left((\mathsf{tpk})^{\mathsf{tsk}_0^{-1}}, C, \pi\right) = 1$, and aborts otherwise.
   - It generates $tx_{\mathsf{frz}} := tx\left(pk, \mathsf{tpk}, v\right)$ and a signature $\sigma_{\mathsf{frz}} \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Sign}^{\mathsf{SIG}}\left(sk, tx_{\mathsf{frz}}\right)$. It posts $\left(tx_{\mathsf{frz}}, \sigma_{\mathsf{frz}}\right)$ on $\mathbb{B}$.
   - It starts solving $\Pi_{\mathsf{VTD}}.\mathsf{ForceOp}\left(C\right)$.
4) Party U_0 returns $(tx_{\mathsf{frz}}, \sigma_{\mathsf{frz}}, \mathsf{tpk}, \mathsf{tsk}_0, C, \pi)$ and party U_1 returns $(\mathsf{tpk}, \mathsf{tsk}_1)$ as output.

Fig. 4: Freeze Protocol for $\mathsf{SIG} = \{\mathsf{Schnorr}, \mathsf{ECDSA}\}$

A. Verifiable Time DLog

We developed a prototype C implementation of Π_{VTD} [34] to demonstrate the feasibility of our construction. The implementation encompasses the Commit and Verify algorithms, simulating thus the functionality that would be carried out by the atomic swap participants during a successful swap. We omit the Setup algorithm as this can be pre-computed and shared across several instances of atomic swaps, and the ForceOp algorithm, as it is only executed in case the coins are not swapped and the running time is pre-defined by the timing hardness $\mathbf{T}$ [48]. For the time-lock puzzles, we leveraged the implementation available in [69].

Computation time. We evaluated the VTD construction Π_{VTD} for different values of the statistical parameter n. We set the threshold to $t = n/2$ (not the same as $\mathbf{T}$ the hiding parameter in Π_{VTD}) as required for soundness of the VTD [34]. We obtain the results shown in Table II. We observe that even with the value $n = 256$ (possibly too high to be used in practice), the overall running time is below 1 second, which is even below the block confirmation time of virtually all cryptocurrencies today that is in the order of several minutes. The practical advantage of VTD shows up when comparing it to verifiable timed signatures (VTS). As reported in [34], a VTS for the variants of Schnorr and ECDSA requires approximately 7 seconds for the Commit algorithm and 10 seconds for Verify with a value $n = 30$ whereas VTD requires only below 0.04 seconds.

Communication overhead. Apart from the *crs* and public parameters of the underlying time-lock puzzle, the overhead imposed by VTD is of (i) n group elements for verifiability of Shamir secret sharing; (ii) a total of $4 \cdot n$ group elements for the time-lock puzzles and related proofs; and (iii) a total of $t-1$ pairs of two group elements (x, r) for the cut and choose style proof. In summary, each party needs to store $5 \cdot n + (t-1) \cdot n$ values of $\mathbb{Z}_q$ to handle a VTD.

TABLE II: Computation time of Π_{VTD} (in seconds).

n	t	$\Pi_{\mathsf{VTD}}.\mathsf{Commit}$	$\Pi_{\mathsf{VTD}}.\mathsf{Verify}$
32	16	0.04	0.03
64	32	0.041	0.033
128	64	0.091	0.076
256	128	0.236	0.255

B. 2-Party Protocols for Adaptor Signatures

Here, we have used the implementation of the 2 party computation protocol of adaptor signatures available at [70].

Computation time. We evaluated both the Schnorr and the ECDSA implementations. We observe that the joint pre-signature generation (Schnorr) requires 4.85 ms while that for ECDSA requires 266.30 ms. For both implementations, the pre-signature verification, witness extraction and pre-signature adaption are rather fast (i.e. < 1 ms). As expected, these results are similar to those in [46].

Communication overhead. The communication overhead is the same as reported in [46] and we report it here for completeness. In particular, the the 2PC protocol $\Gamma^{\mathsf{Schnorr}}_{\mathsf{AdpSg}}$ requires 256 bytes of communication between the two parties while its ECDSA counterpart $\Gamma^{\mathsf{ECDSA}}_{\mathsf{AdpSg}}$ requires 416 bytes.

C. Atomic Swap Protocol

For the sake of clarity we chose not to include several optimisations to the description in Figure 5. We discuss them below and also incorporate them in our implementation.

Implementation-level Optimizations. In the setup phase (Figure 5), we create a new VTD commitment for each coin that is being swapped, requiring thus $n + \tilde{n}$ VTD commitments for both parties put together. However, in practice, one can have the same functionality while each party has to create and force open only a single VTD commitment.

To do this, assume for a moment that fungibility of the transactions is not a problem. Then each party could create a single key pair, embed the corresponding secret key in the VTD and use the corresponding public key repeatedly as an address for each of the coins that it owns (either n or $\tilde{n}$). In order to gain fungibility back (i.e., make each public key look random to the eyes of the blockchain observer), the two parties carry out a coin toss protocol to generate a common randomness r and generate each public key with respect to it. For instance, consider that (pk, sk) was the key generated by P_0. The i-th public key can be computed as $pk^{\mathsf{H}(r||i)}$. Note that given the original public key pk and the randomness r, party P_1 can

Global input: $\left\{v_i^{(0)}, pk_i^{(0)}\right\}_{i\in[n]}, \left\{v_k^{(1)}, pk_k^{(1)}\right\}_{k\in[\tilde{n}]}, \mathbf{T}_0, \mathbf{T}_1, \Delta$. Here $\mathbf{T}_0, \Delta \in \mathbb{N}$ and $\mathbf{T}_1 = \mathbf{T}_0 - \Delta$.
Party P_0's input: $\left\{sk_i^{(0)}\right\}_{i\in[n]}$, *Party P_1's input:* $\left\{sk_i^{(1)}\right\}_{k\in[\tilde{n}]}$

Swap Setup Phase - Freezing coins

Parties P_0 and P_1 freeze the coins that they want to swap by doing the following:

1) For $i \in [n]$, party P_0 plays the role of U_0 and party P_1 plays the role of U_1 in the freeze algorithm (Figure 4). Specifically,
   - Party P_0's input: $\left(v_i^{(0)}, pk_i^{(0)}, sk_i^{(0)}, \mathbf{T}_0\right)$, and party P_1's input: $\left(v_i^{(0)}, pk_i^{(0)}, \mathbf{T}_0\right)$.
   - Party P_0's output: $\left(tx_{\mathsf{frz},i}^{(0)}, \sigma_{\mathsf{frz},i}^{(0)}, pk_i^{(01)}, sk_{0,i}^{(01)}, C_i^{(0)}, \pi_i^{(0)}\right)$, and party P_1's output: $\left(pk_i^{(01)}, sk_{1,i}^{(01)}\right)$.
2) For $k \in [\tilde{n}]$, party P_0 plays the role of U_1 and party P_1 plays the role of U_0 in the freeze algorithm (Figure 4). Specifically,
   - Party P_0's input: $\left(v_k^{(1)}, pk_k^{(1)}, \mathbf{T}_1\right)$, and party P_1's input: $\left(v_k^{(1)}, pk_k^{(1)}, sk_k^{(1)}, \mathbf{T}_1\right)$.
   - Party P_0's output: $\left(pk_k^{(10)}, sk_{0,k}^{(10)}\right)$, and party P_1's output: $\left(tx_{\mathsf{frz},k}^{(1)}, \sigma_{\mathsf{frz},k}^{(1)}, pk_k^{(10)}, sk_{1,k}^{(10)}, C_k^{(1)}, \pi_k^{(1)}\right)$.

Swap Lock Phase

1) Party P_0 picks $(Y, y) \in R$ using $\mathsf{GenR}(1^\lambda)$ and sends Y to party P_1.
2) Parties then setup the transactions doing the following:
   - Party P_1 generates $\left(pk_{\mathsf{swpd},i}^{(1)}, sk_{\mathsf{swpd},i}^{(1)}\right) \leftarrow \Pi_{\mathsf{DS}}^{\mathsf{SIG}}.\mathsf{KGen}(1^\lambda)$ for $i \in [n]$.
   - Party P_1 generates swap transaction $tx_{\mathsf{swp},i}^{(1)} := tx\left(pk_i^{(01)}, pk_{\mathsf{swpd},i}^{(1)}, v_i^{(0)}\right)$ for $i \in [n]$ and sends it to party P_0.
   - Party P_0 generates $\left(pk_{\mathsf{swpd},k}^{(0)}, sk_{\mathsf{swpd},k}^{(0)}\right) \leftarrow \Pi_{\mathsf{DS}}^{\mathsf{SIG}}.\mathsf{KGen}(1^\lambda)$ for $k \in [\tilde{n}]$.
   - Party P_0 generates swap transaction $tx_{\mathsf{swp},k}^{(0)} := tx\left(pk_k^{(10)}, pk_{\mathsf{swpd},k}^{(0)}, v_k^{(1)}\right)$ and sends it to party P_1.
3) For $i \in [n]$, parties P_0 and P_1 run a 2PC protocol $\Gamma_{\mathsf{AdpSg}}^{\mathsf{SIG}}$ that does the following:
   - The 2PC protocol takes as common input $\left(tx_{\mathsf{swp},i}^{(1)}, Y\right)$ from the parties, and as private input $sk_{0,i}^{(01)}$ from party P_0, and as private input $sk_{1,i}^{(01)}$ from party P_1.
   - If $\mathsf{SIG} = \mathsf{Schnorr}$, compute $sk_i^{(01)} := \left(sk_{0,i}^{(01)} + sk_{1,i}^{(01)}\right) \bmod q$, and if $\mathsf{SIG} = \mathsf{ECDSA}$, compute $sk_i^{(01)} := \left(sk_{0,i}^{(01)} \cdot sk_{1,i}^{(01)}\right) \bmod q$.
   - It computes $\tilde{\sigma}_{\mathsf{swp},i}^{(1)} \leftarrow \Pi_{\mathsf{AS}}^{\mathsf{SIG}}.\mathsf{pSign}\left(sk_i^{(01)}, tx_{\mathsf{swp},i}^{(1)}, Y\right)$ and outputs $\tilde{\sigma}_{\mathsf{swp},i}^{(1)}$ first to P_0 and then to P_1.
   - Both parties check if $\Pi_{\mathsf{AS}}^{\mathsf{SIG}}.\mathsf{pVf}\left(pk_i^{(01)}, tx_{\mathsf{swp},i}^{(1)}, Y; \tilde{\sigma}_{\mathsf{swp},i}^{(1)}\right) = 1$, and abort otherwise.
   - In the end, both parties P_0 and P_1 obtain $\left\{\tilde{\sigma}_{\mathsf{swp},i}^{(1)}\right\}_{i\in[n]}$.
4) After the above step is successful, for all $k \in [\tilde{n}]$, party P_0 and P_1 run the 2PC protocol $\Gamma_{\mathsf{AdpSg}}^{\mathsf{SIG}}$ that does the following:
   - The 2PC protocol takes as common input $\left(tx_{\mathsf{swp},k}^{(0)}, Y\right)$ from the parties, and as private input $sk_{0,k}^{(10)}$ from party P_0, and as private input $sk_{1,k}^{(10)}$ from party P_1.
   - If $\mathsf{SIG} = \mathsf{Schnorr}$, compute $sk_k^{(10)} := \left(sk_{0,k}^{(10)} + sk_{1,k}^{(10)}\right) \bmod q$, and, if $\mathsf{SIG} = \mathsf{ECDSA}$, compute $sk_k^{(10)} := \left(sk_{0,k}^{(10)} \cdot sk_{1,k}^{(10)}\right) \bmod q$
   - It computes $\tilde{\sigma}_{\mathsf{swp},k}^{(0)} \leftarrow \Pi_{\mathsf{AS}}^{\mathsf{SIG}}.\mathsf{pSign}\left(sk_k^{(10)}, tx_{\mathsf{swp},k}^{(0)}, Y\right)$ and outputs $\tilde{\sigma}_{\mathsf{swp},k}^{(0)}$ to both parties first to P_0 and then to P_1.
   - Both parties check if $\Pi_{\mathsf{AS}}^{\mathsf{SIG}}.\mathsf{pVf}\left(pk_k^{(10)}, tx_{\mathsf{swp},k}^{(0)}, Y;\ \tilde{\sigma}_{\mathsf{swp},k}^{(0)}\right) = 1$, and abort otherwise.
   - In the end, both parties P_0 and P_1 obtain $\left\{\tilde{\sigma}_{\mathsf{swp},k}^{(0)}\right\}_{k\in[\tilde{n}]}$.

Fig. 5: Atomic Swap protocol run between parties P_0 and P_1 where $\mathsf{SIG} = \{\mathsf{Schnorr}, \mathsf{ECDSA}\}$

verify that the i-th key is correctly generated. While knowing the secret key sk, the party can derive $sk_i := sk \cdot \mathsf{H}(r||i)$.

We can also parallelise the computation of different instances of the 2PC protocol $\Gamma_{\mathsf{AdpSg}}^{\mathsf{SIG}}$. In practice, it would be possible to use the multi-core architecture available at current commodity machines to perform this paralellization for moderate amount of coin swaps.

Performance. With the aforementioned implementation-level optimizations, we do a back-of-the-envelope calculation of the performance of our protocol by extracting the number

Swap Complete Phase

1) For all $k \in [\tilde{n}]$ party P_0 computes $\sigma^{(0)}_{\mathsf{swp},k} \leftarrow \Pi^{\mathsf{SIG}}_{\mathsf{AS}}.\mathsf{Adapt}\left(\tilde{\sigma}^{(0)}_{\mathsf{swp},k}, y\right)$, and posts $\left(tx^{(0)}_{\mathsf{swp},k}, \sigma^{(0)}_{\mathsf{swp},k}\right)$ on the blockchain $\mathbb{B}$.
2) Party P_1 picks $j \in [\tilde{n}]$, and does the following:
   - Compute $y \leftarrow \Pi^{\mathsf{SIG}}_{\mathsf{AS}}.\mathsf{Ext}\left(\sigma^{(0)}_{\mathsf{swp},j}, \tilde{\sigma}^{(0)}_{\mathsf{swp},j}, Y\right)$, and for all $i \in [n]$, compute $\sigma^{(1)}_{\mathsf{swp},i} \leftarrow \Pi^{\mathsf{SIG}}_{\mathsf{AS}}.\mathsf{Adapt}\left(\tilde{\sigma}^{(1)}_{\mathsf{swp},i}, y\right)$
   - It posts $\left(tx^{(1)}_{\mathsf{swp},i}, \sigma^{(1)}_{\mathsf{swp},i}\right)$ for all $i \in [n]$ on the blockchain $\mathbb{B}$ (corresponding to currency of i-th coin).

Swap Timeout Phase

1) If party P_0 fails to post $\left(tx^{(0)}_{\mathsf{swp},j}, \sigma^{(0)}_{\mathsf{swp},j}\right)$ for any $j \in [\tilde{n}]$ on chain before time $\mathbf{T}_1$, party P_1 does the following:
   - Finish computing $sk^{(10)}_{0,j} \leftarrow \Pi_{\mathsf{VTD}}.\mathsf{ForceOp}\left(C^{(1)}_j\right)$.
   - If $\mathsf{SIG} = \mathsf{Schnorr}$, compute $sk^{(10)}_j := \left(sk^{(10)}_{0,j} + sk^{(10)}_{1,j}\right) \mod q$, else if $\mathsf{SIG} = \mathsf{ECDSA}$, compute $sk^{(10)}_j := \left(sk^{(10)}_{0,j} \cdot sk^{(10)}_{1,j}\right) \mod q$.
   - Generate fresh key pairs $\left(pk^{(1)}_{\mathsf{rfnd},i}, sk^{(1)}_{\mathsf{rfnd},j}\right) \leftarrow \Pi^{\mathsf{SIG}}_{\mathsf{DS}}.\mathsf{KGen}(1^\lambda)$, and generate redeem transaction $tx^{(1)}_{\mathsf{rfnd},j} := tx\left(pk^{(10)}_j, pk^{(1)}_{\mathsf{rfnd},j}, v^{(1)}_j\right)$.
   - Generate a signature $\sigma^{(1)}_{\mathsf{rfnd},j} \leftarrow \Pi^{\mathsf{SIG}}_{\mathsf{DS}}.\mathsf{Sign}\left(sk^{(10)}_j, tx^{(1)}_{\mathsf{rfnd},j}\right)$ on the redeem transaction.
   - Reclaim the $v^{(1)}_j$ coins by posting $\left(tx^{(1)}_{\mathsf{rfnd},j}, \sigma^{(1)}_{\mathsf{rfnd},j}\right)$ on the blockchain $\mathbb{B}$ (corresponding to currency of j-th coin).
2) Similarly, if party P_1 fails to post a valid transaction-signature pair $\left(tx^{(1)}_{\mathsf{swp},j}, \sigma^{(1)}_{\mathsf{swp},j}\right)$ for any $j \in [n]$ on chain before time $\mathbf{T}_0$, party P_0 follows steps analogous to above and reclaims the $v^{(0)}_j$ coins by posting $\left(tx^{(0)}_{\mathsf{rfnd},j}, \sigma^{(0)}_{\mathsf{rfnd},j}\right)$ on the blockchain $\mathbb{B}$.

Fig. 6: Swap complete and timeout phase of the atomic swap protocol between P_0 and P_1 where $\mathsf{SIG} = \{\mathsf{Schnorr}, \mathsf{ECDSA}\}$

of invocations to the underlying building blocks, namely (i) adaptor signatures $\Pi^{\mathsf{SIG}}_{\mathsf{AS}}$; (ii) VTD commit and verify; and (iii) standard digital signature scheme $\Pi_{\mathsf{DS}}.\mathsf{Sign}$. We show our results in Table III. Overall we observe that the protocol uses several operations only once. The costliest operation is $\Gamma^{\mathsf{SIG}}_{\mathsf{AdpSg}}$ since it requires interaction between the two participants. Yet, each instance of $\Gamma^{\mathsf{SIG}}_{\mathsf{AdpSg}}$ requires only 267ms for ECDSA and 5ms for Schnorr as reported in Section VI-B. The rest of operations that need to be repeated $n + \tilde{n}$ times can be performed locally by each participant.

Comparison with HTLC. We have taken the HTLC implementation in Solidity available at [71] and calculated the gas costs to compute the three operations required in a swap: (i) create the HTLC contract; (ii) redeem the contract with a valid hash preimage; and (iii) refund the contract in case the timeout expires. The gas costs are shown in Table IV.

We observe that in the case of universal swap, we only require a standard transfer of ETH between two accounts, an inexpensive operation (since it is the basic one) in Ethereum. In contrast, the HTLC-based operations require between 2.7x and 6.3x more gas.

TABLE III: Operations required in our atomic swap protocol.

Op type	**Setup Phase**	**Lock Phase**	**Complete Phase**
Joint key generation	1	0	0
Key generation	0	$(n + \tilde{n})$	0
Signing	$(n + \tilde{n})$	1	0
Signature verify	0	1	0
VTD commit	2	0	0
VTD verify	2	0	0
Joint pre-signing	0	$(n + \tilde{n})$	0
Pre-signature verify	0	$(n + \tilde{n})$	0
Pre-signature adapt	0	0	$(n + \tilde{n})$
Witness extract	0	0	1

TABLE IV: Gas cost of swaps in Ethereum

	Create	Redeem	Refund
HTLC	134320	79752	58065
Our n-to-$\tilde{n}$ Schnorr/ECDSA Swap	21000	21000	21000

Apart from higher gas cost, the HTLC-based approach also requires a higher transaction size, since it needs to store the parameters of the contract (i.e., hash value, receiver and timeout). This additional overhead not only hinders the scalability of the underlying blockchain as the on-chain space is limited, but also the fungibility of the swap, as it is trivial for an observer to match two coins swapped as they share the same hash value.

In summary, although the swap functionality would be easily implementable in a smart contract such as HTLC, our protocol is preferable due to smaller on-chain cost both in gas and transaction size, fungibility guarantees and backwards compatibility with virtually all blockchains available today.

VII. Conclusion

In this work we investigate the problem of atomic swaps of multiple assets across all cryptocurrencies. We propose a *universal protocol* supporting n-to-$\tilde{n}$ swaps. without relying on any special scripts from the blockchain apart from the ability to verify signatures. Following the outline of the generic protocol we give a highly efficient protocol for the specific cases where the transactions are signed with ECDSA or Schnorr signatures. We also explore extensions to multi-party cyclic swaps and cross-curve swaps. In terms of future work, we aim at developing efficient solutions for other signature schemes with different properties (e.g., post-quantum security or aggregatable signatures).

REFERENCES

[1] A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. Moreno-Sanchez, A. Kiayias, and W. J. Knottenbelt, *Sok: Communication across distributed ledgers*, Cryptology ePrint Archive, Report 2019/1128, https://eprint.iacr.org/2019/1128, 2019.

[2] J. Kwon and E. Buchman, *Cosmos: A network of distributed ledgers*, https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md.

[3] D. J. Hosp, T. Hoenisch, and P. Kittiwongsunthorn, *Comit - cryptographically-secure off-chain multi-asset instant transaction network*, 2018. arXiv: 1810.02174 [cs.DC].

[4] S. Thomas and E. Schwartz, *Interledger: A protocol for interledger payments*, https://interledger.org/interledger.pdf.

[5] G. Wood, *Polkadot: Vision for a heterogenous multi-chain framework*, https://polkadot.network/PolkaDotPaper.pdf.

[6] S. Dziembowski, L. Eckey, and S. Faust, "FairSwap: How to fairly exchange digital goods," in *ACM CCS 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds., ACM Press, Oct. 2018, pp. 967–984. DOI: 10.1145/3243734.3243857.

[7] M. Herlihy, *Atomic cross-chain swaps*, 2018. arXiv: 1801.09515 [cs.DC].

[8] M. Herlihy, B. Liskov, and L. Shrira, "Cross-chain deals and adversarial commerce," *arXiv preprint arXiv:1905.09743*, 2019.

[9] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds., ACM Press, 2017, pp. 455–471. DOI: 10.1145/3133956.3134096.

[10] C. Baum, B. David, and T. Frederiksen, *P2dex: Privacy-preserving decentralized cryptocurrency exchange*, Cryptology ePrint Archive, Report 2021/283, https://eprint.iacr.org/2021/283, 2021.

[11] I. Bentov, Y. Ji, F. Zhang, Y. Li, X. Zhao, L. Breidenbach, P. Daian, and A. Juels, *Tesseract: Real-time cryptocurrency exchange using trusted hardware*, Cryptology ePrint Archive, Report 2017/1153, https://eprint.iacr.org/2017/1153, 2017.

[12] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution," in *2019 IEEE European Symposium on Security and Privacy (EuroS P)*, 2019, pp. 142–157. DOI: 10.1109/EuroSP.2019.00020.

[13] J. Van Bulck, D. Oswald, E. Marin, A. Aldoseri, F. D. Garcia, and F. Piessens, "A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19, London, United Kingdom: Association for Computing Machinery, 2019, 1741–1758, ISBN: 9781450367479. DOI: 10.1145/3319535.3363206. [Online]. Available: https://doi.org/10.1145/3319535.3363206.

[14] *What is atomic swap and how to implement it*, https://www.axiomadev.com/blog/what-is-atomic-swap-and-how-to-implement-it/.

[15] *Submarine swap in lightning network*, https://wiki.ion.radar.tech/tech/research/submarine-swap.

[16] *Uniswap*, https://uniswap.org/whitepaper.pdf.

[17] *Raiden network*, https://raiden.network/.

[18] M. Campanelli, R. Gennaro, S. Goldfeder, and L. Nizzardo, "Zero-knowledge contingent payments revisited: Attacks and payments for services," in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds., ACM Press, 2017, pp. 229–243. DOI: 10.1145/3133956.3134060.

[19] S. Bursuc and S. Kremer, "Contingent payments on a public ledger: Models and reductions for automated verification," in *ESORICS 2019, Part I*, K. Sako, S. Schneider, and P. Y. A. Ryan, Eds., ser. LNCS, vol. 11735, Springer, Heidelberg, Sep. 2019, pp. 361–382. DOI: 10.1007/978-3-030-29959-0_18.

[20] I. Tsabary, M. Yechieli, A. Manuskin, and I. Eyal, *Mad-htlc: Because htlc is crazy-cheap to attack*, 2021. arXiv: 2006.12031 [cs.CR].

[21] tinyurl.com/2z59vhys.

[22] R. W. F. Lai, V. Ronge, T. Ruffing, D. Schröder, S. A. K. Thyagarajan, and J. Wang, "Omniring: Scaling private payments without trusted setup," in *ACM CCS 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds., ACM Press, Nov. 2019, pp. 31–48. DOI: 10.1145/3319535.3345655.

[23] A. Poelstra, *Mimblewimble*, https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf.

[24] D. Schwartz, N. Youngs, A. Britto, *et al.*, "The ripple protocol consensus algorithm," *Ripple Labs Inc White Paper*, vol. 5, no. 8, 2014.

[25] D. Mazieres, "The stellar consensus protocol: A federated model for internet-level consensus," *Stellar Development Foundation*, vol. 32, 2015.

[26] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, May 2014, pp. 459–474. DOI: 10.1109/SP.2014.36.

[27] https://thecharlatan.ch/Monero-Unlock-Time-Privacy/.

[28] https://github.com/mimblewimble/grin/issues/25.

[29] https://github.com/zcash/zcash/issues/344.

[30] https://medium.com/\spacefactor\@m{}YcashFoundation/announcing-ycash-the-first-friendly-fork-of-the-zcash-blockchain-ac386ed6368c.

[31] R. Canetti, "Security and composition of multiparty cryptographic protocols," *Journal of Cryptology*, vol. 13, no. 1, pp. 143–202, Jan. 2000. DOI: 10.1007/s001459910006.

[32] D. Boneh and M. Naor, "Timed commitments," in *CRYPTO 2000*, M. Bellare, Ed., ser. LNCS, vol. 1880, Springer, Heidelberg, Aug. 2000, pp. 236–254. DOI: 10.1007/3-540-44598-6_15.

[33] J. A. Garay and M. Jakobsson, "Timed release of standard digital signatures," in *FC 2002*, M. Blaze, Ed., ser. LNCS, vol. 2357, Springer, Heidelberg, Mar. 2003, pp. 168–182.

[34] S. A. K. Thyagarajan, A. Bhat, G. Malavolta, N. Döttling, A. Kate, and D. Schröder, "Verifiable timed signatures made practical," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20, Virtual Event, USA: Association for Computing Machinery, 2020, 1733–1750, ISBN: 9781450370899. DOI: 10.1145/3372297.3417263. [Online]. Available: https://doi.org/10.1145/3372297.3417263.

[35] S. A. K. Thyagarajan, G. Malavolta, F. Schmidt, and D. Schröder, *Paymo: Payment channels for monero*, Cryptology ePrint Archive, Report 2020/1441, https://eprint.iacr.org/2020/1441, 2020.

[36] P. Moreno-Sanchez, A. Blue, D. V. Le, S. Noether, B. Goodell, and A. Kate, "Dlsag: Non-interactive refund transactions for interoperable payment channels in monero," in *Financial Cryptography and Data Security*, J. Bonneau and N. Heninger, Eds., Cham: Springer International Publishing, 2020, pp. 325–345.

[37] *Tx1*, https://tinyurl.com/y3n9hvqb.

[38] *Tx2*, https://tinyurl.com/y4yqoj2c.

[39] *Tx3*, https://tinyurl.com/y3m82xg5.

[40] *Tx4*, https://tinyurl.com/y3h6rh3f.

[41] *Tx5*, https://tinyurl.com/y2fsa3ak.

[42] M. Zhandry, "The magic of ELFs," in *CRYPTO 2016, Part I*, M. Robshaw and J. Katz, Eds., ser. LNCS, vol. 9814, Springer, Heidelberg, Aug. 2016, pp. 479–508. DOI: 10.1007/978-3-662-53018-4_18.

[43] B. Schoenmakers, M. Veeningen, and N. de Vreede, "Trinocchio: Privacy-preserving outsourcing by distributed verifiable computation," in *ACNS 16*, M. Manulis, A.-R. Sadeghi, and S. Schneider, Eds., ser. LNCS, vol. 9696, Springer, Heidelberg, Jun. 2016, pp. 346–366. DOI: 10.1007/978-3-319-39555-5_19.

[44] *Engineers demonstrate zcash/bitcoin atomic swaps*, https://news.bitcoin.com/engineers-demonstrate-zcashbitcoin-atomic-swaps/.

[45] J. Liu, T. Jager, S. A. Kakvi, and B. Warinschi, "How to build time-lock encryption," *Des. Codes Cryptography*, vol. 86, no. 11, 2549–2586, Nov. 2018, ISSN: 0925-1022. DOI: 10.1007/s10623-018-0461-x. [Online]. Available: https://doi.org/10.1007/s10623-018-0461-x.

[46] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," in *NDSS 2019*, The Internet Society, Feb. 2019.

[47] L. Aumayr, O. Ersoy, A. Erwig, S. Faust, K. Hostakova, M. Maffei, P. Moreno-Sanchez, and S. Riahi, *Generalized bitcoin-compatible channels*, Cryptology ePrint Archive, Report 2020/476, https://eprint.iacr.org/2020/476, 2020.

[48] S. A. K. Thyagarajan, A. Bhat, G. Malavolta, N. Döttling, A. Kate, and D. Schröder, "Verifiable timed signatures made practical," in *ACM CCS 20*, ACM Press, 2020, pp. 1733–1750. DOI: 10.1145/3372297.3417263.

[49] "Personal communication," To Appear at ACM CCS 2021.

[50] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, IEEE, 2001, pp. 136–145.

[51] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in *TCC 2007*, S. P. Vadhan, Ed., ser. LNCS, vol. 4392, Springer, Heidelberg, Feb. 2007, pp. 61–85. DOI: 10.1007/978-3-540-70936-7_4.

[52] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal on Computing*, vol. 17, no. 2, pp. 281–308, Apr. 1988.

[53] M. Backes and D. Hofheinz, "How to break and repair a universally composable signature functionality," in *ISC 2004*, K. Zhang and Y. Zheng, Eds., ser. LNCS, vol. 3225, Springer, Heidelberg, Sep. 2004, pp. 61–72.

[54] J. Katz, U. Maurer, B. Tackmann, and V. Zikas, "Universally composable synchronous computation," in *TCC 2013*, A. Sahai, Ed., ser. LNCS, vol. 7785, Springer, Heidelberg, Mar. 2013, pp. 477–498. DOI: 10.1007/978-3-642-36594-2_27.

[55] S. Dziembowski, L. Eckey, S. Faust, J. Hesse, and K. Hostáková, "Multi-party virtual state channels," in *EUROCRYPT 2019, Part I*, Y. Ishai and V. Rijmen, Eds., ser. LNCS, vol. 11476, Springer, Heidelberg, May 2019, pp. 625–656. DOI: 10.1007/978-3-030-17653-2_21.

[56] L. Aumayr, O. Ersoy, A. Erwig, S. Faust, K. Hostakova, M. Maffei, P. Moreno-Sanchez, and S. Riahi, "Generalized bitcoin-compatible channels.," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 476, 2020.

[57] A. R. Choudhuri, M. Green, A. Jain, G. Kaptchuk, and I. Miers, "Fairness in an unfair world: Fair multiparty computation from public bulletin boards," in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds., ACM Press, 2017, pp. 719–728. DOI: 10.1145/3133956.3134092.

[58] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," in *CRYPTO 2014, Part II*, J. A. Garay and R. Gennaro, Eds., ser. LNCS, vol. 8617,

Springer, Heidelberg, Aug. 2014, pp. 421–439. DOI: 10.1007/978-3-662-44381-1_24.

[59] R. L. Rivest, A. Shamir, and D. A. Wagner, “Time-lock puzzles and timed-release crypto,” Cambridge, MA, USA, Tech. Rep., 1996.

[60] A. Shamir, “How to share a secret,” *Communications of the Association for Computing Machinery*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[61] G. Malavolta and S. A. K. Thyagarajan, “Homomorphic time-lock puzzles and applications,” in *CRYPTO 2019, Part I*, A. Boldyreva and D. Micciancio, Eds., ser. LNCS, vol. 11692, Springer, Heidelberg, Aug. 2019, pp. 620–649. DOI: 10.1007/978-3-030-26948-7_22.

[62] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, “Secure distributed key generation for discrete-log based cryptosystems,” in *EUROCRYPT’99*, J. Stern, Ed., ser. LNCS, vol. 1592, Springer, Heidelberg, May 1999, pp. 295–310. DOI: 10.1007/3-540-48910-X_21.

[63] Y. Lindell, “Fast secure two-party ECDSA signing,” in *CRYPTO 2017, Part II*, J. Katz and H. Shacham, Eds., ser. LNCS, vol. 10402, Springer, Heidelberg, Aug. 2017, pp. 613–644. DOI: 10.1007/978-3-319-63715-0_21.

[64] A. De Santis, S. Micali, and G. Persiano, “Non-interactive zero-knowledge proof systems,” in *Conference on the Theory and Application of Cryptographic Techniques*, Springer, 1987, pp. 52–72.

[65] S. Noether, *Discrete logarithm equality across groups*, https://web.getmonero.org/zh-cn/resources/research-lab/pubs/MRL-0010.pdf.

[66] H. Corrigan-Gibbs and D. Kogan, “The discrete-logarithm problem with preprocessing,” in *EUROCRYPT 2018, Part II*, J. B. Nielsen and V. Rijmen, Eds., ser. LNCS, vol. 10821, Springer, Heidelberg, 2018, pp. 415–447. DOI: 10.1007/978-3-319-78375-8_14.

[67] W. Diffie and M. E. Hellman, “New directions in cryptography,” *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[68] J. Gugger, *Bitcoin-monero cross-chain atomic swap*, Cryptology ePrint Archive, Report 2020/1126, https://eprint.iacr.org/2020/1126, 2020.

[69] A. Bhat, *Liblhtlp - c library implementing linearly homomorphic time lock puzzles*, https://github.com/verifiable-timed-signatures/liblhtlp.

[70] E. Tairi, *Anonymous atomic locks (a2l)*, https://github.com/etairi/A2L.

[71] Github Project, https://github.com/chatch/hashed-timelock-contract-ethereum.

[72] TierNolan, *Atomic swap - bitcoin wiki*, https://en.bitcoin.it/wiki/Atomic_swap.

[73] M. Herlihy, B. Liskov, and L. Shrira, “Cross-chain deals and adversarial commerce,” *Proceedings of the VLDB Endowment*, vol. 13, no. 2, 100–113, 2019, ISSN: 2150-8097. DOI: 10.14778/3364324.3364326. [Online]. Available: http://dx.doi.org/10.14778/3364324.3364326.

[74] A. Zamyatin, D. Harz, J. Lind, P. Panayiotou, A. Gervais, and W. J. Knottenbelt, *Xclaim: Trustless, interoperable cryptocurrency-backed assets*, Cryptology ePrint Archive, Report 2018/643, https://eprint.iacr.org/2018/643, 2018.

[75] S. Dziembowski, L. Eckey, and S. Faust, *Fairswap: How to fairly exchange digital goods*, Cryptology ePrint Archive, Report 2018/740, https://eprint.iacr.org/2018/740, 2018.

[76] A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. Moreno-Sanchez, A. Kiayias, and W. J. Knottenbelt, *Sok: Communication across distributed ledgers*, Cryptology ePrint Archive, Report 2019/1128, https://eprint.iacr.org/2019/1128, 2019.

[77] S. Thyagarajan and G. Malavolta, “Lockable signatures for blockchains: Scriptless scripts for all signatures,” in *2021 2021 IEEE Symposium on Security and Privacy (SP)*, Los Alamitos, CA, USA: IEEE Computer Society, 2021, pp. 937–954. DOI: 10.1109/SP40001.2021.00065. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP40001.2021.00065.

[78] J. Camenisch, S. Krenn, and V. Shoup, “A framework for practical universally composable zero-knowledge protocols,” in *ASIACRYPT 2011*, D. H. Lee and X. Wang, Eds., ser. LNCS, vol. 7073, Springer, Heidelberg, Dec. 2011, pp. 449–467. DOI: 10.1007/978-3-642-25385-0_24.

[79] B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds., *ACM CCS 2017*, ACM Press, 2017.

APPENDIX

A. Solution to Exercise

The solution to the fun exercise posed in Section I-B is the transaction [37]. This is the swap transaction on the Bitcoin side of the protocol.

B. Related Work

We give an overview of the related work in the following.

HTLC-Based Atomic Swaps. First introduced by TierNolan [72], HTLC-based atomic swaps have constituted the core building block for several cryptocurrency transfers and exchange protocols [7], [73]. These protocols however suffer from the drawbacks on HTLC- and timelock-compatibility as described previously in this section.

Other Approaches for Atomic Swaps. Atomic swaps based on the *Atomic Multi-Hop Locks (AHML)* primitive [46] depart from HTLC-based ones in that the cryptographic condition to coordinate the swap is embedded in the creation of the digital signature that authorizes the swap, thereby departing from the need of hash functions. Yet, AMHL-based swaps as in [46] still rely on ledgers compatible with timelock functionality and support single-asset swaps. Recently, [68] have proposed an atomic swap protocol between Bitcoin and Monero that does not require hash functionality at any of the ledgers while timelock functionality is only required at one of the ledgers (i.e., in this case Bitcoin). Yet, this protocol is single-asset swap and tailored to these two ledgers.

Decentralized Exchanges. Leveraging the expressive scripting language available at cryptocurrencies like Ethereum, decentralized exchanges are implemented as a smart contract that encodes the (somewhat complex) logic to exchange multiple assets among multiple users. This approach is, however, supported only by those cryptocurrencies with expressive scripting language such as that in Ethereum and forces uses with assets in other ledgers to *migrate* their assets to Ethereum [74], [75].

Other Cross-chain Communication. Apart from the cryptocurrency transfers and exchanges, cross-chain communication is a critical component for scalability solutions such as sharding, feature extensions via sidechains, as well as bootstrapping of new systems. Although these applications are out of the scope of this paper, we find them an interesting future research direction. We refer the reader to [76] for further reading.

Scriptless Payment Channel Network. Lockable Signature was a recently introduced tool [77] useful to construct payment channel networks scriptlessly from any signature scheme. However the resulting protocol still relies on time-lock scripts from the currencies to realize payment expiry, and therefore is not universal.

C. Security Definitions of Adaptor signatures

The correctness definition of adaptor signatures is described below.

Definition 5 (Pre-signature Correctness): An adaptor signature scheme Π_{AS} satisfies pre-signature correctness if for every $\lambda \in \mathbb{N}$, every message $m \in \{0,1\}^*$ and every statement/witness pair $(Y, y) \in R$, the following holds:

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

Next, we formally define the security properties of an adaptor signature scheme.

Definition 6 (Unforgeability): An adaptor signature scheme Π_{AS} is aEUF-CMA secure if for every PPT adversary $\mathcal{A}$ there exists a negligible function negl such that:

$$\Pr\big[\mathsf{aSigForge}_{\mathcal{A},\Pi_{\mathsf{AS}}}(\lambda) = 1\big] \leq \mathsf{negl}(\lambda)$$

where the experiment $\mathsf{aSigForge}_{\mathcal{A},\Pi_{\mathsf{AS}}}$ is defined as follows:

Definition 7 (Pre-signature Adaptability): An adaptor signature scheme Π_{AS} satisfies pre-signature adaptability if for any $\lambda \in \mathbb{N}$, any message $m \in \{0,1\}^*$, any statement/witness pair $(Y, y) \in R$, any key pair $(sk, pk) \leftarrow \mathsf{KGen}(1^\lambda)$ and any pre-signature $\hat{\sigma} \leftarrow \{0,1\}^*$ with $\mathsf{pVf}(pk, m, Y, \hat{\sigma}) = 1$ we have:

$$\Pr[\mathsf{Vf}(pk, m, \mathsf{Adapt}(\hat{\sigma}, y)) = 1] = 1$$

Definition 8 (Witness Extractability): An adaptor signature scheme Π_{AS} is *witness extractable* if for every PPT adversary $\mathcal{A}$, there exists a negligible function negl such that the following holds:

$$\Pr[\mathsf{aWitExt}_{\mathcal{A},\Pi_{\mathsf{AS}}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

```
$\mathsf{aSigForge}_{\mathcal{A},\Pi_{\mathsf{AS}}}(\lambda)$
    $\mathcal{Q} := \emptyset$
    $(sk, pk) \leftarrow \mathsf{KGen}(1^\lambda)$
    $m \leftarrow \mathcal{A}^{\mathsf{Sign}\mathcal{O}(\cdot),\mathsf{pSign}\mathcal{O}(\cdot,\cdot)}(pk)$
    $(Y, y) \leftarrow \mathsf{GenR}(1^\lambda)$
    $\hat{\sigma} \leftarrow \mathsf{pSign}(sk, m, Y)$
    $\sigma \leftarrow \mathcal{A}^{\mathsf{Sign}\mathcal{O}(\cdot),\mathsf{pSign}\mathcal{O}(\cdot,\cdot)}(\hat{\sigma}, Y)$
    return $(m \notin \mathcal{Q} \wedge \mathsf{Vf}(pk, m, \sigma))$

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

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

Fig. 7: Unforgeabiltiy experiment of adaptor signatures

```
$\mathsf{aWitExt}_{\mathcal{A},\Pi_{\mathsf{AS}}}(\lambda)$
    $\mathcal{Q} := \emptyset$
    $(sk, pk) \leftarrow \mathsf{KGen}(1^\lambda)$
    $(m, Y) \leftarrow \mathcal{A}^{\mathsf{Sign}\mathcal{O}(\cdot),\mathsf{pSign}\mathcal{O}(\cdot,\cdot)}(pk)$
    $\hat{\sigma} \leftarrow \mathsf{pSign}(sk, m, Y)$
    $\sigma \leftarrow \mathcal{A}^{\mathsf{Sign}\mathcal{O}(\cdot),\mathsf{pSign}\mathcal{O}(\cdot,\cdot)}(\hat{\sigma})$
    $y' := \mathsf{Ext}(pk, \sigma, \hat{\sigma}, Y)$
    return $(m \notin \mathcal{Q} \wedge (Y, y') \notin R$
     $\wedge \mathsf{Vf}(pk, m, \sigma))$

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

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

Fig. 8: Witness extractability experiment for adaptor signatures

where the experiment $\mathsf{aWitExt}_{\mathcal{A},\Pi_{\mathsf{AS}}}$ is defined as follows

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

Definition 9 (Secure Adaptor Signature Scheme): An adaptor signature scheme Π_{AS} is secure, if it is aEUF-CMA secure, pre-signature adaptable and witness extractable.

D. Non-Interactive Zero Knowledge Proofs.

Let $R : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ be a n NP-witness-relation with corresponding NP language $\mathcal{L} := \{x : \exists w \text{ s.t. } R(x, w) = 1\}$. A non-interactive zero-knowledge proof (NIZK) [64] system for the relation R is initialized with a setup algorithm $\mathsf{Setup}(1^\lambda)$ that, outputs a common reference string *crs* and a trapdoor td. A prover can show the validity of a statement x (provided he has a witness w) by invoking $\mathcal{P}_{\mathsf{NIZK},\mathcal{L}_R}(crs, x, w)$, which outputs a proof π. The verifier checks the proof π efficiently checked using $\mathcal{V}_{\mathsf{NIZK},\mathcal{L}_R}(crs, x, \pi)$. We require a NIZK system to be (1) *zero-knowledge*, where the verifier does not learn more than the validity of the statement x, and (2) *simulation sound extractable*, if there exists an extractor algorithm $\mathcal{E}$ that on input the common reference string *crs*, a trapdoor information td, the statement x and the proof π, and outputs a witness w such that $(x, w) \in R$ with high probability. For formal UC-style definition of security we refer the reader to [78].

E. Verifiable Timed Dlog

Definition 10 (Soundness): A VTD scheme $\Pi_{\mathsf{VTD}} = (\mathsf{Commit}, \mathsf{Verify}, \mathsf{Open}, \mathsf{ForceOp})$ for a group $\mathbb{G}$ with generator G and order q is *sound* if there is a negligible function $\mathsf{negl}(\lambda)$

such that for all probabilistic polynomial time adversaries $\mathcal{A}$ and all $\lambda \in \mathbb{N}$, we have:

$$\Pr\left[b_1 = 1 \wedge b_2 = 0 : \begin{matrix}(H, C, \pi, \mathbf{T}) \leftarrow \mathcal{A}(1^\lambda) \\ x \leftarrow \mathsf{ForceOp}(C) \\ b_1 := \mathsf{Verify}(H, C, \pi) \\ b_2 := (H = G^x)\end{matrix}\right] \leq \mathsf{negl}(\lambda)$$

We say that a VTD is *simulation-sound* if it is sound even when the prover has access to simulated proofs for (possibly false) statements of his choice; i.e., the prover must not be able to compute a valid proof for a fresh false statement of his choice. In the following definition we present the definition of privacy.

Definition 11 (Timed Privacy): A VTD scheme $\Pi_{\mathsf{VTD}} = (\mathsf{Commit}, \mathsf{Verify}, \mathsf{Open}, \mathsf{ForceOp})$ for a group $\mathbb{G}$ with generator G and order q is *private* if there exists a PPT simulator $\mathcal{S}$, a negligible function $\mathsf{negl}(\lambda)$, and a polynomial $\tilde{\mathbf{T}}$ such that for all polynomials $\mathbf{T} > \tilde{\mathbf{T}}$, all PRAM algorithms $\mathcal{A}$ whose running time is at most $t < \mathbf{T}$, all messages $m \in \{0,1\}^*$, and all $\lambda \in \mathbb{N}$ it holds that

$$\left| \begin{matrix} \Pr\left[\mathcal{A}(H, C, \pi) = 1 : \begin{matrix} x \leftarrow \{0,1\}^\lambda \\ H := G^x \\ (C, \pi) \leftarrow \mathsf{Commit}(x, \mathbf{T}) \end{matrix}\right] \\ - \Pr\left[\mathcal{A}(H, C, \pi) = 1 : \begin{matrix} x \leftarrow \{0,1\}^\lambda \\ H := G^x \\ (C, \pi) \leftarrow \mathcal{S}(H, \mathbf{T}) \end{matrix}\right] \end{matrix} \right| \leq \mathsf{negl}(\lambda)$$

1) Construction of Verifiable Timed Dlog: The construction uses a NIZK for proving range proofs over time-lock puzzles. The language is specified below.

$$\mathcal{L}_{\mathsf{range}} := \begin{Bmatrix} stmt = (Z, a, b, \mathbf{T}) : \exists wit = (x, r) \text{ s.t.} \\ (Z \leftarrow \mathsf{PGen}(\mathbf{T}, x; r)) \ \wedge (x \in [a, b]) \end{Bmatrix}$$

The formal construction is given in Figure 9.

F. Security analysis of construction of Atomic Swaps from Adaptor Signatures for Schnorr and ECDSA signatures

Proof 1 (Proof of Theorem 5.1): We now prove that our protocol in Figure 5 securely US-realizes the atomic swap functionality from Figure 3.

We describe a simulator $\mathcal{S}$ that handles either of the parties P_0 or P_1 that are corrupted by a PPT $\mathcal{A}$ and simulates the real world execution protocol while interacting with the ideal functionality $\mathcal{F}_{\mathsf{swap}}$. We have a static corruption where the environment $\mathcal{E}$ at the beginning of a session specifies the corrupted parties and the honest parties. The simulator $\mathcal{S}$ faithfully impersonates the honest party. For operations exclusively among corrupted users, the environment does not expect any interaction with the simulator. Similarly, communications exclusively among honest nodes happen through secure channels and therefore the attacker does not gather any additional information other than the fact that the communication took place. For simplicity, we omit these operations in the description of our simulator. The operations to be simulated for a n-to-$\tilde{n}$ atomic swap are described in the following.

In describing $\mathcal{S}$'s operations for swapping, we begin by describing a series of hybrid executions, where we begin with a real world execution and gradually change the simulation in these hybrids and then we argue about the proximity of neighbouring experiments. Simulator $\mathcal{S}$'s execution for the payment operation is defined as the final hybrid's execution. Below we describe the hybrid executions first and later argue about their proximity. Note that the switching of hybrid executions is performed over every session, but one at a time and we only discuss here a single time for simplicity and readability.

Hybrid $\mathcal{H}_0$: This is the same as the real execution of the protocol in Figure 5.

Hybrid $\mathcal{H}_1$: This is the same as the above execution except now the 2PC protocol $\Gamma^{\mathsf{SIG}}_{\mathsf{JKGen}}$ in the freezing coins of swap setup phase to generate shared keys is simulated using the 2PC simulator $\mathcal{S}_{\mathsf{2pc},1}$ for the corrupted parties. Notice that such a simulator is guaranteed to exist for the 2PC protocol $\Gamma^{\mathsf{SIG}}_{\mathsf{JKGen}}$. Rest of the execution is unchanged from $\mathcal{H}_0$.

Hybrid $\mathcal{H}_2$: This is the same as the above execution except now the 2PC protocol Γ_{AdpSg} in the swap lock phase to generate the pre-signatures is simulated using $\mathcal{S}_{\mathsf{2pc},2}$ for the corrupted parties.

Hybrid $\mathcal{H}_3$: This is the same as the above execution except now if in some session q_3, the adversary corrupts user P_1 and the adversary outputs σ^* on a transaction $tx^{(1)}_{\mathsf{swp},i}$ under the public key $pk^{(01)}_i$ for some $i \in [n]$ such that $\Pi^{\mathsf{SIG}}_{\mathsf{DS}}.\mathsf{Vf}\left(pk^{(01)}_i, tx^{(1)}_{\mathsf{swp},i}, \sigma^*\right) = 1$, before the simulator initiates the swap on behalf of P_0, the simulator aborts by outputting $\mathtt{abort}_1$.

Hybrid $\mathcal{H}_4$: This is the same as the above execution except now if in some session q_4, the adversary corrupts P_0 and outputs σ^* on a transaction $tx^{(0)}_{\mathsf{swp},k}$ under the public key $pk^{(10)}_k$ for some $k \in [\tilde{n}]$ such that $\Pi^{\mathsf{SIG}}_{\mathsf{DS}}.\mathsf{Vf}\left(pk^{(10)}_k, tx^{(0)}_{\mathsf{swp},k}, \sigma^*\right) = 1$. The simulator computes $y' \leftarrow \Pi^{\mathsf{SIG}}_{\mathsf{AS}}.\mathsf{Ext}\left(\sigma^*, \tilde{\sigma}^{(0)}_{\mathsf{swp},k}, Y\right)$ and if $(Y, y') \notin R$, the simulator aborts by outputting $\mathtt{abort}_2$.

Hybrid $\mathcal{H}_5$: This is the same as the above execution except now if in some session q_5, the adversary corrupts P_0 and outputs σ^* on a transaction $tx^{(0)}_{\mathsf{swp},k}$ under the public key $pk^{(10)}_k$ for some $k \in [\tilde{n}]$ such that $\Pi^{\mathsf{SIG}}_{\mathsf{DS}}.\mathsf{Vf}\left(pk^{(10)}_k, tx^{(0)}_{\mathsf{swp},k}, \sigma^*\right) = 1$.

- the simulator computes $y \leftarrow \Pi^{\mathsf{SIG}}_{\mathsf{AS}}.\mathsf{Ext}\left(\sigma^*, \tilde{\sigma}^{(0)}_{\mathsf{swp},k}, Y\right)$ and $(Y, y) \in R$
- the simulator computes for all $i \in [n]$,

$$\sigma^{(1)}_{\mathsf{swp},i} \leftarrow \Pi^{\mathsf{SIG}}_{\mathsf{AS}}.\mathsf{Adapt}\left(\tilde{\sigma}^{(1)}_{\mathsf{swp},i}, y\right)$$

and there exists some $i^* \in [n]$ such that

$$\Pi^{\mathsf{SIG}}_{\mathsf{DS}}.\mathsf{Vf}\left(pk^{(01)}_{i^*}, tx^{(1)}_{\mathsf{swp},i^*}, \sigma^{(1)}_{\mathsf{swp},i^*}\right) = 0$$

Setup: On input 1^λ the setup algorithm does the following.

- Run $\mathsf{SetupZK}(1^\lambda)$ to generate crs_{range}
- Generate the public parameters $pp \leftarrow \mathsf{PSetup}(1^\lambda, \mathbf{T})$
- Output $crs := (crs_{\mathsf{range}}, pp)$

Commit and Prove: On input (crs, wit) the Com algorithm does the following.

- Parse $wit := x$, $crs := (crs_{\mathsf{range}}, pp)$, $H := G^x$
- For all $i \in [t-1]$ sample a uniform $x_i \leftarrow \mathbb{Z}_q$ and set $H_i := G^{x_i}$
- For all $i \in \{t, \dots, n\}$ compute
$$x_i = \left(x - \sum_{j \in [t]} x_j \cdot \ell_j(0) \right) \cdot \ell_i(0)^{-1}$$
$$H_i = \left(\frac{H}{\prod_{j \in [t]} H_j^{\ell_j(0)}} \right)^{\ell_i(0)^{-1}}$$
where $\ell_i(\cdot)$ is the i-th Lagrange polynomial basis.
- For $i \in [n]$, generate puzzles with corresponding range proofs as shown below
$$r_i \leftarrow \{0,1\}^\lambda, Z_i \leftarrow \mathsf{PGen}(\mathsf{p}, x_i; r_i)$$
$$\pi_{\mathsf{range},i} \leftarrow \mathcal{P}_{\mathsf{NIZK},\mathcal{L}_{\mathsf{range}}}(crs_{\mathsf{range}}, (Z_i, a, b, \mathbf{T}), (x_i, r_i))$$
- Compute $I \leftarrow H'\left(H, (H_1, Z_1, \pi_{\mathsf{range},1}), \dots, (H_n, Z_n, \pi_{\mathsf{range},n})\right)$
- The Com algorithm outputs $C := (Z_1, \dots, Z_n, \mathbf{T})$ and $\pi := (\{H_i, \pi_{\mathsf{range},i}\}_{i \in [n]}, I, \{x_i, r_i\}_{i \in I})$
- Finally output (H, C, π)

Verification: On input (crs, H, C, π) the Verify algorithm does the following.

- Parse $C := (Z_1, \dots, Z_n, \mathbf{T})$, $\pi := (\{H_i, \pi_{\mathsf{range},i}\}_{i \in [n]}, I, \{x_i, r_i\}_{i \in I})$ and $crs := (crs_{\mathsf{range}}, pp)$
- If any of the following conditions is satisfied output 0, else return 1:
  1) There exists some $j \notin I$ such that $\prod_{i \in I} H_i^{\ell_i(0)} \cdot H_j^{\ell_j(0)} \neq H$
  2) There exists some $i \in [n]$ such that $\mathcal{V}_{\mathsf{NIZK},\mathcal{L}_{\mathsf{range}}}(crs_{\mathsf{range}}, (Z_i, a, b, \mathbf{T}), \pi_{\mathsf{range},i}) \neq 1$
  3) There exists some $i \in I$ such that $Z_i \neq \mathsf{PGen}(\mathsf{p}, x_i; r_i)$ or $H_i = G^{x_i}$
  4) $I \neq H'\left(H, (H_1, Z_1, \pi_{\mathsf{range},1}), \dots, (H_n, Z_n, \pi_{\mathsf{range},n})\right)$

Open: The Open algorithm outputs $(x, \{r_i\}_{i \in [n]})$.

Force Open: The $\mathsf{ForceOp}$ algorithm take as input $C := (Z_1, \dots, Z_n, \mathbf{T})$ and works as follows:

- Runs $x_i \leftarrow \mathsf{Solve}(pp, Z_i)$ for $i \in [n]$ to obtain all shares. $\mathsf{ForceOp}$ has to solve only $(n - t + 1)$ puzzles, as $t - 1$ puzzles are already opened.
- Output $x := \sum_{j \in [t]} (x_j) \cdot \ell_j(0)$ where wlog., the first t are valid shares.

Fig. 9: Verifiable Timed Dlog for a group $\mathbb{G}$ with generator G and order q, where $H = G^x$, and x is the discrete log value committed to

, the simulator aborts by outputting $\mathtt{abort}_3$.

Hybrid $\mathcal{H}_6$: This is the same as the above execution except now if in some session q_6, an adversarial P_0, outputs any transaction tx^* and a signature σ^* such that $\Pi^{\mathsf{SIG}}_{\mathsf{DS}}.\mathsf{Vf}\left(pk_i^{(01)}, tx^*, \sigma^*\right) = 1$ for some $i \in [n]$ before time $\mathbf{T}_0$, the simulator aborts by outputting $\mathtt{abort}_{\mathrm{PRIV},0}$.

Hybrid $\mathcal{H}_7$: This is the same as the above execution except now if in some session q_7, an adversarial P_1, outputs any transaction tx^* and a signature σ^* such that $\Pi^{\mathsf{SIG}}_{\mathsf{DS}}.\mathsf{Vf}\left(pk_k^{(10)}, tx^*, \sigma^*\right) = 1$ for some $k \in [\tilde{n}]$ before time $\mathbf{T}_1$, the simulator aborts by outputting $\mathtt{abort}_{\mathrm{PRIV},1}$.

Hybrid $\mathcal{H}_8$: This is the same execution as above except now, if in some session q_8, the adversary corrupts P_0, and the simulator obtains $sk' \leftarrow \Pi_{\mathsf{VTD}}.\mathsf{ForceOp}\left(C_j^{(1)}\right)$ for some $j \in [\tilde{n}]$. The simulator aborts with $\mathtt{abort}_4$, if $sk' \neq sk_{0,j}^{(10)}$ where $sk_{0,j}^{(10)}$ is the secret share of the adversary generated in the freeze coin phase.

Hybrid $\mathcal{H}_9$: This is the same execution as above except now, if in some session q_9, the adversary corrupts P_1, and the simulator obtains $sk' \leftarrow \Pi_{\mathsf{VTD}}.\mathsf{ForceOp}\left(C_j^{(0)}\right)$ for some $j \in [n]$. The simulator aborts with $\mathtt{abort}_5$, if $sk' \neq sk_{1,j}^{(01)}$ where $sk_{1,j}^{(01)}$ is the secret share of the adversary generated in the freeze coin phase.

Simulator $\mathcal{S}$: The execution of the simulator is defined as the execution in $\mathcal{H}_9$ while it interacts with the ideal functionality $\mathcal{F}_{\mathsf{swap}}$. The simulator receives $(\mathtt{swap}_1, \mathsf{id}, V, \tilde{V}, \mathsf{PK}, \tilde{\mathsf{PK}}, U_1)$ and $(\mathtt{swap}_2, \mathsf{id}, V, \tilde{V}, \mathsf{PK}, \tilde{\mathsf{PK}}, U_0)$ from $\mathcal{F}_{\mathsf{swap}}$. It proceeds as in the execution of $\mathcal{H}_9$ by simulating the view of the adversary appropriately as it receives messages from the ideal functionality $\mathcal{F}_{\mathsf{swap}}$. If the simulated view deviates from the execution of the ideal functionality, we note that the simulation must have already aborted (as discussed in cases of abort in the above hybrids).

Below we discuss the indistinguishability arguments and we use the notation $\approx_c, \approx_{\mathbf{T}}$ to denote computational indistinguishability for a PPT algorithm, and indistinguishability for depth $\mathbf{T}$ bounded algorithms, respectively.

$\mathcal{H}_0 \approx_c \mathcal{H}_1$: the indistinguishability follow from the security of the 2PC protocols, namely $\Gamma^{\mathsf{SIG}}_{\mathsf{JKGen}}$ in the freeze coin phase. Security of the 2PC protocol $\Gamma^{\mathsf{SIG}}_{\mathsf{JKGen}}$ for the derivation of keys guarantees the existence of $\mathcal{S}_{\mathsf{2pc},1}$.

$\mathcal{H}_1 \approx_c \mathcal{H}_2$: the indistinguishability follow from the security of the 2PC protocols, namely Γ_{AdpSg} in the swap lock phase. The security of the 2PC protocol for the pre-signature generation Γ_{AdpSg} guarantees the existence of $\mathcal{S}_{\mathsf{2pc},2}$. Notice that the simulator extracts the adversaries key shares in $\Gamma^{\mathsf{SIG}}_{\mathsf{JKGen}}$ using $\mathcal{S}_{\mathsf{2pc},1}$ and uses them in the simulation of $\mathcal{S}_{\mathsf{2pc},2}$.

$\mathcal{H}_2 \approx_c \mathcal{H}_3$: the only difference between the hybrids is that in $\mathcal{H}_3$ the simulator aborts with $\mathtt{abort}_1$, if in some session q_3, the adversary corrupts user P_1 and the adversary outputs σ^* on a transaction $tx^{(1)}_{\mathsf{swp},i}$ under the public key $pk^{(01)}_i$ for some $i \in [n]$ such that $\Pi^{\mathsf{SIG}}_{\mathsf{DS}}.\mathsf{Vf}\left(pk^{(01)}_i, tx^{(1)}_{\mathsf{swp},i}, \sigma^*\right) = 1$, before the simulator initiates the swap on behalf of P_0. Via Lemma 1, we bound the probability $\Pr[\mathtt{abort}_1|\mathcal{H}_3]$ to be negligible. Therefore the indistinguishability between $\mathcal{H}_2$ and $\mathcal{H}_3$ follows.

Lemma 1: There exists a negligible function negl such that $\Pr[\mathtt{abort}_1|\mathcal{H}_3] \leq \mathsf{negl}(\lambda)$

Proof 2 (Lemma 1): To show this, we consider the following hybrid executions. Using standard hybrid arguments we show the indistinguishability of the hybrids. Using the final hybrid execution we show a reduction to the unforgeability property of the adaptor signature scheme and show that $\Pr[\mathtt{abort}_1|\mathcal{H}_3] \leq \mathsf{negl}(\lambda)$. Note that these hybrid executions are only designed to prove the above lemma.

Hybrid $\mathcal{H}_{3,0}$: Is the same execution as $\mathcal{H}_3$ as in session q_3, except now the simulator chooses the keys $pk^{(01)}_i$ for $i \in [n]$ and $pk^{(10)}_j$ for $j \in [\tilde{n}]$ uniformly at random and chooses the adversarial shares of the corresponding secret keys also to be chosen unfirmly at random. Rest of the execution is unchanged.

Hybrid $\mathcal{H}'_3$: Is the same execution as $\mathcal{H}_{3,0}$ as in session q_3, except now the hard relation is chosen as follows. The simulator samples $Y \leftarrow R$ without the corresponding witness and simulates the NIZK proof $\pi \leftarrow \mathcal{S}_{\mathsf{NIZK}}(Y)$ using the NIZK simulator $\mathcal{S}_{\mathsf{NIZK}}$ for the language $\mathcal{L}_R$. Rest of the execution is unchanged.

Hybrid $\mathcal{H}''_3$: Is the same execution as $\mathcal{H}'_3$ as in session q_3, except now the pre-signatures are hard-coded into the simulation of Γ_{AdpSg}.

We now show that the above hybrids are indistinguishable.

$\mathcal{H}_3 \equiv \mathcal{H}_{3,0}$: the hybrids are identical as the adversary only sees random shares of secret keys of uniformly chosen shared public keys.

$\mathcal{H}_{3,0} \approx_c \mathcal{H}'_3$: the indistinguishability follows immediately from the zero-knowledge property of the NIZK system $(\mathcal{P}_{\mathsf{NIZK},\mathcal{L}_R}, \mathcal{V}_{\mathsf{NIZK},\mathcal{L}_R})$, for which the simulator $\mathcal{S}_{\mathsf{NIZK}}$ is guaranteed to exist.

$\mathcal{H}'_3 \equiv \mathcal{H}''_3$: the executions are functionally equivalent. We only change the way the output of the simulated 2-PC (Γ_{AdpSg}) is generated and therefore the hybrid executions $\mathcal{H}'_3$ and $\mathcal{H}''_3$ are identical.

By a standard hybrid argument we can see that $\mathcal{H}_3 \approx_c \mathcal{H}''_3$. This means that the probability with which the abort event $\mathtt{abort}_1$ occurs in $\mathcal{H}_3$ is negligibly close to the probability of it occurring in $\mathcal{H}''_3$.

We now show that $\Pr[\mathtt{abort}_1|\mathcal{H}''_3] \leq \mathsf{negl}(\lambda)$. To do this, we show a reduction $\mathcal{R}$ that plays against the unforgeability property of the adaptor signature scheme $\Pi^{\mathsf{SIG}}_{\mathsf{AS}}$ and uses the adversary $\mathcal{A}$ that triggers $\mathtt{abort}_1$ in $\mathcal{H}''_3$ in the q_3-th session, as a sub-routine. The reduction $\mathcal{R}$ plays in the $\mathsf{aSigForge}$ experiment and gets as input a public key pk. The reduction guesses the session q_3 (from a polynomial number of sessions). It guesses an index $i^* \in [n]$ and sets $pk^{(01)}_{i^*} := pk$ and sets other keys $\left\{pk^{(01)}_i\right\}_{i\in[n]\setminus i^*}, \left\{pk^{(10)}_k\right\}_{k\in[\tilde{n}]}$ by choosing their respective secret keys itself. The reduction follows the execution as in $\mathcal{H}''_3$ in the swap lock phase until the reduction outputs $tx^{(1)}_{\mathsf{swp},i^*}$ to its challenger to learn $\left(\tilde{\sigma}^{(1)}_{\mathsf{swp},i^*}, Y\right)$. The reduction uses this Y to setup the pre-signatures $\left\{\tilde{\sigma}^{(1)}_{\mathsf{swp},i}\right\}_{i\in[n]\setminus i^*}$ and $\left\{\tilde{\sigma}^{(0)}_{\mathsf{swp},k}\right\}_{k\in[\tilde{n}]}$. The adversary outputs a signature σ^*. The reduction outputs σ^* to its own challenger.

Clearly the reduction is efficient. Notice that if $\Pi^{\mathsf{SIG}}_{\mathsf{DS}}.\mathsf{Vf}\left(pk, tx^{(1)}_{\mathsf{swp},i^*}, \sigma^*\right) = 1$, the reduction succeeds in winning $\mathsf{aSigForge}$ experiment. Therefore, if $\Pr[\mathtt{abort}_1|\mathcal{H}''_3]$ is non-negligible, the reduction outputs a valid forgery with the same non-negligible probability (except for a loss of a polynomial factor owing to the guessing of the session q_3 and the index i^*). This is a contradiction to the unforgeability property of the adaptor signature scheme $\Pi^{\mathsf{SIG}}_{\mathsf{AS}}$. This implies $\Pr[\mathtt{abort}_1|\mathcal{H}''_3] \leq \mathsf{negl}(\lambda)$ and the lemma follows immediately. □

$\mathcal{H}_3 \approx_c \mathcal{H}_4$: The only difference between the hybrids is that in $\mathcal{H}_4$, the simulator aborts with $\mathtt{abort}_2$ if in some session q_4, the adversary corrupts P_0 and does the following:

- the adversary outputs σ^* on a transaction $tx^{(0)}_{\mathsf{swp},k}$ under the public key $pk^{(10)}_k$ for some $k \in [\tilde{n}]$ such that $\Pi^{\mathsf{SIG}}_{\mathsf{DS}}.\mathsf{Vf}\left(pk^{(10)}_k, tx^{(0)}_{\mathsf{swp},k}, \sigma^*\right) = 1$

- the simulator computes $y' \leftarrow \Pi^{\mathsf{SIG}}_{\mathsf{AS}}.\mathsf{Ext}\left(\sigma^*, \tilde{\sigma}^{(0)}_{\mathsf{swp},k}, Y\right)$

and we have $(Y, y') \notin R$. Via Lemma 2, we bound the probability of the abort event happening in $\mathcal{H}_4$ to be atmost negligible in the security parameter. The indistinguishability of the hybrids follows immediately.

Lemma 2: There exists a negligible function negl such that $\Pr[\mathtt{abort}_2|\mathcal{H}_4] \leq \mathsf{negl}(\lambda)$.

Proof 3 (Lemma 2): To show this, we consider the following hybrid executions. Note that these hybrid executions are designed to prove the above lemma and do not feature in our main line of hybrids of simulation. We have the adversary corrupting party P_0.

q_4 $\mathtt{abort}_2$

Hybrid $\mathcal{H}_{4,0}$: Is the same execution as $\mathcal{H}_4$ as in session q_4, except now the simulator chooses the keys $pk^{(01)}_i$ for $i \in [n]$ and $pk^{(10)}_j$ for $j \in [\tilde{n}]$ uniformly at random and chooses the adversarial shares of the corresponding secret keys also to be chosen uniformly at random. Rest of the execution is unchanged.

Hybrid $\mathcal{H}'_4$: Is the same execution as $\mathcal{H}_{4,0}$ as in session q_4, except now instead of just simulating the 2-PC protocol of the pre-signature generation, now the corresponding pre-signatures for $n + \tilde{n}$ executions of the 2-PC protocols are hard-coded into the output of the respective simulators that are simulating for $\Gamma^{\mathsf{SIG}}_{\mathsf{AdpSg}}$.

We now argue about the indistinguishability of the hybrids.

$\mathcal{H}_4 \equiv \mathcal{H}_{4,0}$: : the hybrids are identical as the adversary only sees random shares of secret keys of uniformly chosen shared public keys.

$\mathcal{H}_{4,0} \equiv \mathcal{H}'_4$: the executions are functionally equivalent. We only change the way the output of the simulated 2-PC ($\Gamma^{\mathsf{SIG}}_{\mathsf{AdpSg}}$) is generated and therefore the hybrid executions $\mathcal{H}_{4,0}$ and $\mathcal{H}'_4$ are identical.

By a standard hybrid argument we can show that $\mathcal{H}_4 \equiv \mathcal{H}'_4$. This means that the probability with which the adversary triggers $\mathtt{abort}_2$ in some session q_4 in $\mathcal{H}_4$ must be the same probability of it happening in $\mathcal{H}'_4$.

We now show that $\Pr[\mathtt{abort}_2|\mathcal{H}'_4] \leq \mathsf{negl}(\lambda)$. To show this, we show a reduction $\mathcal{R}$ to the witness extractability property of the adaptor signature scheme $\Pi^{\mathsf{SIG}}_{\mathsf{AS}}$. The reduction plays in the $\mathsf{aWitExt}$ experiment and is given as input a public key pk. The reduction guesses the session q_4 and an index $k^* \in [\tilde{n}]$. It sets $pk^{(10)}_{k^*} := pk$ and chooses a random bit string as the adversary's share of the secret key. The reduction sets other keys as in $\mathcal{H}'_4$. When the adversary sends (Y, π), the reduction checks if $\mathcal{V}_{\mathsf{NIZK},\mathcal{L}_R}(Y, \pi) = 1$, and aborts otherwise. The reduction later sends $\left(tx^{(0)}_{\mathsf{swp},k^*}, Y\right)$ to its own challenger. It gets back a pre-signature $\tilde{\sigma}$. The reduction hard-codes $\tilde{\sigma}$ as the pre-signature $\tilde{\sigma}^{(0)}_{\mathsf{swp},k^*}$. For the rest of the execution, the reduction faithfully simulates as in $\mathcal{H}'_4$. Whenever the adversary outputs a signature σ^*, the reduction checks simply outputs σ^* to its own challenger.

Notice that if the adversary triggers $\mathtt{abort}_2$ event, then the reduction outputs a signature that violates the witness extractability property of $\Pi^{\mathsf{SIG}}_{\mathsf{AS}}$. That is, if $\Pr[\mathtt{abort}_2|\mathcal{H}'_4]$ is non-negligible, then the success probability of reduction $\mathcal{R}$ is also (except with a loss of a polynomial factor) close to the same non-negligible probability. This is a contradiction to the witness extractability. Therefore we have that $\Pr[\mathtt{abort}_2|\mathcal{H}'_4] \leq \mathsf{negl}(\lambda)$. The lemma follows immediately. □

$\mathcal{H}_4 \equiv \mathcal{H}_5$: The only difference between the hybrids is that in $\mathcal{H}_5$, the simulator aborts with $\mathtt{abort}_3$ if in some session q_5, the adversary corrupts P_0 and does the following:

- the adversary outputs σ^* on a transaction $tx^{(0)}_{\mathsf{swp},k}$ under the public key $pk^{(10)}_k$ for some $k \in [\tilde{n}]$ such that

$$\Pi^{\mathsf{SIG}}_{\mathsf{DS}}.\mathsf{Vf}\left(pk^{(10)}_k, tx^{(0)}_{\mathsf{swp},k}, \sigma^*\right) = 1$$

- the simulator computes $y \leftarrow \Pi^{\mathsf{SIG}}_{\mathsf{AS}}.\mathsf{Ext}\left(\sigma^*, \tilde{\sigma}^{(0)}_{\mathsf{swp},k}, Y\right)$ and $(Y, y) \in R$
- the simulator computes for all $i \in [n]$,

$$\sigma^{(1)}_{\mathsf{swp},i} \leftarrow \Pi^{\mathsf{SIG}}_{\mathsf{AS}}.\mathsf{Adapt}\left(\tilde{\sigma}^{(1)}_{\mathsf{swp},i}, y\right)$$

and there exists some $i^* \in [n]$ such that

$$\Pi^{\mathsf{SIG}}_{\mathsf{DS}}.\mathsf{Vf}\left(pk^{(01)}_{i^*}, tx^{(1)}_{\mathsf{swp},i^*}, \sigma^{(1)}_{\mathsf{swp},i^*}\right) = 0$$

. We show that the probability of $\mathtt{abort}_3$ event triggered in $\mathcal{H}_5$ is 0. The argument follows similar to Lemma 2, except here we do not have a reduction in the end, instead we have that probability of the abort event being triggered is 0 from the pre-signature adaptability property of $\Pi^{\mathsf{SIG}}_{\mathsf{AS}}$.

$\mathcal{H}_5 \approx_c \mathcal{H}_6$: the only difference between the hybrids is that in $\mathcal{H}_6$ the simulator aborts with $\mathtt{abort}_{\mathrm{PRIV},0}$ when in some session q_6, an adversarial P_0, outputs any transaction tx^* and a signature σ^* such that $\Pi^{\mathsf{SIG}}_{\mathsf{DS}}.\mathsf{Vf}\left(pk^{(01)}_i, tx^*, \sigma^*\right) = 1$ for some $i \in [n]$ before time $\mathbf{T}_0$. We now bound the probability of this abort event as being negligible using Lemma 3.

Lemma 3: There exists a negligible function negl such that $\Pr[\mathtt{abort}_{\mathrm{PRIV},0}|\mathcal{H}_6] \leq \mathsf{negl}(\lambda)$.

Proof 4 (Lemma 3): To show this, we consider the following hybrid executions. Note that these hybrid executions are designed to show that the abort event $\mathtt{abort}_{\mathrm{PRIV},0}$ occurs with negligible probability. These hybrids do not feature in our main line of hybrids of simulation. Here we have the case with a corrupt P_0

Hybrid $\mathcal{H}'_6$: Is the same execution as $\mathcal{H}_6$ in session q_6, except now the simulator chooses the keys $pk^{(01)}_i$ for $i \in [n]$ uniformly at random and chooses the adversarial shares of the secret keys also to be chosen uniformly at random. Rest of the execution is unchanged.

Hybrid $\mathcal{H}_{6,j}, j \in [n]$: Is the same execution as $\mathcal{H}'_6$ as in the session q_6, except now the first j verifiable timed dlog's are generated using the simulator $\mathcal{S}_{\mathsf{VTD}}$, where if $\mathsf{SIG} = \mathsf{Schnorr}$ then $\left(C^{(0)}_j, \pi^{(0)}_j\right) \leftarrow \mathcal{S}_{\mathsf{VTD}}\left(\frac{pk^{(01)}_j}{G^{sk^{(01)}_{0,j}}}, \mathbf{T}_0\right)$. If $\mathsf{SIG} = \mathsf{ECDSA}$ then $\left(C^{(0)}_j, \pi^{(0)}_j\right) \leftarrow \mathcal{S}_{\mathsf{VTD}}\left(\left(pk^{(01)}_j\right)^x, \mathbf{T}_0\right)$ where $x :=$

$\left(sk_{0,j}^{(01)}\right)^{-1}$. Note that the simulator of the execution is aware of the keys $sk_{0,j}^{(01)}$ as it extracted them from the adversary. Rest of the execution is the same as in $\mathcal{H}_6'$.

Below we argue the indistinguishability of the above hybrids.

$\underline{\mathcal{H}_6 \equiv \mathcal{H}_6'}$: the hybrids are identical as the adversary only sees random shares of secret keys of uniformly chosen shared public keys.

$\underline{\mathcal{H}_{6,j} \approx_{\mathbf{T}_0} \mathcal{H}_{6,j+1}}$: Let $\mathcal{H}_{6,0} = \mathcal{H}_6'$. The Indistinguishability of $\mathcal{H}_{6,j}$ and $\mathcal{H}_{6,j+1}$ for depth $\mathbf{T}_0$ bounded adversaries follows from the timed privacy of Π_{VTD}.

To show this, we show a reduction $\mathcal{R}$ to the timed privacy of Π_{VTD} that uses the depth $\mathbf{T}_0$ bounded distinguisher of $\mathcal{H}_{6,j}$ and $\mathcal{H}_{6,j+1}$ as a sub-routine. The reduction $\mathcal{R}$ is given as inputs (H, C, π). The reduction guesses the session q_6 and $i \in [n]$ and sets $pk_{1,i}^{(01)} := H$, which is honest party P_1's share of the public key $pk_i^{(01)}$. The reduction can do this, because it can choose the adversarial share $sk_{0,i}^{(01)}$ randomly and set $pk_i^{(01)} := H \cdot G^{sk_{0,i}^{(01)}}$ if $\mathsf{SIG} = \mathsf{Schnorr}$ and $pk_i^{(01)} := H^{sk_{0,i}^{(01)}}$ if $\mathsf{SIG} = \mathsf{ECDSA}$. With this, the reduction can call $\mathcal{S}_{\mathsf{2pc},1}$ with inputs $\left(pk_i^{(01)}, sk_{0,i}^{(01)}\right)$ and simulate in $\Gamma_{\mathsf{JKGen}}^{\mathsf{SIG}}$ for the i-th iteration. The reduction also sets $C_i^{(0)} := C$ and $\pi_i^{(0)} := \pi$. For the rest of the execution the reduction simulates exactly as in $\mathcal{H}_{6,j}$.

The reduction is efficient as it only performs polynomial time operations during the simulation. To argue the success probability, notice that if (H, C, π) was indeed generated as $(C, \pi) \leftarrow \Pi_{\mathsf{VTD}}.\mathsf{Commit}(x, \mathbf{T}_0)$ and $H := G^x$, then the reduction has perfectly simulated $\mathcal{H}_{6,j}$ to the underlying distinguisher. On the other hand, if (H, C, π) was generated as $(C, \pi) \leftarrow \mathcal{S}_{\mathsf{VTD}}(H, \mathbf{T}_0)$, then the reduction has perfectly simulated $\mathcal{H}_{6,j+1}$ to the underlying distinguisher. Therefore if the depth $\mathbf{T}_0$ bounded distinguisher is able to distinguish between the hybrids with non-negligible probability, then the reduction can break the timed privacy of Π_{VTD}.

By a standard hybrid argument we have that $\mathcal{H}_6 \approx_{\mathbf{T}_0} \mathcal{H}_{6,n}$. Notice that in $\mathcal{H}_{6,n}$ every timed dlog is generated by $\mathcal{S}_{\mathsf{VTD}}$ for the adversarial P_0 and no information about the shares of the honest party P_1 is available in the view of the adversary. Since $\mathcal{H}_6$ is indistinguishable from $\mathcal{H}_{6,n}$ for depth $\mathbf{T}_0$ bounded adversaries and when P_1 is honest, we have that in session q_6, the probability with which the adversary triggers $\mathtt{abort}_{\text{PRIV},0}$ in $\mathcal{H}_6$ must be the same in $\mathcal{H}_{6,n}$ except with negligible difference. Now, for the adversary to trigger $\mathtt{abort}_{\text{PRIV},0}$ in $\mathcal{H}_{6,n}$, it has to output σ^* on some transaction tx^* under the public key $pk_i^{(01)}$ for some $i \in [n]$.

We show that the probability with which the adversary can do so in $\mathcal{H}_{6,n}$ is at most negligible in the security parameter by reducing the occurrence of the event to the unforgeability of the signature scheme $\Pi_{\mathsf{DS}}^{\mathsf{SIG}}$. We construct a reduction $\mathcal{R}'$ that guesses the q_6-th session, and the index $i^* \in [n]$. It receives as input a public key pk and simulates the view of the adversary $\mathcal{A}$ faithfully as in $\mathcal{H}_{6,n}$ except that it sets $pk_{i^*}^{(01)} = pk$. If the adversary outputs a signature σ^* on some transaction tx^* under pk, the reduction $\mathcal{R}'$ outputs the same signatures as its forgery in EUF-CMA game. We conclude that since the signature scheme $\Pi_{\mathsf{DS}}^{\mathsf{SIG}}$ is unforgeable, we have that $\Pr[\mathtt{abort}_{\text{PRIV},0} | \mathcal{H}_{6,n}] \leq \mathsf{negl}(\lambda)$. This implies that $\Pr[\mathtt{abort}_{\text{PRIV},0} | \mathcal{H}_6] \leq \mathsf{negl}(\lambda)$ which proves the lemma. □

$\underline{\mathcal{H}_6 \approx_c \mathcal{H}_7}$: the only difference between the hybrids is that in $\mathcal{H}_7$ the simulator aborts with $\mathtt{abort}_{\text{PRIV},1}$ when in some session q_7, an adversarial P_1, outputs any transaction tx^* and a signature σ^* such that $\Pi_{\mathsf{DS}}^{\mathsf{SIG}}.\mathsf{Vf}\left(pk_k^{(10)}, tx^*, \sigma^*\right) = 1$ for some $k \in [\tilde{n}]$ before time $\mathbf{T}_1$. We can bound the probability of this abort event as being negligible with a analogous argument as in Lemma 3.

$\underline{\mathcal{H}_7 \approx_c \mathcal{H}_8 \approx_c \mathcal{H}_9}$: The indistinguishability of the hybrids follows from the soundness of Π_{VTD}. This concludes the proof. □

G. Extensions

In this section we outline how to modify our main protocol to support multi-party (cyclic) atomic swaps.

Atomic Swaps for any Cycle Before explaining how to extend our protocol (Figure 5) to the multi-party (> 2) settings, let us establish some notation. We denote our n-to-$\tilde{n}$ atomic swap as

$$P_0 \xrightarrow{n} P_1 \xrightarrow{\tilde{n}} P_0.$$

Consider the settings where there are three parties P_0, P_1 and P_2. Glossing over the exchange rates, consider a case where P_0 has 1 BTC and wants to swap that with 1 ETH. P_2 on the other hand has 1 ETH but only wants to swap that with 1 XRP. These two users cannot swap directly as P_0 does not have the required XRP to offer to P_2. However, if P_1 has 1 ETH and 1 XRP and is willing to facilitate the swap between P_0 and P_2, we can hope to extend our protocol to have the following sequence of swaps

$$P_0 \xrightarrow{\text{1 BTC}} P_1 \xrightarrow{\text{1 XRP}} P_2 \xrightarrow{\text{1 ETH}} P_1 \xrightarrow{\text{1 ETH}} P_0$$

Observe that now P_0 obtains the desired 1 ETH from P_1 while P_2 swaps its 1 ETH for 1 XRP with P_1 and finally the swap concludes for P_0 as he pays 1 BTC to P_1. Most generally, we would like to extend our protocol to support swaps of the form

$$P_0 \xrightarrow{n_0} P_1 \xrightarrow{n_1} \cdots \xrightarrow{n_k} P_{k+1} \xrightarrow{\tilde{n}_k} P_k \xrightarrow{\tilde{n}_{k-1}} \cdots \xrightarrow{\tilde{n}_0} P_0$$

where $n_0, n_1, \ldots, n_k$ and $\tilde{n}_k, \tilde{n}_{k-1}, \ldots, \tilde{n}_0$ are the numbers coins of different currencies that are to be swapped.

One could extend our protocol by running sequentially the lock phase (from left to right) among each pair of parties, using the same hard instance Y in the execution of $\Gamma_{\mathsf{AdpSg}}^{\mathsf{SIG}}$, which is sampled by P_0, knowing the corresponding witness y. To avoid coin theft, it is important that the lock phase between parties (P_i, P_{i+1}) starts only after the lock phase between (P_{i-1}, P_i) is completed. While syntactically correct, this attempt suffers from wormhole attacks [46], where some malicious nodes can "skip" intermediate honest nodes and steal their fees. Fortunately, [46] also describes a countermeasure to prevent such attacks. Their solution is to re-randomize the instance Y for each instance of the lock phase, by computing $\bar{Y} = Y \cdot G^{\bar{y}}$ where $\bar{y}$ is a session-specific randomizing factor, which is known by the left party P_{i-1}. This way, P_{i-1} can still recover the correct witness $(y + \bar{y})$, while $\bar{Y}$ looks uniformly distributed to all other participants. Naturally, this extension requires us to tune the timing parameters of each freezing phase, setting a cascade of times $\mathbf{T}_0, \mathbf{T}_1 + \Delta, \mathbf{T}_1 + 2\Delta, \ldots,$ analogously to [46]. However, the crucial difference is that in our approach the timing parameters only affect the hardness of the VTD and are *never* posted on-chain (i.e., no need for custom scripting language on the blockchain).

Batching VTD commitments For a single swap, party P_0 has to open n VTD commitments, while party P_1 has to open $\tilde{n}$ VTD commitments. Since opening of the commitments involves computational work, this could prove prohibitive for the parties if they have moderate computing devices. We can solve this issue, for instance, by letting the party P_0 pack n commitments into a single commitment, such that opening this one commitment lets the party learn all the values in the individual commitments. This means party P_0 only needs to compute on opening a single commitment, rather than n commitments. Opening the single commitment, P_0 learns n discrete log values that it would have learnt if it had solved the n VTD commitments individually. Such a packing strategy is possible via homomorphic operations (provided the commitments have a large message space) in the (VTD) scheme [34].

n-to-$\tilde{n}$ Swaps for Monero Our techniques from Figure 5 can also be extended to the transaction scheme of Monero, thus giving the first n-to-$\tilde{n}$ swap protocol for Monero that is efficient, does not require any hard fork, and enables coin-swaps with other currencies supporting Schnorr/ECDSA signature verification. Monero uses a linkable ring signature based transaction scheme, where the sender of a transaction generates a ring signature on the transaction with a linkability tag. The sender's key is a member of the ring of public keys or anonymity set, and the ring signature ensures that the sender's key remains hidden in this anonymity set. The linkability tag helps prevent double spending of the same key, as the tags are unique for a key and are public once the transaction is signed and posted on the blockchain.

Previous works [35], [36] propose atomic swap protocol for a 1-to-1 swap for Monero (only). Their techniques are reminiscent of adaptor signatures (Schnorr or ECDSA) for swap protocol but instead they have pre-signature generation and adaption tailored for the linkable ring signature scheme used in Monero. Specifically, P_0 and P_1 run an efficient 2-party protocol to generate a pre-signature (of the linkable ring signature) $\tilde{\sigma}^{(1)}_{\mathsf{swp}}$ on the transaction $tx^{(1)}_{\mathsf{swp}}$ with respect to the hard instance Y. Party P_1 and P_0 then run another instance of the 2-party protocol to generate another pre-signature (of the linkable ring signature) $\tilde{\sigma}^{(0)}_{\mathsf{swp}}$ on the transaction $tx^{(0)}_{\mathsf{swp}}$ with respect to the same hard instance Y. The swap complete phase proceeds in the natural way, where P_0 adapt the pre-signature $\tilde{\sigma}^{(0)}_{\mathsf{swp}}$ into a valid linkable ring signature $\sigma^{(0)}_{\mathsf{swp}}$ on $tx^{(0)}_{\mathsf{swp}}$, which lets P_1 adapt $\tilde{\sigma}^{(1)}_{\mathsf{swp}}$ into a valid linkable ring signature $\tilde{\sigma}^{(0)}_{\mathsf{swp}}$ on $tx^{(1)}_{\mathsf{swp}}$.

We can extend their protocols to support n-to-$\tilde{n}$ swaps, by similarly running the swap lock phase as in Figure 5, except instead of Schnorr or ECDSA use the adaptor linkable ring signature for Monero introduced in [35], [36]. Since public-secret key pairs in Monero consist of group elements with their discrete logarithm (H, h), we can use the same VTD commitment scheme as in Figure 5 to ensure that the parties can get their coins refunded after a timeout.

H. Generic Construction of Atomic Swaps

We present here our generic atomic swap protocol for a n-to-$\tilde{n}$ swap of coins between two users P_0 and P_1. Fundamental building blocks of our protocol are multi-lock signatures, commitment schemes, 2 party protocols for different functionalities and verifiable timed signatures. We briefly look at some of these primitives.

1) Multi-lock Signatures: We introduce a new cryptographic notion of multi-lock signatures. On a high level, this primitive lets users to create a lock ℓk on the signatures on a set of messages $\{m_1, \ldots, m_n\}$ under the public keys $\{pk_1, \ldots, pk_n\}$, respectively, with respect to another signature $\tilde{\sigma}$ (referred to as the locking signature) on a message $\tilde{m}$ under a public key $\tilde{pk}$. This means that given the signature $\tilde{\sigma}$ and the lock ℓk, one can release the signatures $\{\sigma_1, \ldots, \sigma_n\}$ where each σ_i is a valid signature on the message m_i under the public key pk_i. The security property of interest here is that given just the lock ℓk, it reveals no information about the n signatures that are locked or the locking signature. Below we formally define the interfaces, notion of correctness and the properties of interest for a multi-lock signature.

Definition 12 (Multi-lock Signatures): A multi-lock signature scheme MLS is defined with respect to a signature scheme Π_{DS} and consists of two PPT algorithms $(\mathsf{mLock}, \mathsf{mRel})$ as defined below.

$\underline{(\ell k) \leftarrow \mathsf{mLock}(n, S, \tilde{\sigma})}$: The lock algorithm takes as input the following:

- n: the number of signatures to be locked.
- $S := \{\sigma_j\}_{j \in [n]}$: a set of n signatures
- $\tilde{\sigma}$: a signature

It outputs a lock ℓk.

$\underline{S/\bot \leftarrow \mathsf{mRel}(\ell k, M, \tilde{m}, \mathsf{PK}, \tilde{pk}, \tilde{\sigma})}$: The release algorithm takes as input a a lock ℓk, a message set M of n messages, a message $\tilde{m}$, a public key set $\mathsf{PK}\{pk_j\}_{j \in [n]}$, another public key $\tilde{pk}$ and a signature $\tilde{\sigma}$. It either outputs a signature set $S := \{\sigma_i\}_{i \in [n]}$ or a special symbol $\bot$.

Correctness of mulit-lock signatures is as follows.

Definition 13 (Correctness): A multi-lock signature scheme MLS is *correct* with respect to the signature scheme Π_{DS} if for,

- all $\lambda, n \in \mathbb{N}$,
- all messages $\tilde{m} \in \{0,1\}^\lambda$ and all sets of messages $M := \{m_1, \ldots, m_{\tilde{n}}\}$ where each message is from $\{0,1\}^\lambda$,
- for all key pairs $(\tilde{pk}, \tilde{sk})$ in the support of $\Pi_{\mathsf{DS}}.\mathsf{KGen}$ and all sets of keys $\mathsf{PK} := \{pk_1, \ldots, pk_n\}$, $\mathsf{SK} := \{sk_1, \ldots, sk_n\}$ where the key pairs (pk_i, sk_i) for $i \in [\tilde{n}]$ are in the support of $\Pi_{\mathsf{DS}}.\mathsf{KGen}$,
- for all $S := \{\sigma_i\}_{i \in [n]}$ such that $\sigma_i \leftarrow \mathsf{Sign}(sk_i, m_i)$
- for all $\tilde{\sigma} \leftarrow \mathsf{Sign}(\tilde{sk}, \tilde{m})$
- for all $(\ell k) \leftarrow \mathsf{mLock}(n, S, \tilde{\sigma})$
- and for all $S \leftarrow \mathsf{mRel}(\ell k, M, \tilde{m}, \mathsf{PK}, \tilde{pk}, \tilde{\sigma})$, where $S := \{\sigma_i\}_{i \in [n]}$,

then we have that, $\Pi_{\mathsf{DS}}.\mathsf{Vf}(pk_i, m_i, \sigma_i) = 1$ for all $i \in [n]$.

Below we define the security notions for multi-lock signatures. Note that we consider without loss of generality only signatures where the signing algorithm is deterministic.

In this work we only consider signature schemes with a *deterministic* signing algorithm. This is however without loss of generality, as shown by the following (well known) lemma.

Lemma 4: Let $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ be a signature scheme with probabilistic Sign algorithm. Then there exists a signature scheme $(\mathsf{KGen}, \mathsf{Sign}^*, \mathsf{Vf})$ where Sign^* is deterministic.

Proof 5 (Sketch): The new signing algorithm Sign^* is defined to be $\mathsf{Sign}(sk, m; \mathsf{PRF}(sk, m))$, where PRF is a cryptographic pseudorandom function.

Unlockability. This property ensures that a correctly generated lock can be released to reveal a set of valid locked signatures, when provided with a valid locking signature. This intuition is formally captured in Figure 10. In the experiment, the adversary gets to choose $(M, \tilde{m})$ where M is a set of n messages. It also has access to a signing oracle for the key $\tilde{sk}$. The experiment generates a lock ℓk honestly by running $\mathsf{mLock}(n, S, \tilde{\sigma})$ and gives the lock to the adversary. The adversary returns a signature $\tilde{\sigma}^*$ which is used to release the lock ℓk and obtain a candidate set of locked signatures S'. The adversary wins the experiment if there exists some signature $\sigma_i \in S'$ that is an invalid signature on the message m_i under pk_i (condition b_0) while $\tilde{\sigma}^*$ is a valid signature on the message $\tilde{m}$ under $\tilde{pk}$ (condition b_1). A multi-lock signature scheme is said to satisfy unlockability if the adversary wins the above experiment at most with negligible probability in the security parameter.

Definition 14 (Unlockability): A multi-lock signature MLS is *unlockable* if there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, all $n \in p(\lambda)$ where $p(\cdot)$ is any polynomial and for all PPT adversaries $\mathcal{A}$, it holds that

$$\Pr[\mathsf{EUn\text{-}CMA}_{\mathcal{A},\mathsf{MLS}}(\lambda, n) = 1] \leq \mathsf{negl}(\lambda)$$

where the experiment $\mathsf{EUn\text{-}CMA}$ is defined in Figure 10.

```
$\mathsf{EUn\text{-}CMA}_{\mathcal{A},\mathsf{MLS}}(\lambda, n)$
$Q := \emptyset$
$(\tilde{pk}, \tilde{sk}) \leftarrow \Pi_{\mathsf{DS}}.\mathsf{KGen}(1^\lambda)$
for $i \in [n]$ do
    $(pk_i, sk_i) \leftarrow \Pi_{\mathsf{DS}}.\mathsf{KGen}(1^\lambda)$
$(M, \tilde{m}, \mathsf{st}) \leftarrow \mathcal{A}^{\mathsf{Sign}\mathcal{O}}(\mathsf{SK}, \tilde{pk})$,
    where $\mathsf{SK} := \{sk_i\}_{i \in [n]}$
parse $M := \{m_1, \ldots, m_n\}$
for $i \in [n]$ do
    $\sigma_i \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Sign}(sk_i, m_i)$
$S := \{\sigma_1, \ldots, \sigma_n\}$
$\tilde{\sigma} \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Sign}(\tilde{sk}, \tilde{m})$
$\ell k \leftarrow \mathsf{mLock}(n, S, \tilde{\sigma})$
$\tilde{\sigma}^* \leftarrow \mathcal{A}^{\mathsf{Sign}\mathcal{O}(\tilde{sk}, \cdot)}(\mathsf{st}, \ell k)$
$S' \leftarrow \mathsf{mRel}(\ell k, M, \tilde{m}, \mathsf{PK}, \tilde{pk}, \tilde{\sigma}^*)$,
    where $\mathsf{PK} := \{pk_i\}_{i \in [n]}$
parse $S' := \{\sigma_1, \ldots, \sigma_n\}$
$b_0 := \bigwedge_{i \in [n]} (\Pi_{\mathsf{DS}}.\mathsf{Vf}(pk_i, m_i, \sigma_i) = 0)$
$b_1 := \left(\Pi_{\mathsf{DS}}.\mathsf{Vf}(\tilde{pk}, \tilde{m}, \tilde{\sigma}^*) = 1\right)$
return $b_0 \wedge b_1$

$\mathsf{Sign}\mathcal{O}(\tilde{sk}, m)$
$\sigma \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Sign}(\tilde{sk}, m)$
return $\sigma$
```

Fig. 10: Experiment for unlockability of multi-lock signatures

Hiding. This property ensures that a correctly generated lock reveals no information about the locked or the locking signatures. The above intuition is captured in Figure 11. In the experiment, the adversary gets to choose $(M, \tilde{m})$ where M is a set of n messages, while having access to signing oracles with respect to keys $sk_1, \ldots, sk_n$ and $\tilde{sk}$. The experiment chooses a bit b uniformly random and if $b = 0$, it generates a lock correctly using $\mathsf{mLock}(n, \mathsf{SK}, \tilde{sk}, M, \tilde{m})$. If $b = 1$, the experiment uses a simulator Sim that only takes as input n, the public key set PK (corresponding to SK), the public key $\tilde{pk}$ (corresponding to $\tilde{sk}$) and the messages $M, \tilde{m}$ and outputs a lock ℓk. The adversary is given ℓk and outputs a guess b^*. The adversary wins the experiment if the guess was correct, i.e. $b = b^*$ (condition b_0) and if $\tilde{m}$ was never queried before to the signing oracle with respect to $\tilde{sk}$. The latter condition is necessary to avoid trivial attacks where the adversary uses a signature on $\tilde{m}$ to run the release algorithm. A multi-lock signature is said to satisfy hiding if the adversary wins the above experiment with probability negligibly close to $1/2$.

Definition 15 (Hiding): A multi-lock signature MLS is *hiding* if there exists a negligible function negl and a simulator algorithm Sim, such that for all $\lambda \in \mathbb{N}$, all $n \in p(\lambda)$ where $p(\cdot)$ is any polynomial and for all PPT adversaries $\mathcal{A}$ it holds that

$$\Pr[\mathsf{EHi\text{-}CMA}_{\mathcal{A},\mathsf{LS}}(\lambda, n) = 1] \leq 1/2 + \mathsf{negl}(\lambda)$$

where the experiment $\mathsf{EHi\text{-}CMA}$ is defined in Figure 11.

In the following we describe our generic construction of a multi-lock signature.

Construction in the ROM. Let $\Pi_{\mathsf{DS}} := (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ be a digital signature scheme (with deterministic signing algorithm) and let $H_i : \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda$ for $i \in [n]$ be hash functions. Our generic construction of multi-lock signature is specified in Figure 12. In our construction, a lock consists of a set of values ℓ_i for $i \in [n]$, where each ℓ_i is an xor of a locked

```
$\mathsf{EHi\text{-}CMA}_{\mathcal{A},\mathsf{MLS}}(\lambda, n)$
$\tilde{Q} := \emptyset$
$(\tilde{pk}, \tilde{sk}) \leftarrow \Pi_{\mathsf{DS}}.\mathsf{KGen}(1^\lambda)$
for $i \in [\tilde{n}]$ do
    $(pk_i, sk_i) \leftarrow \Pi_{\mathsf{DS}}.\mathsf{KGen}(1^\lambda)$
$\mathbb{O} := \{\{\mathsf{Sign}\mathcal{O}^i\}_{i\in[n]}, \mathsf{Sign}\mathcal{O}\}$
$(M, \tilde{m}, \mathsf{st}) \leftarrow \mathcal{A}^{\mathbb{O}}(\tilde{pk}, \mathsf{PK})$
        where $\mathsf{PK} := \{pk_i\}_{i\in[n]}$
parse $M := \{m_1, \ldots, m_n\}$
$b \leftarrow \{0, 1\}$
if $b = 0$
    for $i \in [n]$ do
        $\sigma_i \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Sign}(sk_i, m_i)$
    $S := \{\sigma_1, \ldots, \sigma_n\}$
    $\tilde{\sigma} \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Sign}(\tilde{sk}, \tilde{m})$
    $\ell k \leftarrow \mathsf{mLock}(n, S, \tilde{\sigma})$
else
    $\ell k \leftarrow \mathsf{Sim}(n, \mathsf{PK}, \tilde{pk}, M, \tilde{m})$
$b^* \leftarrow \mathcal{A}^{\mathbb{O}}(\mathsf{st}, \ell k)$
$b_0 := (b = b^*)$
$b_1 := (\tilde{m} \notin \tilde{Q})$
return $b_0 \wedge b_1$
```

```
$\mathsf{Sign}\mathcal{O}^i(sk_i, m)$
$\sigma \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Sign}(sk_i, m)$
return $\sigma$

$\mathsf{Sign}\mathcal{O}(\tilde{sk}, \tilde{m})$
$\tilde{\sigma} \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Sign}(\tilde{sk}, \tilde{m})$
$\tilde{Q} := \tilde{Q} \cup \{\tilde{m}\}$
return $\tilde{\sigma}$
```

Fig. 11: Experiment for hiding of multi-lock signatures

signature σ_i and the hash H_i of the locking signature $\tilde{\sigma}$. Provided the hash of the locking signature $H_i(\tilde{\sigma})$ is a random string with high entropy, we can see that intuitively the locked signature σ_i is hidden inside ℓ_i, similar to a one-time pad. The release procedure is an xor of $H_i(\tilde{\sigma})$ and the values ℓ_i in the lock for $i \in [n]$, similar to the decryption of a one-time pad.

```
$\mathsf{mLock}(n, S, \tilde{\sigma})$
parse $S := \{\sigma_1, \ldots, \sigma_n\}$
parse $M := \{m_1, \ldots, m_n\}$
for $i \in [n]$ do
    $\ell_i := \sigma_i \oplus H(\tilde{\sigma}, i)$
$\ell k := (\ell_1, \ldots, \ell_n)$
return $\ell k$
```

```
$\mathsf{mRel}(\ell k, M, \tilde{m}, \mathsf{PK}, \tilde{pk}, \tilde{\sigma})$
parse $\ell k := (\ell_1, \ldots, \ell_n)$
for $i \in [n]$ do
    $\sigma_i \leftarrow \ell_i \oplus H_i(\tilde{\sigma})$
    if $\sigma_i = \bot$ then
        return $\bot$
$S := \{\sigma_1, \ldots, \sigma_n\}$
return $S$
```

Fig. 12: Generic construction of Multi-lock Signatures

In the theorem below, we prove that our generic construction satisfies unlockability (Definition 14) and hiding (Definition 15), as defined before. In favor of a simpler analysis, we model each of the hash functions H_i as random oracles, that can be instantiated as $H(i||\cdot)$ where H is a random oracle.

Theorem A.1 (Unlockability & Hiding): Let $\Pi_{\mathsf{DS}} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ be a strongly unforgeable digital signature scheme. Then $\Pi_{\mathsf{MLS}} := (\mathsf{mLock}, \mathsf{mRel})$ is unlockable and hiding in the random oracle model.

Proof 6: Unlockability. Consider and adversary $\mathcal{A}$ that violates the unlockability property of Π_{MLS}. We construct a reduction $\mathcal{R}$ against the strong unforgeability of Π_{DS} that makes use of $\mathcal{A}$ as a sub-routine. The reduction obtains a public key $\tilde{pk}$ as its input. The reduction $\mathcal{R}$ chooses keys $(pk_i, sk_i) \leftarrow \Pi_{\mathsf{DS}}.\mathsf{KGen}(1^\lambda)$ for $i \in [n]$ and gives $(\{sk_1, \ldots, sk_n\}, \tilde{pk})$ as inputs to $\mathcal{A}$. Any signing oracle query from $\mathcal{A}$ with respect to $\tilde{pk}$ is answered by the reduction $\mathcal{R}$ using its own signing oracle in the unforgeability experiment. Random oracle queries by $\mathcal{A}$ are answered by $\mathcal{R}$ via lazy sampling by picking a random value in the range of H and returning it to $\mathcal{A}$. Whenever $\mathcal{A}$ outputs challenge messages $(\{m_1, \ldots, m_n\}, \tilde{m})$ the reduction generates $\sigma_i \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Sign}(sk_i, m_i)$ and obtains $\tilde{\sigma}$ by querying its signing oracle with $\tilde{m}$. It generates the lock ℓk as in EUn-CMA by first setting $H(\tilde{\sigma})$ and returns the lock to the adversary $\mathcal{A}$. The adversary outputs $\tilde{\sigma}^*$ and the reduction simply outputs $(\tilde{m}, \tilde{\sigma}^*)$ as its forgery. Now observe that the only way $\mathcal{A}$ can win EUn-CMA experiment is if $\tilde{\sigma}^*$ is a valid signature on $\tilde{m}$ such that $H(\tilde{\sigma}^*) \neq H(\tilde{\sigma})$. Since the hash function H is deterministic, this implies that $\tilde{\sigma}^* \neq \Pi_{\mathsf{DS}}.\mathsf{Sign}(\tilde{sk}, \tilde{m})$. Therefore the reduction $\mathcal{R}$ has output a valid forgery $\tilde{\sigma}^*$, which is a contradiction to the strong unforgeability of Π_{DS}.

Hiding. The simulator Sim simply returns a set of binary strings of the appropriate length sampled uniformly at random. Observe that to win the experiment, $\mathcal{A}$ does not query $\tilde{m}$ to the signing oracle $\Pi_{\mathsf{DS}}.\mathsf{Sign}(\tilde{sk}, \cdot)$. We now have two cases where, either the adversary guesses the signature $\Pi_{\mathsf{DS}}.\mathsf{Sign}(\tilde{sk}, \tilde{m})$ and queries it to the random oracles H_i, or it does not query the random oracles on $\Pi_{\mathsf{DS}}.\mathsf{Sign}(\tilde{sk}, \tilde{m})$. In the former case, this is clearly a break against the strong unforgeability of the signature scheme Π_{DS} and therefore this case occurs only with negligible probability. In the latter case, the value $H_i(\Pi_{\mathsf{DS}}.\mathsf{Sign}(\tilde{sk}, \tilde{m}))$ for every $i \in [n]$ is uniformly distributed (in the random oracle model) as the adversary does not query the oracles on this point. It follows that in this case the simulated and the real experiments are identical to the eyes of the adversary and therefore the probability that $b^* = b$ is exactly $1/2$. Therefore the overall success probability of the adversary breaking hiding is $1/2 + \mathsf{negl}(\lambda)$.

Construction in the Standard Model. In the following we sketch a construction of multi-lock signatures in the standard model. Let H be a sufficiently stretching leakage resilient pseudorandom generator [42], i.e. a PRG whose output is pseudorandom even if the seed is computationally unpredictable (instead of uniformly sampled). Then a multi-lock signature can be computed as

$$\sigma_1 \| \ldots \| \sigma_n \oplus H(\tilde{\sigma}).$$

It is not hard to see that the signatures $(\sigma_1, \ldots, \sigma_n)$ can be recovered deterministically given $\tilde{\sigma}$. By the unforgeability of our signature scheme, $\tilde{\sigma}$ is hard to predict, which implies that $H(\tilde{\sigma})$ is computationally indistinguishable from uniform. Therefore, the multi-lock signature is unforgeable. Leakage resilient PRGs (with arbitrary polynomial stretch) can be constructed from extremely lossy functions [42], which in turn can be built from the subexponential hardness of the DDH problem.

2) Commitment Schemes: A commitment scheme is a digital analogue of sealing a message inside an envelope. Formally, it consists of the following tuple of efficient algorithms: A commitment generation algorithm $\mathsf{Commit}(1^\lambda, m)$ that takes as input a security parameter and a message m to commit to, and outputs a commitment c and a corresponding opening information d. The opening algorithm $\mathsf{Open}(c, d)$ takes as input a commitment c and a opening information d and outputs the committed message m or outputs a special symbol $\perp$ if d is not the valid opening information for the commitment c. In addition to the standard binding and hiding properties (who's UC formalization can be found in [50]), we require that the commitment scheme has unique openings. I.e. for all commitments there exists a single valid message that causes the Open algorithm to accept.

3) Verifiable Timed Signatures: Verifiable timed signatures was proposed by Boneh and Naor [32] and subsequently studied and improved by [33], [34]. Here, a committer creates a timed commitment C of a signature σ on a message m under a public key pk, such that the commitment C can be opened in time $\mathbf{T}$. The committer also generates a proof π to prove that the commitment is well-formed. The verifier can verify if the commitment is well-formed and proceeds to force open the commitment and learns the embedded signature σ in time $\mathbf{T}$. We recall the formal definition below.

Definition 16 (Verifiable Timed Signatures): A VTS for a signature scheme $\Pi_{\mathsf{DS}} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ is a tuple of four algorithms $(\mathsf{Commit}, \mathsf{Verify}, \mathsf{Open}, \mathsf{ForceOp})$ where:

$\underline{(C, \pi) \leftarrow \mathsf{Commit}(\sigma, \mathbf{T})}$: the commit algorithm (randomized) takes as input a signature σ (generated using $\Pi.\mathsf{Sign}(sk, m)$) and a hiding time $\mathbf{T}$ and outputs a commitment C and a proof π.

$\underline{0/1 \leftarrow \mathsf{Verify}(pk, m, C, \pi)}$: the verify algorithm takes as input a public key pk, a message m, a commitment C of hardness $\mathbf{T}$ and a proof π and accepts the proof by outputting 1 if and only if, the value σ embedded in c is a valid signature on the message m with respect to the public key pk (i.e., $\Pi_{\mathsf{DS}}.\mathsf{Vf}(pk, m, \sigma) = 1$). Otherwise it outputs 0.

$\underline{(\sigma, r) \leftarrow \mathsf{Open}(C)}$: the open phase where the committer takes as input a commitment C and outputs the committed signature σ and the randomness r used in generating C.

$\underline{\sigma \leftarrow \mathsf{ForceOp}(C)}$: the force open algorithm takes as input the commitment C and outputs a signature σ.

The security requirements for a VTS are that (*soundness*) the user is convinced that, given C, the $\mathsf{ForceOp}$ algorithm will produce the committed signature σ in time $\mathbf{T}$ and that (*privacy*) all PRAM algorithms whose running time is at most t (where $t < \mathbf{T}$) succeed in extracting σ from the commitment C and π with at most negligible probability. Below we give the formal security definitions.

Definition 17 (Soundness): A VTS scheme $\Pi_{\mathsf{VTS}} = (\mathsf{Commit}, \mathsf{Verify}, \mathsf{Open}, \mathsf{ForceOp})$ for a signature scheme $\Pi_{\mathsf{DS}} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ is *sound* if there is a negligible function negl such that for all probabilistic polynomial time adversaries $\mathcal{A}$ and all $\lambda \in \mathbb{N}$, we have:

$$\Pr\left[b_1 = 1 \wedge b_2 = 0 : \begin{array}{c} (pk, m, C, \pi, \mathbf{T}) \leftarrow \mathcal{A}(1^\lambda) \\ (\sigma, r) \leftarrow \mathsf{ForceOp}(C) \\ b_1 := \mathsf{Verify}(pk, m, C, \pi) \\ b_2 := \mathsf{Vf}(pk, m, \sigma) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

We say that a VTS is *simulation-sound* if it is sound even when the prover has access to simulated proofs for (possibly false) statements of his choice; i.e., the prover must not be able to compute a valid proof for a fresh false statement of his choice. In the following definition we present the definition of privacy.

Definition 18 (Timed Privacy): A VTS scheme $\Pi_{\mathsf{VTS}} = (\mathsf{Commit}, \mathsf{Verify}, \mathsf{Open}, \mathsf{ForceOp})$ for a signature scheme $\Pi_{\mathsf{DS}} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ is *private* if there exists a PPT simulator $\mathcal{S}$, a negligible function $\mathsf{negl}(\lambda)$, and a polynomial $\tilde{\mathbf{T}}$ such that for all polynomials $\mathbf{T} > \tilde{\mathbf{T}}$, all PRAM algorithms $\mathcal{A}$ whose running time is at most $t < \mathbf{T}$, all messages $m \in \{0,1\}^*$, and all $\lambda \in \mathbb{N}$ it holds that

$$\left| \Pr\left[\mathcal{A}(pk, m, C, \pi) = 1 : \begin{array}{c} (pk, sk) \leftarrow \mathsf{KGen}(1^\lambda) \\ m \leftarrow \mathcal{A}(pk) \\ \sigma \leftarrow \mathsf{Sign}(sk, m) \\ (C, \pi) \leftarrow \mathsf{Commit}(\sigma, \mathbf{T}) \end{array}\right] - \Pr\left[\mathcal{A}(pk, m, C, \pi) = 1 : \begin{array}{c} (pk, sk) \leftarrow \mathsf{KGen}(1^\lambda) \\ m \leftarrow \mathcal{A}(pk) \\ (C, \pi) \leftarrow \mathcal{S}(pk, \mathbf{T}) \end{array}\right] \right| \leq \mathsf{negl}(\lambda)$$

4) 2-Party Protocols: We require 2 party computation protocols for various functionalities as detailed below.

Joint Key Generation. We require a 2 party protocol Γ_1 for creating a shared public key tpk whose secret key is shared among parties U_0 and U_1. Specifically, the protocol computes a commitment c of the secret key tsk (of the public key tpk) and the opening information d of this commitment. Finally, the protocol gives as output to U_0 a share d_0 of the opening information d and a share tsk_0 of the secret key tsk. Analogously, U_1 receives d_1 and tsk_1, such that $\mathsf{tsk} = \mathsf{tsk}_0 \oplus \mathsf{tsk}_1$ and $d = d_0 \oplus d_1$.

Joint Signature Generation. We require another interactive 2 party protocol Γ_2 for jointly generating a signature σ on a message m under a shared public key tpk. Each party inputs their share of the secret key and the output of Γ_2 is such that party U_0 receives nothing while U_1 receives the signature σ.

Joint Lock Generation. Finally, we require the 2 party protocol Γ_3 for jointly computing mLock of Π_{MLS} that is run among party P_0 and P_1. The protocol Γ_3 takes as input n secret key shares and opening information shares from each party, commitments to the n secret keys, and n shared public keys corresponding to the n locked signatures and secret key shares of a shared public key corresponding to the locking signature. The output of Γ_3 is ordered in its output, giving the lock ℓk first to P_0 before giving the locking signature $\tilde{\sigma}$ to P_1.

Looking ahead, this ordering is crucial since we want balance security in our atomic swap protocol. If instead, $\tilde{\sigma}$ was output to P_1 first, the party could use it to initiate the swap and abort the computation in Γ_3. This results in party P_0 not learning the lock ℓk and therefore not able to call mRel and learn the signatures $\{\sigma_1, \ldots, \sigma_n\}$, which means P_0 cannot complete the swap. We can instantiate these 2 party protocols with a general purpose 2 party computation protocols [31].

5) High level overview: The protocol as described in Figure 15, begins with party wanting to swap $v_i^{(0)}$ for $i \in [n]$ coins for $v_k^{(1)}$ for $k \in [\tilde{n}]$ coins of P_1. Just as in the adaptor signature based construction Figure 5, the protocol proceeds in three phases.

Swap Setup Phase. The coins to swap are frozen in joint keys during this phase. Specifically, the parties call a Freeze interface in Figure 13, where P_0 sends the i-th coin $v_i^{(0)}$ from its keys $pk_i^{(0)}$ to a joint address $pk_i^{(01)}$ for $i \in [n]$. Analogously party P_1 does the same by sending the k-th coin $v_k^{(1)}$ from its key $pk_k^{(1)}$ to a joint keys $pk_k^{(10)}$ for $k \in [\tilde{n}]$. At the end of this phase, for every $i \in [n]$, party P_0 also learns VTS commitments $C_i^{(0)}$ with timing hardness $\mathbf{T}_0$, which embeds a valid signature on a redeem transaction $tx^{(0)}_{\mathsf{rfnd},i}$ that spends all the coins from $pk_i^{(01)}$ to some key of P_0. This is to ensure that coins $v_i^{(0)}$ are not forever locked in the joint key $pk_i^{(01)}$. Similarly, for every $k \in [\tilde{n}]$, P_1 also learn VTS commitments $C_k^{(1)}$ with timing hardness $\mathbf{T}_1$ that embeds a signature which lets P_1 redeem the coins from $pk_k^{(10)}$ to itself.

In this phase, both parties additionally learn commitments to the joint secret keys corresponding to the joint public keys. Each party receives a share of this secret key and a share of the opening information of the corresponding commitment. We require this to ensure that the parties input consistent values during the swap lock phase, which is next.

Swap Lock Phase. During this phase, the parties run a 2 party protocol Γ_3 for $k \in [\tilde{n}]$ times, each time for a n-to-1 swap. That is, all the coins $v_i^{(0)}$ for the k-th coin $v_k^{(1)}$. The parties generate appropriate swap transactions ($tx^{(1)}_{\mathsf{swp},i}$ and $tx^{(0)}_{\mathsf{swp},k}$ for $i \in [n]$ and $k \in [\tilde{n}]$) and run Γ_3 to generate the lock ℓk_k and the signature $\sigma^{(0)}_{\mathsf{swp},k}$ where $\sigma^{(0)}_{\mathsf{swp},k}$ is a valid signature on $tx^{(0)}_{\mathsf{swp},k}$ (transaction spending $v_k^{(1)}$ from $pk_k^{(10)}$ to some key of P_0) under the public key $pk_k^{(10)}$. Internally, Γ_3 checks if the parties input the correct secret key shares by checking their input with respect to the commitments that were generated in the freeze coin phase. At the end of the phase, party P_1 receives $(\ell k_1, \ldots, \ell k_{\tilde{n}})$ while party P_0 receives $\left(\sigma^{(0)}_{\mathsf{swp},1}, \ldots, \sigma^{(0)}_{\mathsf{swp},\tilde{n}}\right)$.

Swap Complete Phase. During the swap complete phase, party P_0 initiates the swap by posting the transaction $tx^{(0)}_{\mathsf{swp},k}$ and the associated signature $\sigma^{(0)}_{\mathsf{swp},k}$ for $k \in [\tilde{n}]$. Party P_1 can complete the swap by using $\sigma^{(0)}_{\mathsf{swp},k}$ and ℓk_k to run mRel algorithm of Π_{MLS}, and learning $\left\{\sigma^{(1)}_{\mathsf{swp},1}, \ldots, \sigma^{(1)}_{\mathsf{swp},n}\right\}$. With these signatures, party P_1 can complete the swap by posting $tx^{(1)}_{\mathsf{swp},i}$ and $\sigma^{(1)}_{\mathsf{swp},i}$ for $i \in [n]$ on the blockchain.

Swap Time-out Phase. If party P_0 does not post a swap transaction with a valid signature before time $\mathbf{T}_1$ or if P_1 does not posts its swap transaction and a signature before time $\mathbf{T}_0$, the other party by then force opens the VTS commitment of the corresponding coin. Using the redeem transaction for that coin, and the force opened signature, the party can redeem the coins to itself during this time-out phase.

The security of our generic construction for atomic swap as described in Figure 15 is stated in the following theorem. The formal analysis of the theorem can be found in Appendix H5.

Theorem A.2: Let Π_{MLS} be a hiding and an unlockable multi-lock signature scheme with respect to the signature scheme Π_{DS} that is strongly unforgeable. Let $(\mathsf{Commit}, \mathsf{Open})$ be a UC-secure commitment scheme and $\Gamma_1, \Gamma_2, \Gamma_3$ be UC-secure 2PC for general functions. Let Π_{VTS} be a timed private and sound verifiable timed signature scheme. Then, the atomic swap protocol described in Figure 15 with access to $(\mathcal{F}_{\mathbb{B}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{clock}})$ UC-realizes the functionality $\mathcal{F}_{\mathsf{swap}}$.

Proof 7 (Proof of Theorem A.2): We now prove that our protocol in Figure 15 securely UC-realizes the atomic swap functionality from Figure 3.

We describe a simulator $\mathcal{S}$ that handles either of the parties P_0 or P_1 that are corrupted by a PPT $\mathcal{A}$ and simulates the real world execution protocol while interacting with the ideal functionality $\mathcal{F}_{\mathsf{swap}}$. We have a static corruption where the environment $\mathcal{E}$ at the beginning of a session specifies the corrupted parties and the honest parties. The simulator $\mathcal{S}$ faithfully impersonates the honest party. For operations exclusively among corrupted users, the environment does not expect any interaction with the simulator. Similarly, communications exclusively among honest nodes happen through secure channels and therefore the attacker does not gather any additional information other than the fact that the communication took place. For simplicity, we omit these operations in the description of our simulator. The random oracle H is simulated by $\mathcal{S}$ via lazy-sampling. The operations to be simulated for a n-to-$\tilde{n}$ atomic swap are described in the following.

In describing $\mathcal{S}$'s operations for swapping, we begin by describing a series of hybrid executions, where we begin with a real world execution and gradually change the simulation in these hybrids and then we argue about the proximity of neighbouring experiments. Simulator $\mathcal{S}$'s execution for the payment operation is defined as the final hybrid's execution. Below we describe the hybrid executions first and later argue about their proximity. Note that the switching of hybrid executions is performed over every session, but one at a time and we only discuss here a single time for simplicity and readability.

Hybrid $\mathcal{H}_0$: This is the same as the real execution of the protocol in Figure 15.

Hybrid $\mathcal{H}_1$: This is the same as the above execution except now the 2PC protocol Γ_1 in the freezing coins of swap setup phase to generate shared keys is simulated using $\mathcal{S}_{\mathsf{2pc},1}$ for the corrupted parties. Rest of the execution is unchanged as in the previous hybrid.

Input: First party U_0 has inputs $(pk, sk, v, \mathbf{T})$ and second party U_1 has inputs $(pk, v, \mathbf{T})$. Parties U_0 and U_1 do the following:

1) They run a 2PC protocol Γ_1 that computes the following functionality:
   - Compute $(\mathsf{tpk}, \mathsf{tsk}) \leftarrow \Pi_{\mathsf{DS}}.\mathsf{KGen}(1^\lambda)$ and $(c, d) \leftarrow \Pi_{\mathsf{C}}.\mathsf{Commit}\left(1^\lambda, \mathsf{tsk}\right)$.
   - Sample uniformly at random $(\mathsf{tsk}_0, \mathsf{tsk}_1)$ and (d_0, d_1) such that $\mathsf{tsk} = \mathsf{tsk}_0 \oplus \mathsf{tsk}_1$ and $d = d_0 \oplus d_1$.
   - U_0 receives $(\mathsf{tpk}, c, \mathsf{tsk}_0, d_0)$ as output and U_1 receives $(\mathsf{tpk}, c, \mathsf{tsk}_1, d_1)$ as output
2) Party U_0 generates $(pk_{\mathsf{rfnd}}, sk_{\mathsf{rfnd}}) \leftarrow \Pi_{\mathsf{DS}}.\mathsf{KGen}(1^\lambda)$.
3) Parties U_0 and U_1 generate $tx_{\mathsf{rfnd}} := tx\,(\mathsf{tpk}, pk_{\mathsf{rfnd}}, v)$.
4) They run a 2PC protocol Γ_2 to compute $\sigma_{\mathsf{rfnd}} \leftarrow \pi_{\mathsf{DS}}.\mathsf{Sign}\,(\mathsf{tsk}_0 \oplus \mathsf{tsk}_1, tx_{\mathsf{rfnd}})$, such that U_1 receives σ_{rfnd} as output, while U_0 receives nothing.
5) Party U_1 generates $(C, \pi) \leftarrow \Pi_{\mathsf{VTS}}.\mathsf{Commit}\,(\sigma_{\mathsf{rfnd}}, \mathbf{T})$ and gives (C, π) to party U_0.
6) Party U_0 checks if $\Pi_{\mathsf{VTS}}.\mathsf{Verify}\,(\mathsf{tpk}, tx_{\mathsf{rfnd}}, C, \pi) = 1$, and aborts otherwise.
7) If the above check is successful, party U_0 generates $tx_{\mathsf{frz}} := tx\,(pk, \mathsf{tpk}, v)$ and a signature $\sigma_{\mathsf{frz}} \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Sign}\,(sk, tx_{\mathsf{frz}})$. It sends $(tx_{\mathsf{frz}}, \sigma_{\mathsf{frz}})$ to party U_1.
8) Party U_0 sets its output as $(tx_{\mathsf{frz}}, \sigma_{\mathsf{frz}}, tx_{\mathsf{rfnd}}, \mathsf{tpk}, c, \mathsf{tsk}_0, d_0, C, \pi)$ and party U_1 sets its output as
   $(tx_{\mathsf{frz}}, \sigma_{\mathsf{frz}}, tx_{\mathsf{rfnd}}, \mathsf{tpk}, c, \mathsf{tsk}_1, d_1)$.

Fig. 13: User U_0 and U_1 running freeze during a atomic swap protocol

Swap Complete Phase

Party P_0 posts the transactions and signatures $\left(tx^{(0)}_{\mathsf{swp},i}, \sigma^{(0)}_{\mathsf{swp},i}\right)$ for all $i \in [\tilde{n}]$ on the blockchain $\mathbb{B}$.

Party P_1 observes $\left(tx^{(0)}_{\mathsf{swp},j}, \sigma^{(0)}_{\mathsf{swp},j}\right)$ for all $j \in [\tilde{n}]$ on the blockchain $\mathbb{B}$.

1) It then picks $j' \in [\tilde{n}]$ and computes $S \leftarrow \mathsf{mRel}\left(\ell k_{j'}, \left\{tx^{(1)}_{\mathsf{swp},i}\right\}_{i\in[n]}, tx^{(0)}_{\mathsf{swp},j'}, \left\{pk^{(01)}_i\right\}_{i\in[n]}, pk^{(10)}_{j'}\right)$.
2) It parses $S := \left\{\sigma^{(1)}_{\mathsf{swp},1}, \ldots, \sigma^{(1)}_{\mathsf{swp},n}\right\}$
3) It posts $\left(tx^{(1)}_{\mathsf{swp},i}, \sigma^{(1)}_{\mathsf{swp},i}\right)$ for all $i \in [n]$ on the blockchain $\mathbb{B}$.

Swap Time-out Phase

1) If party P_0 fails to post a transaction-signature pair $\left(tx^{(0)}_{\mathsf{swp},j}, \sigma^{(0)}_{\mathsf{swp},j}\right)$ for any $j \in [\tilde{n}]$ on chain before time $\mathbf{T}_1$, party P_1 does the following:
   - Finish computing $\sigma^{(1)}_{\mathsf{rfnd},j} \leftarrow \Pi_{\mathsf{VTS}}.\mathsf{ForceOp}\left(C^{(1)}_j\right)$.
   - Reclaim the $v^{(1)}_j$ coins by posting $\left(tx^{(1)}_{\mathsf{rfnd},j}, \sigma^{(1)}_{\mathsf{rfnd},j}\right)$ on the blockchain $\mathbb{B}$.
2) Similarly, if party P_1 fails to post a transaction-signature pair $\left(tx^{(1)}_{\mathsf{swp},j}, \sigma^{(1)}_{\mathsf{swp},j}\right)$ for any $j \in [n]$ on chain before time $\mathbf{T}_0$, party P_0 does the following:
   - Finish computing $\sigma^{(0)}_{\mathsf{rfnd},j} \leftarrow \Pi_{\mathsf{VTS}}.\mathsf{ForceOp}\left(C^{(0)}_j\right)$.
   - Reclaim the $v^{(0)}_j$ coins by posting $\left(tx^{(0)}_{\mathsf{rfnd},j}, \sigma^{(0)}_{\mathsf{rfnd},j}\right)$ on the blockchain $\mathbb{B}$.

Fig. 14: Swap complete and time-out phase of the generic atomic swap protocol run between parties P_0 and P_1

Hybrid $\mathcal{H}_2$: This is the same as the above execution except now the 2PC protocol Γ_2 in the freezing coins of swap setup phase to generate σ_{rfnd} is simulated using $\mathcal{S}_{\mathsf{2pc},2}$ for the corrupted parties.

Hybrid $\mathcal{H}_3$: This is the same as the above execution except now the 2PC protocol Γ_3 in the swap lock phase for generating $\left(\ell k_k, \sigma^{(0)}_{\mathsf{swp},k}\right)$ for $k \in [\tilde{n}]$ is simulated using $\mathcal{S}_{\mathsf{2pc},3}$ for the corrupted parties. The simulator ensures that ordering is maintained, that is party P_1 receives ℓk_k first before party P_0 receives $\sigma^{(0)}_{\mathsf{swp},k}$.

Hybrid $\mathcal{H}_4$: This is the same as the above execution except now the function output (given as input to the 2PC simulator $\mathcal{S}_{\mathsf{2pc},3}$) is computed differently. Specifically, for all $k \in [\tilde{n}]$, we no longer perform the following checks:

- For all $j \in [n]$, $sk^{(01)}_{0,j} \oplus sk^{(01)}_{1,j} \overset{?}{=} \Pi_{\mathsf{C}}.\mathsf{Open}\left(d^{(01)}_{0,j} \oplus d^{(01)}_{1,j}, c^{(01)}_j\right)$
- $sk^{(10)}_{0,k} \oplus sk^{(10)}_{1,k} \overset{?}{=} \Pi_{\mathsf{C}}.\mathsf{Open}\left(d^{(10)}_{0,k} \oplus d^{(10)}_{1,k}, c^{(10)}_k\right)$

Instead check if the secret key share input by the adversary is equal to the adversarial key obtained from the freeze coins phase. Specifically, check if $sk^{(01)}_{0,j} \overset{?}{=} sk^{(01)}_{\mathcal{A},j}$ and $sk^{(10)}_{0,j} \overset{?}{=} sk^{(10)}_{\mathcal{A},j}$ where $sk^{(01)}_{\mathcal{A},j}$ and $sk^{(10)}_{\mathcal{A},j}$ were received by the adversary

Global input: $\left\{v_i^{(0)}, pk_i^{(0)}\right\}_{i\in[n]}, \left\{v_i^{(1)}, pk_i^{(1)}\right\}_{i\in[\tilde{n}]}, \mathbf{T}_0, \mathbf{T}_1, \Delta$. Here $\mathbf{T}_0, \mathbf{T}_1, \Delta \in \mathbb{N}$ and $\mathbf{T}_1 = \mathbf{T}_0 - \Delta$.

Party P_0's input: $\left\{sk_i^{(0)}\right\}_{i\in[n]}$

Party P_1's input: $\left\{sk_i^{(1)}\right\}_{i\in[\tilde{n}n]}$

Swap Setup Phase - Freezing coins

Parties P_0 and P_1 freeze the coins that they want to swap by doing the following:

1) For $i \in [n]$, the parties do the following
   - party P_0 plays the role of U_0 and P_1 plays the role of U_1 in running freeze algorithm (Figure 13) where P_0 has inputs $\left(pk_i^{(0)}, sk_i^{(0)}, v_i^{(0)}, \mathbf{T}_0\right)$ and P_1 has inputs $\left(pk_i^{(0)}, v_i^{(0)}, \mathbf{T}_0\right)$.
   - At the end, P_0 receives as output $\left(tx^{(0)}_{\mathsf{frz},i}, \sigma^{(0)}_{\mathsf{frz},i}, tx^{(0)}_{\mathsf{rfnd},i}, pk_i^{(01)}, c_i^{(01)}, sk^{(01)}_{0,i}, d^{(01)}_{0,i}\, tx^{(0)}_{\mathsf{rfnd},i}, C_i^{(0)}, \pi_i^{(0)}\right)$ and P_1 receives $\left(tx^{(0)}_{\mathsf{frz},i}, \sigma^{(0)}_{\mathsf{frz},i}, tx^{(0)}_{\mathsf{rfnd},i}, pk_i^{(01)}, c_i^{(01)}, sk^{(01)}_{1,i}, d^{(01)}_{1,i}\right)$. Note that we have the secret key corresponding to $pk_i^{(01)}$ as $sk_i^{(01)} = sk^{(01)}_{0,i} \oplus sk^{(01)}_{1,i}$.
2) For $i \in [\tilde{n}]$, the parties do the following
   - Party P_0 plays the role of U_1 and party P_1 plays the role of U_0 in running freeze algorithm (Figure 13) where P_0 has inputs $\left(pk_i^{(1)}, v_i^{(1)}, \mathbf{T}_1\right)$ and P_1 has inputs $\left(pk_i^{(1)}, sk_i^{(1)}, v_i^{(1)}, \mathbf{T}_1\right)$.
   - At the end, P_0 receives as output $\left(tx^{(1)}_{\mathsf{frz},i}, \sigma^{(1)}_{\mathsf{frz},i}, tx^{(1)}_{\mathsf{rfnd},i}, pk_i^{(10)}, c_i^{(10)}, sk^{(10)}_{0,i}, d^{(10)}_{0,i}\right)$ and P_1 receives $\left(tx^{(1)}_{\mathsf{frz},i}, \sigma^{(1)}_{\mathsf{frz},i}, tx^{(1)}_{\mathsf{rfnd},i}, pk_i^{(10)}, c_i^{(10)}, sk^{(10)}_{1,i}, d^{(10)}_{1,i}, C_i^{(1)}, \pi_i^{(1)}\right)$. Note that we have the secret key corresponding to $pk_i^{(10)}$ as $sk_i^{(10)} = sk^{(10)}_{0,i} \oplus sk^{(10)}_{1,i}$.
3) Parties post $\left(\left(tx^{(0)}_{\mathsf{frz},1}, \sigma^{(0)}_{\mathsf{frz},1}\right), \ldots, \left(tx^{(0)}_{\mathsf{frz},n}, \sigma^{(0)}_{\mathsf{frz},n}\right), \left(tx^{(1)}_{\mathsf{frz},1}, \sigma^{(1)}_{\mathsf{frz},1}\right), \ldots, \left(tx^{(1)}_{\mathsf{frz},\tilde{n}}, \sigma^{(1)}_{\mathsf{frz},\tilde{n}}\right)\right)$ on the blockchain $\mathbb{B}$.
4) Party P_0 starts computing $\Pi_{\mathsf{VTS}}.\mathsf{ForceOp}\left(C_i^{(0)}\right)$ for $i \in [n]$ while party P_1 starts computing $\Pi_{\mathsf{VTS}}.\mathsf{ForceOp}\left(C_i^{(1)}\right)$ for $i \in [\tilde{n}]$.

Swap Lock Phase - n-to-$\tilde{n}$ swap

1) Parties P_0 and P_1 generate swap locks by first doing the following:
   - Party P_1 generates $\left(pk^{(1)}_{\mathsf{swpd},i}, sk^{(1)}_{\mathsf{swpd},i}\right) \leftarrow \Pi_{\mathsf{DS}}.\mathsf{KGen}(1^\lambda)$ for $i \in [n]$.
   - Party P_1 generates $tx^{(1)}_{\mathsf{swp},i} := tx\left(pk_i^{(01)}, pk^{(1)}_{\mathsf{swpd},i}, v_i^{(0)}\right)$ for $i \in [n]$ and sends it to party P_0.
2) Then for each $k \in \tilde{n}$ the parties run an instance of the lock generation for a n-to-1 swap. The k-th instance is as follows.
   - Party P_0 generates $\left(pk^{(0)}_{\mathsf{swpd},k}, sk^{(0)}_{\mathsf{swpd},k}\right) \leftarrow \Pi_{\mathsf{DS}}.\mathsf{KGen}(1^\lambda)$.
   - Party P_0 goes on to generate $tx^{(0)}_{\mathsf{swp},k} := tx\left(pk_k^{(10)}, pk^{(0)}_{\mathsf{swpd},k}, v_k^{(1)}\right)$ and sends it to party P_1.
   - Parties P_0 and P_1 run a 2PC protocol Γ_3 to compute $\ell k_k \leftarrow \mathsf{mLock}\left(n, \left\{\sigma^{(1)}_{\mathsf{swp},j}\right\}_{j\in[n]}, \sigma^{(0)}_{\mathsf{swp},k}\right)$. More precisely, the 2PC protocol does the following:
     - Take as common input $\left(\left\{pk_j^{(01)}, c_j^{(01)}\right\}_{j\in[n]}, pk_k^{(10)}, c_k^{(10)}\right)$ from the parties, and as private input $\left(\left\{sk^{(01)}_{0,j}, d^{(01)}_{0,j}\right\}_{j\in[n]}, sk^{(10)}_{0,k}, d^{(10)}_{0,k}\right)$ from party P_0, and as private input $\left(\left\{sk^{(01)}_{1,j}, d^{(01)}_{1,j}\right\}_{j\in[n]}, sk^{(10)}_{1,k}, d^{(10)}_{1,k}\right)$ from party P_1.
     - For all $j \in [n]$, check whether $sk^{(01)}_{0,j} \oplus sk^{(01)}_{1,j} = \Pi_{\mathsf{C}}.\mathsf{Open}\left(d^{(01)}_{0,j} \oplus d^{(01)}_{1,j}, c_j^{(01)}\right)$ and abort otherwise.
     - Check whether $sk^{(10)}_{0,k} \oplus sk^{(10)}_{1,k} = \Pi_{\mathsf{C}}.\mathsf{Open}\left(d^{(10)}_{0,k} \oplus d^{(10)}_{1,k}, c_k^{(10)}\right)$ and abort otherwise.
     - For $j \in [n]$, compute $\sigma^{(1)}_{\mathsf{swp},j} \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Sign}\left(sk^{(01)}_{0,j} \oplus sk^{(01)}_{1,j}, tx^{(1)}_{\mathsf{swp},j}\right)$.
     - Compute $\sigma^{(0)}_{\mathsf{swp},k} \leftarrow \Pi_{\mathsf{DS}}.\mathsf{Sign}\left(sk^{(10)}_{0,k} \oplus sk^{(10)}_{1,k}, tx^{(0)}_{\mathsf{swp},k}\right)$.
     - Compute $\ell k_k \leftarrow \mathsf{mLock}\left(n, \left\{\sigma^{(1)}_{\mathsf{swp},j}\right\}_{j\in[n]}, \sigma^{(0)}_{\mathsf{swp},k}\right)$.
     - The 2PC protocol Γ_3 first outputs ℓk_k to P_1.
     - If the above step is successful, Γ_3 outputs $\sigma^{(0)}_{\mathsf{swp},k}$ to P_0.
3) In the end, party P_0 obtains $\left(\sigma^{(0)}_{\mathsf{swp},1}, \ldots, \sigma^{(0)}_{\mathsf{swp},\tilde{n}}\right)$ while party P_1 obtains $(\ell k_1, \ldots, \ell k_{\tilde{n}})$.

Fig. 15: Generic Atomic Swap protocol run between parties P_0 and P_1

as output from the freeze coin phases when we have the adversary corrupting P_0 (analogous case when P_1 is corrupted). Here $sk_{0,j}^{(01)}$ and $sk_{0,j}^{(10)}$ are the inputs of the adversary to the 2PC protocol Γ_3. The simulator extracts these inputs by interacting with the ideal functionality of the 2PC protocol of Γ_3. The simulator uses $\mathcal{S}_{\mathsf{2pc},3}$ to simulate the 2PC of Γ_3 with this new check.

Hybrid $\mathcal{H}_5$: Is the same execution as $\mathcal{H}_4$, except now for all the shares of the opening information that adversary receives in the freeze coin phase are switched to random strings. Specifically, $d_{\mathcal{A},i}^{(01)}$ for all $i \in [n]$ and $d_{\mathcal{A},j}^{(10)}$ for all $j \in [\tilde{n}]$ that the adversary receives from the freeze coin are switched to random strings.

Hybrid $\mathcal{H}_6$: Is the same execution as $\mathcal{H}_5$, except now for all the commitments $c_i^{(01)}$ for all $i \in [n]$ and $c_j^{(10)}$ for all $j \in [\tilde{n}]$ that are generated by the freeze coin phase are replaced with commitments to 0^λ.

Hybrid $\mathcal{H}_7$: Is the same execution as $\mathcal{H}_6$, except now for all $k \in [\tilde{n}]$ the lock ℓk_k is hardcoded in the simulation by hardcoding first the signatures $\sigma_{\mathsf{swp},i}^{(1)}$ for $i \in [n]$ and $\sigma_{\mathsf{swp},j}^{(0)}$ for $j \in [\tilde{n}]$ in the simulation. Here $\sigma_{\mathsf{swp},i}^{(1)}$ is a signature on $tx_{\mathsf{swp},i}^{(1)}$ under the key $pk_i^{(01)}$ and $\sigma_{\mathsf{swp},j}^{(0)}$ is a signature on $tx_{\mathsf{swp},j}^{(0)}$ under they key $pk_j^{(10)}$.

Hybrid $\mathcal{H}_8$: Is the same execution as $\mathcal{H}_7$, except now for an honest P_1, if in some q_8-th session, the adversary outputs σ^* such that $\Pi_{\mathsf{DS}}.\mathsf{Vf}\left(pk_k^{(10)}, tx_{\mathsf{swp},k}^{(0)}, \sigma^*\right) = 1$ for some $k \in [\tilde{n}]$, and

$$S \leftarrow \mathsf{mRel}\left(\ell k_k, \left\{tx_{\mathsf{swp},i}^{(1)}\right\}_{i\in[n]}, tx_{\mathsf{swp},k}^{(0)}, \left\{pk_{i\in[n]}^{(01)}\right\}_{i\in[n]}, pk_k^{(10)}\right)$$

. Parse

$$S := \left\{\sigma_{\mathsf{swp},1}^{(1)}, \ldots, \sigma_{\mathsf{swp},n}^{(1)}\right\}$$

. And if there exists some $j \in [n]$

$$\Pi_{\mathsf{DS}}.\mathsf{Vf}\left(pk_j^{(01)}, tx_{\mathsf{swp},j}^{(1)}, \sigma_{\mathsf{swp},j}^{(1)}\right) = 0$$

, the simulation aborts by outputting a special symbol $\mathtt{abort}_1$. This abort happens when the adversary outputs a valid signature such that honest user P_1 pays the adversary through $tx_{\mathsf{swp},k}^{(0)}$ but is unable to complete the swap by computing a valid signature on all the transactions $\left\{tx_{\mathsf{swp},1}^{(1)}, \ldots, tx_{\mathsf{swp},n}^{(1)}\right\}$.

Hybrid $\mathcal{H}_9$: Is the same execution as $\mathcal{H}_8$ except now, if in some q_9-th session, before an honest P_0 initiates the swap, adversary $\mathcal{A}$ outputs a signature σ^* such that for some $i \in [n]$

$$\Pi_{\mathsf{DS}}.\mathsf{Vf}\left(pk_i^{(01)}, tx_{\mathsf{swp},i}^{(1)}, \sigma^*\right) = 1$$

the simulation aborts by outputting a special symbol $\mathtt{abort}_2$. This abort happens when the adversary is able to release a lock correctly before P_0 initiates the swap process.

Hybrid $\mathcal{H}_{10}$: Is the same execution as $\mathcal{H}_9$ except now, if in some session q_{10}, the adversarial party (for example P_0) posts $\left(tx_{\mathsf{rfnd},i}^{(0)}, \sigma_{\mathsf{rfnd},i}^{(0)}\right)$ on chain before time $\mathbf{T}_0$ for any $i \in [n]$, the simulator aborts by outputting $\mathtt{abort}_3$. Analogous case when P_1 is corrupt and it posts $\left(tx_{\mathsf{rfnd},k}^{(1)}, \sigma_{\mathsf{rfnd},k}^{(1)}\right)$ on chain before time $\mathbf{T}_1$ for any $k \in [\tilde{n}]$.

Hybrid $\mathcal{H}_{11}$: Is the same execution as $\mathcal{H}_{10}$ except now, if for example P_0 is corrupt and in some session q_{11} the simulator obtains $\sigma_{\mathsf{rfnd},i}^{(1)} \leftarrow \Pi_{\mathsf{VTS}}.\mathsf{ForceOp}\left(C_i^{(1)}\right)$ for some $i \in [\tilde{n}]$, and

$$\Pi_{\mathsf{DS}}.\mathsf{Vf}\left(pk_i^{(10)}, tx_{\mathsf{rfnd},i}^{(1)}, \sigma_{\mathsf{rfnd},i}^{(1)}\right) = 0$$

, the simulator aborts by outputting $\mathtt{abort}_4$. Analogous case if P_1 is corrupt.

Simulator $\mathcal{S}$: The execution of the simulator is defined as the execution in $\mathcal{H}_{11}$ while it interacts with the ideal functionality $\mathcal{F}_{\mathsf{swap}}$. The simulator receives $(\mathtt{swap}_1, \mathsf{id}, V, \tilde{V}, \mathsf{PK}, \tilde{\mathsf{PK}}, \tilde{\mathsf{SK}}, U_1)$ and $(\mathtt{swap}_2, \mathsf{id}, V, \tilde{V}, \mathsf{PK}, \tilde{\mathsf{PK}}, \mathsf{SK}, U_0)$ from $\mathcal{F}_{\mathsf{swap}}$. It proceeds as in the execution of $\mathcal{H}_9$ by simulating the view of the adversary appropriately as it receives messages from the ideal functionality $\mathcal{F}_{\mathsf{swap}}$. If the simulated view deviates from the execution of the ideal functionality, we note that the simulation must have already aborted (as discussed in cases of abort in the above hybrids).

Below we discuss the indistinguishability arguments and we use the notation $\approx_c, \approx_{\mathbf{T}}$ to denote computational indistinguishability for a PPT algorithm, and indistinguishability for depth $\mathbf{T}$ bounded algorithms, respectively.

$\mathcal{H}_0 \approx_c \mathcal{H}_1$ and $\mathcal{H}_1 \approx_c \mathcal{H}_2$ and $\mathcal{H}_2 \approx_c \mathcal{H}_3$: the indistinguishability follow from the security of the 2PC protocols in the freeze coin phase and in the swap lock phase. Security of the 2PC protocol for the derivation of keys guarantees the existence of $\mathcal{S}_{\mathsf{2pc},1}$, the security of the 2PC protocol for the signature generation guarantees the existence of $\mathcal{S}_{\mathsf{2pc},2}$, and the security of the 2PC protocol for the lock generation algorithm, which guarantees the existence of $\mathcal{S}_{\mathsf{2pc},3}$.

$\mathcal{H}_3 \equiv \mathcal{H}_4$: Notice that given the commitment scheme Π_{C} is perfectly binding with unique openings, the old checks (in $\mathcal{H}_3$) pass if and only if the new checks (in $\mathcal{H}_4$) pass. This means that the function computed by the 2PC simulator in both hybrids is equivalent and therefore the two hybrids $\mathcal{H}_3$ and $\mathcal{H}_4$ are identical. To be more precise, we can set intermediate hybrid executions where we switch the checks one after the other for $n \cdot \tilde{n}$ executions of mLock. We can argue about the equivalence of these intermediate hybrids by the perfect binding property of the commitment scheme Π_{C}.

$\mathcal{H}_4 \equiv \mathcal{H}_5$: Notice that in $\mathcal{H}_4$ as in $\mathcal{H}_5$ the adversary simply sees random shares of the opening information of commitment $c_i^{(01)}$ for $i \in [n]$ and $c_j^{(10)}$ for $j \in [\tilde{n}]$. Therefore the view of the adversary in this hybrid is identical to its view in $\mathcal{H}_5$.

$\mathcal{H}_5 \approx_c \mathcal{H}_6$: Given that the only difference between the hybrids is how the commitments $c_i^{(01)}$ for $i \in [n]$ and $c_j^{(10)}$ for $j \in [\tilde{n}]$ are generated, the indistinguishability of the two hybrids $\mathcal{H}_5$ and $\mathcal{H}_6$ follows from the computational hiding property of the commitment scheme Π_{C}.

$\mathcal{H}_6 \equiv \mathcal{H}_7$: Notice that the change made in hybrid $\mathcal{H}_7$ does not affect the functionality of the computation. We only change the way the output of the simulated 2PC is generated and therefore the execution in $\mathcal{H}_6$ and $\mathcal{H}_7$ are identical.

$\mathcal{H}_7 \approx_c \mathcal{H}_8$: The only change in hybrid $\mathcal{H}_8$ is the abort in the simulation with the special symbol $\texttt{abort}_1$. Notice that this event occurs when the adversary outputs σ^* such that

$$\Pi_{\mathsf{DS}}.\mathsf{Vf}\left(pk_k^{(10)}, tx_{\mathsf{swp},k}^{(0)}, \sigma^*\right) = 1$$

for some $k \in [\tilde{n}]$, and

$$S \leftarrow \mathsf{mRel}\left(\ell k_k, \left\{tx^{(1)}_{\mathsf{swp},i}\right\}_{i\in[n]}, tx^{(0)}_{\mathsf{swp},k}, \left\{pk^{(01)}_{i\in[n]}\right\}_{i\in[n]}, pk_k^{(10)}\right)$$

. Parse $S := \left\{\sigma^{(1)}_{\mathsf{swp},1}, \ldots, \sigma^{(1)}_{\mathsf{swp},n}\right\}$ and if there exists some $j \in [n]$ $\Pi_{\mathsf{DS}}.\mathsf{Vf}\left(pk_j^{(01)}, tx^{(1)}_{\mathsf{swp},j}, \sigma^{(1)}_{\mathsf{swp},j}\right) = 0$ We now show that the probability with which this abort event occurs is at most negligible in the security parameter, i.e., $\Pr[\texttt{abort}_1|\mathcal{H}_8] \leq \mathsf{negl}(\lambda)$. To show this, we construct a reduction $\mathcal{R}$ against the unlockability property of the lockable signatures.

We in fact reduce to a weaker notion of unlockability where the adversary in $\mathsf{EUn\text{-}CMA}$ is only given oracle access to $\mathsf{Sign}(sk_1, \cdot), \ldots, \mathsf{Sign}(sk_n, \cdot)$ (apart from $\mathsf{Sign}(\tilde{sk}, \cdot)$) and the public keys $(pk_1, \ldots, pk_n, \tilde{pk})$. In this regard consider the reduction $\mathcal{R}$ that is given as input public keys $(pk_1, \ldots, pk_n, \tilde{pk})$. The reduction guesses the session q_8 where the adversary triggers the abort $\texttt{abort}_1$ and it guesses an index $k \in [\tilde{n}]$. It simulates the execution of $\mathcal{H}_8$ faithfully except it sets $pk_i^{(01)} = pk_i$ for $i \in [n]$ and $pk_k^{(10)} = \tilde{pk}$. Rest of the keys are set according to $\mathcal{H}_8$. The reduction outputs M and $\tilde{m}$ to its challenger, where $M := \left\{tx^{(1)}_{\mathsf{swp},1}, \ldots, tx^{(1)}_{\mathsf{swp},n}\right\}$ and $\tilde{m} := tx^{(0)}_{\mathsf{swp},k}$ and obtains a lock ℓk. The reduction sets $\ell k_k = \ell k$ and hardcodes in the simulation as done in $\mathcal{H}_8$. The adversary participating in the hybrid $\mathcal{H}_8$ outputs a signature σ^* and the reduction $\mathcal{R}$ outputs the same.

Clearly the reduction $\mathcal{R}$ runs in polynomial time as it only performs efficient simulation operations. Notice that the winning conditions of $\mathsf{EUn\text{-}CMA}$ are precisely the conditions that lead to an abort $\texttt{abort}_1$ in hybrid $\mathcal{H}_8$. It is easy to see that if the adversary triggers the abort event $\texttt{abort}_1$ with non-negligible probability in session q_1, the reduction also succeeds in winning $\mathsf{EUn\text{-}CMA}$ with the same non-negligible probability. Since this is a contradiction, it must be the case that $\texttt{abort}_1$ occurs with negligible probability.

$\mathcal{H}_8 \approx_c \mathcal{H}_9$: Notice that the only difference between the hybrids is in the abort event $\texttt{abort}_2$ in $\mathcal{H}_9$. The abort event occurs if in some q_2-th session the adversary is able to output σ^* such that for some $i \in [n]$ $\Pi_{\mathsf{DS}}.\mathsf{Vf}(pk_i^{(01)}, tx^{(1)}_{\mathsf{swp},i}, \sigma^*) = 1$. We will show that the probability with which the abort event is triggered is bound by a negligible probability with the help of Lemma 5.

Lemma 5: There exists a negligible function negl such that $\Pr[\texttt{abort}_2|\mathcal{H}_9] \leq \mathsf{negl}(\lambda)$

Proof 8 (Lemma 5): To show this, we consider the following hybrid executions. Note that these hybrid execution are only designed to show that the abort event $\texttt{abort}_2$ occurs with negligible probability. In other words, these hybrids do not feature in our main line of hybrids as we have above.

Hybrid $\mathcal{H}_{9,k}, k \in [\tilde{n}]$: Is the same execution as $\mathcal{H}_9$ where the sender has not yet initiated the release phase in the q_2-th session, except now the first k iterations of generating locks by the simulator is done by hardcoding the ℓk_k in the simulator $\mathcal{S}_{\mathsf{2pc},3}$ where the lock $\ell k_k \leftarrow \mathsf{Sim}(n, \mathsf{PK}, \tilde{pk}, M, \tilde{m})$, where $\mathsf{PK} := \left\{pk_1^{(01)}, \ldots, pk_n^{(01)}\right\}$, $\tilde{pk} := pk_k^{(10)}$, $M := \left\{tx^{(1)}_{\mathsf{swp},1}, \ldots, tx^{(1)}_{\mathsf{swp},n}, \ldots\right\}$, and $\tilde{m} := tx^{(0)}_{\mathsf{swp},k}$.

$\mathcal{H}_{9,k} \approx_c \mathcal{H}_{9,k+1}$: Let $\mathcal{H}_{9,0} = \mathcal{H}_9$ and the indistinguishability of hybrids $\mathcal{H}_{9,k}$ and $\mathcal{H}_{9,k+1}$ follows immediately from the hiding property of multi-lock signatures. To show this we construct a reduction algorithm $\mathcal{R}$ that plays in the hiding experiment $\mathsf{EHi\text{-}CMA}$ and runs a distinguisher (of the hybrids) as a sub-routine. The reduction gets as input $(\mathsf{PK}, \tilde{pk})$ where $\mathsf{PK} := \{pk_1, \ldots, pk_n\}$. It sets $pk_i^{(01)} := pk_i$ for $i \in [n]$ and $pk_{k+1}^{(10)} = \tilde{pk}$. The reduction outputs $M, \tilde{m}$ where $M := \left\{tx^{(1)}_{\mathsf{swp},1}, \ldots, tx^{(1)}_{\mathsf{swp},n}, \ldots\right\}$, and $\tilde{m} := tx^{(0)}_{\mathsf{swp},k+1}$ to its challenger and obtains a lock ℓk which it sets as $\ell k_{k+1} = \ell k$. The reduction hardcodes ℓk_{k+1} in the simulator $\mathcal{S}_{\mathsf{2pc},3}$. Rest of the simulation by $\mathcal{R}$ is according to $\mathcal{H}_{9,k}$. The distinguisher outputs a bit b' and the reduction outputs the same bit to its challenger.

Clearly the reduction $\mathcal{R}$ is efficient. To argue about the success probability, notice that if $b = 0$ in $\mathsf{EHi\text{-}CMA}$ the reduction $\mathcal{R}$ simulates $\mathcal{H}_{9,k}$, and if $b = 1$, the reduction $\mathcal{R}$ simulates $\mathcal{H}_{9,k+1}$. Therefore if the distinguisher outputs $b' = b$ with non-negligible probability more than half, the reduction succeeds with the same non-negligible probability more than half, which is a contradiction to the hiding property of the lockable signatures.

By standard hybrid argument we have that $\mathcal{H}_{9,0} \approx_c \mathcal{H}_{9,\tilde{n}}$. Notice that in $\mathcal{H}_{9,\tilde{n}}$, every lock is generated via Sim and does not leak any information about signatures or honest party's secret keys to the adversary. Since $\mathcal{H}_9$ is indistinguishable from $\mathcal{H}_{9,\tilde{n}}$, provided that P_0 has not initiated the swap in session q_2, we have that the probability with which the adversary triggers abort event $\texttt{abort}_2$ in $\mathcal{H}_9$ must be the same in $\mathcal{H}_{9,\tilde{n}}$ except with a negligible difference. For the adversary to trigger $\texttt{abort}_2$ in $\mathcal{H}_{9,\tilde{n}}$, it has to output σ^* on $tx^{(1)}_{\mathsf{rfnd},i}$ under $pk_i^{(01)}$ for some $i \in [n]$.

We now show that the probability with which the adversary can do this in $\mathcal{H}_{9,\tilde{n}}$ is at most negligible in the security parameter, i.e., $\Pr[\texttt{abort}_2|\mathcal{H}_{9,\tilde{n}}] \leq \mathsf{negl}(\lambda)$, by reducing the occurance of the event to the unforgeability of the signature scheme. We construct a reduction algorithm $\mathcal{R}'$ that guesses the q_2-th session and some index $i^* \in [n]$. It receives as input a public key pk and simulates the view for the adversary $\mathcal{A}$ faithfully as in $\mathcal{H}_{9,\tilde{n}}$ except that it sets $pk_{i^*}^{(01)} := pk$. If the adversary outputs a σ^* on $tx^{(1)}_{\mathsf{rfnd},i^*}$ under public key $pk_{i^*}^{(01)}$, the reduction $\mathcal{R}'$ simply outputs $(tx^{(1)}_{\mathsf{rfnd},i^*}, \sigma^*)$ as its forgery in the $\mathsf{EUF\text{-}CMA}$ game. Clearly $\mathcal{R}'$ is efficient and if $\mathcal{A}$ outputs such a valid signature and a message with non-negligible

probability, clearly $\mathcal{R}'$ wins $\mathsf{EUF\text{-}CMA}$ with the same non-negligible probability. This is a contradiction and therefore bounds the probability with which $\mathtt{abort}_2$ is triggered to be negligible. This proves that $\mathtt{abort}_2$ happens in $\mathcal{H}_{9,\tilde{n}}$ with negligible probability and since $\mathcal{H}_9 \approx_c \mathcal{H}_{9,\tilde{n}}$, we have that $\mathtt{abort}_2$ happens in $\mathcal{H}_9$ only with negligible probability. □
This proves our initial claim that $\mathcal{H}_8 \approx_c \mathcal{H}_9$.

$\underline{\mathcal{H}_9 \approx_c \mathcal{H}_{10}}$: The only difference between the hybrids is the abort event $\mathtt{abort}_3$ in $\mathcal{H}_{11}$ in some session q_3. The argument for the probability of this event being triggered being at most negligible follows immediately from Lemma 6.

Lemma 6: There exists a negligible function negl such that $\Pr[\mathtt{abort}_3|\mathcal{H}_{10}] \leq \mathsf{negl}(\lambda)$

Proof 9 (Lemma 6): To show this, we consider the following hybrid executions. Note that these hybrid executions are designed to prove the lemma and do not feature in our main line of hybrids of simulation. We consider the case of a corrupt P_0.

Hybrid $\mathcal{H}'_{10}$: Is the same execution as $\mathcal{H}_{10}$ in session q_3, except now the simulator chooses the keys $pk_i^{(01)}$ for $i \in [n]$ uniformly at random and chooses the adversarial shares of the secret keys also to be chosen uniformly at random. Rest of the execution is unchanged.

Hybrid $\mathcal{H}_{10,j}, j \in [n]$: Is the same execution as $\mathcal{H}'_{10}$ in session q_3, except now the first verifiable timed signatures are generated using the simulator $\mathcal{S}_{\mathsf{VTS}}$. That is, $\left(C_j^{(0)}, \pi_j^{(0)}\right) \leftarrow \mathcal{S}_{\mathsf{VTS}}\left(pk_j^{(01)}, tx_{\mathsf{rfnd},j}^{(0)}, \mathbf{T}_0\right)$. Rest of the execution is the same as in $\mathcal{H}'_{10}$.

$\underline{\mathcal{H}_{10} \equiv \mathcal{H}'_{10}}$: the hybrids are identical as the adversary only sees random shares of secret keys for randomly sampled shared keys.

$\underline{\mathcal{H}_{10,j} \approx_{\mathbf{T}_0} \mathcal{H}_{10,j+1}}$: the indistinguishability of the hybrids for a depth $\mathbf{T}_0$ bounded adversary follows from the timed privacy of Π_{VTS}.

To show this, we give a reduction $\mathcal{R}$ to the timed privacy of Π_{VTS} that uses the depth $\mathbf{T}_0$ bounded distinguisher of the hybrids as a sub-routine. The reduction gets as input pk. The reduction guesses $i \in [n]$ and sets $pk_i^{(01)} := pk$. It then sends $tx_{\mathsf{rfnd},i}^{(0)}$ to its challenger to get (pk, C, π). The reduction sets $C_i^{(0)} := C$ and $\pi_i^{(0)} := \pi$. The reduction simulates rest of the execution exactly as in $\mathcal{H}_{10,j}$.

The reduction is clearly efficient. To argue the success probability notice that if (C, π) was generated by $\Pi_{\mathsf{VTS}}.\mathsf{Commit}\left(\sigma_{\mathsf{rfnd},i}^{(0)}, \mathbf{T}_0\right)$ then the reduction has perfectly simulated $\mathcal{H}_{10,j}$ for the distinguisher. If we had $(C, \pi) \leftarrow \mathcal{S}_{\mathsf{VTS}}(pk, \mathbf{T}_0)$, then the reduction has simulated $\mathcal{H}_{10,j+1}$. Therefore if the depth $\mathbf{T}_0$ bounded distinguisher is able to distinguish between the hybrids with non-negligible probability, then the reduction break the timed privacy of Π_{VTS}.

By a standard hybrid argument we can see that $\mathcal{H}_{10} \approx_{\mathbf{T}_0} \mathcal{H}_{10,n}$. Notice that in $\mathcal{H}_{10,n}$ every VTS given to P_0 is generated by the simulator $\mathcal{S}_{\mathsf{VTS}}$. Therefore no information about the shares of the honest party P_1 is available in the view of the adversary. Since $\mathcal{H}_{10}$ is indistinguishable from $\mathcal{H}_{10,n}$ for depth $\mathbf{T}_0$ distinguishers, we have that in session q_3 the probability with which the adversary triggers $\mathtt{abort}_3$ in $\mathcal{H}_{10}$ must be negligibly close to the probability of it in $\mathcal{H}_{10,n}$. Now, for the adversary to trigger $\mathtt{abort}_3$ in $\mathcal{H}_{10,n}$, it has to output σ^* on the transaction $tx_{\mathsf{rfnd},i}^{(0)}$ under the public key $pk_i^{(01)}$ for some $i \in [n]$.

We show that the probability with which the adversary can do so in $\mathcal{H}_{10,n}$ is at most negligible in the security parameter by reducing the occurrence of the event to the unforgeability of the signature scheme Π_{DS}. We construct a reduction $\mathcal{R}'$ that guesses the q_3-th session, and the index $i^* \in [n]$. It receives as input a public key pk and simulates the view of the adversary $\mathcal{A}$ faithfully as in $\mathcal{H}_{10,n}$ except that it sets $pk_{i^*}^{(01)} = pk$. If the adversary outputs a signature σ^* on some transaction $tx_{\mathsf{rfnd},i}^{(0)}$ under pk, the reduction $\mathcal{R}'$ outputs the same signatures as its forgery in $\mathsf{EUF\text{-}CMA}$ game. We conclude that since the signature scheme Π_{DS} is unforgeable, we have that $\Pr[\mathtt{abort}_3|\mathcal{H}_{10,n}] \leq \mathsf{negl}(\lambda)$. This implies that $\Pr[\mathtt{abort}_3|\mathcal{H}_{10}] \leq \mathsf{negl}(\lambda)$ which proves the lemma.

□

$\underline{\mathcal{H}_{10} \approx_s \mathcal{H}_{11}}$: The only difference between the hybrids is the abort event $\mathtt{abort}_4$ in $\mathcal{H}_{11}$. The argument for the probability of this event being triggered being at most negligible follows immediately from the soundness of Π_{VTS}.

Our final simulator $\mathcal{S}$ for the channel payment is defined as the execution in $\mathcal{H}_{11}$ and this concludes the proof.