AuxChannel: Enabling Efficient Bi-Directional Channel for Scriptless Blockchains

Zhimei Sui
Monash Univerisity
Melbourne, Australia
zhimei.sui1@monash.edu

Joseph K. Liu
Monash Univerisity
Melbourne, Australia
joseph.liu@monash.edu

Jiangshan Yu
Monash Univerisity
Melbourne, Australia
jiangshan.yu@monash.edu

Man Ho Au
The University of Hong Kong
Hong Kong, China
allenau@cs.hku.hk

Jia Liu
Fetch.ai
UK
jia.liu@fetch.ai

ABSTRACT

Payment channels have been a promising solution to blockchain scalability. While payment channels for script-empowered blockchains (such as Bitcoin and Ethereum) have been well studied, developing payment channels for scriptless blockchains (such as Monero) is considered challenging. In particular, enabling bidirectional payment on scriptless blockchains remains an open challenge.

This work closes this gap by providing AuxChannel, the first bi-directional payment channel protocol for scriptless blockchains, meaning that building payment channels only requires the support of verifiably encrypted signature (aka adaptor signature) on the underlying blockchain. AuxChannel leverages verifiably encrypted signature to create a commitment for each off-chain payment and deploys a verifiable decentralised key escrow service to resolve dispute. To enable efficient construction of AuxChannel, we introduce a new cryptographic primitive, named *Consecutive Verifiably Encrypted Signature (CVES)*, as a core building block and it can also be of independent interest for other applications. We provide and implement a provably secure instantiation on Schnorr-based CVES. We also provide a formal security analysis on the security of the proposed AuxChannel.

KEYWORDS

Scriptless Blockchain, Payment channel, Scalability

1 INTRODUCTION

Payment channel [21] is a promising solution to improve the limited throughput of blockchain (in particular, Proof-of-Work blockchain such as Bitcoin). It allows users to make multiple transactions without committing all of them on-chain. In a typical payment channel, only two transactions, namely the first one representing the initial state and second one representing the final state of all transactions in between, are added to the blockchain while a large number of intermediate transactions can be executed between the participants without needing to be recorded on-chain. This greatly improves the blockchain throughput as only two transactions (representing a large number of transactions) need to go through the slow and expensive process of the distributed blockchain consensus [29].

Payment channels can be uni-directional or bi-directional. A uni-directional payment channel only enables one party to pay the other within a limited lifespan, whereas a bi-directional payment channel allows duplex payments. Thus, the uni-directional payment channel will be closed if the payer's balance is insufficient to make a new payment. In contract, bi-directional payment channels are more flexible, as it allows parties to pay each other and rebalance the channel.

Deploying either type of payment channels requires the underlying blockchain to support script. For unidirectional payment channel, the script is mainly used to unlock the payer's fund when the payee is not responding.

In particular, with unidirectional payment channel, as a payer performs incremential payment to a payee (by creating a transaction signed by the payer), the payee always has an incentive to close the channel with the latest state (where the payee has largest balance) by cross-signing the received payment. However, it is possible that the payee is not responding the request to close the channel, for example when the payee has never received any payment. In this case, the payer's balance is locked. To resolve this issue, a timelock function is desired to unlock the payer's deposit. Time-lock function (such as hashed time lock [30]) defines a timer, such that upon timeout the payer will get full refund of its channel balance unless the payee has reacted to it by closing the channel before timeout.

However, time lock is not sufficient to resolve dispute in bidirectional payment channels. Suppose Alice and Bob are establishing a bidirectional payment channel together. Initially, both parties deposit 5 coins in the channel respectively. After several sessions of payments, Alice has 10 coins and Bob has 0 coin. To close the channel, the final state (Alice has 10 coins and Bob has nothing) should be posted on chain. Two misbehaviours could happen: (1) if closing a channel requires the permission from both parties, Bob can refuse to close the channel, as he loses nothing but Alice would lose all the money; and (2) if it only requires a single side permission, Bob has the motivation to close the channel by using a revoked state (a state is revoked by another successful channel update triggered by a transaction within the channel), where Bob has more coins than his final balance at the closing state. A script contract to specify dispute policy is required to resolve the above cases.

Nevertheless, some blockchains do not support script language, and therefore cannot implement such dispute mechanism. We call such blockchains *scripless blockchains*. Deploying a bi-directional payment channel on scritpless blockchains, such as Monero and ZCash [43], remains an open challenge.

Our Contributions. Our papers contains the following merits:

(1) We propose **AuxChannel** in enabling efficient bi-directional payment channel for scriptless blockchains with unlimited lifespan. AuxChannel guarantees that *neither parties can prevent the channel from closing or steal coins from the counterpart*. We facilitate this by a secret release mechanism on a script-enabled blockchain. Namely, both parties commit their secrets to the script-enabled blockchain. When a party is not following the protocol, the counterpart can request the script-enabled blockchain to obtain the other's secret to recover the full signature for validating the payment. This naive approach is still not practical, as it not only incurs $O(n)$ (where n is the number of off-chain transactions) communication complexity with the script-enabled blockchain, but also requires both parties to update their secrets for each new payment through the smart contract on the script-enabled blockchain, thus making it expensive, slow, and non-scalable. We reduce the communication complexity to $O(1)$ and does not need the on-chain update for each payment by using novel cryptographic techniques. In AuxChannel, both parties only need to commit their initial secrets on the script-enabled blockchain. Later if a party is not following the protocol, the counterpart can request for the initial secret and self-derive the latest state to recover the full signature and thus validate the payment and close the channel.

(2) In order to construct AuxChannel, we further propose a new cryptographic primitive called **Consecutive Verifiably Encrypted Signature (CVES)** (as known as Consecutive Adaptor Signature). CVES allows a signer to encrypt his signatures by using a sequence of his encryption keys. These encrypted signatures are verifiable, and more importantly the encryption-decryption key-pairs are consecutive, which means the latest key-pairs are derived from their previous session (consecutive). A verifier can ensure both the *correctness* of the encrypted signature and the *consecutiveness* of signer's encryption keys. The key generation function is one-way, meaning that the previous key-pairs can be used to generate the next one but not the opposite flow. (We remark that CVES can be also regarded as the *consecutive* version of adaptor signature [18], in which a new "adapted secret" can be generated in a verifiable and consecutive way from its ancestor.) CVES is not just the core building block of AuxChannel but we believe it is also of independent interest for further research.

2 RELATED WORK

This work focuses on the payment channel scheme, and introduces a new bi-directional payment channel protocol for scriptless blockchains. We then summarize some works related to payment channel protocols below.

Existing payment channel protocols for scriptless blockchain. While providing bi-directional payment channels on scriptless blockchains remains unsolved, ways to provide *unidirectional* payment channel for scriptless blockchain have been explored.

Currently there are three attempts to provide *unidirectional* payment channel for scriptless blockchain, namely the use of time proof (e.g. DLSAG [28] and Z-Channel [44]) and timed commitment (e.g. PayMo [38]), as all of them provide the time-lock function without requiring the support of script language.

Both DLSAG channel for Monero and Z-Channel for Zerocash deploy time proof, which is a verifiable commitment as a predefined timelock. It is contained in the input of a transaction. The output of the transaction can only be spent after the predefined time proof. To avoid repetition, we use DLSAG channel as an example to specify how the time commitment works in a payment channel.

DLSAG channel makes it possible for Monero to support payment channels, while it has the limited lifespan which means a channel has to be closed within a predefined time. This limits the number of transactions that can be processed in the channel. Further, it only supports one-way payment. This may exhaust the balance in the channel quickly (even before the predefined channel lifespan) and the channel parties will have to close and re-establish a channel between them. Also, this solution requires an update on the mining software, which means a hard fork in the blockchain is needed to support time-proof verification. Thus it makes it difficult for being adopted in Monero. Even for that, it is only designed specifically for Monero, but not other scriptless blockchains.

Another approach is timed commitment. PayMo uses timed commitment to provide payment channel for Monero users. A user creates a timed commitment transaction such that the receiver of the commitment can force the opening of the commitment and learn the signature only after a pre-specified time. PayMo has extra computation requirement for opening the timed commitment, while it can be mitigated by a third party service [36]. Thus PayMo is fully compatible with the transaction in Monero. However, it still has limited lifespan and supports one-way payment only.

Existing bi-directional payment channel protocols for blockchains with limited scripts. Bolt [20], a bi-directional payment channel protocol, focuses on solving the linkability issues in channels and can be built on ZCash. Deploying Bolt requires that the zero-knowledge proof can be verified on the underlying blockchain, which is not compatible with other scriptless blockchains, such as Monero. A recent work from Aumayr et al. [2] formalizes the generalized payment channel for Blockchains with limited scripts, which requires the underlying blockchain to support adaptor signature scheme, relative time-lock and constant number of Boolean operations. Another recent work, Sleepy Channels [4], supports bi-directional payment channel with limited life-span, and also have the requirement of supporting time-lock on-chain. In addition, running Sleepy Channels between untrusted parties requires extra collateral, which could be equal to the channel capacity for each channel party, to guaranteed the secure and fast channel closure. While our work focuses on building bi-directional payment channels with unlimited lifespan for scripltess blockchains, such that building payment channels only requires the underlying blockchain to support verifiably encrypted signature (aka adaptor signature).

Other related works. In 2013, Satoshi Nakamoto, the author and creator of Bitcoin whitepaper and project, described a *high-frequency transactions technique* in a personal email [32]. Participants can close the channel with any intermediate state and double-spend the off-chain funding input. According to its design, this technique only works on a strong assumption that all participants follow the protocol honestly. This idea is considered as the prototype of payment channel schemes. This is the first off-chain transaction idea on blockchain with obvious disadvantages, which has no punishment and on-chain deposit designed. At the same year, Spilman-styled payment channel [34] was proposed to fill the insecure design in Satoshi's *high-frequency transactions technique* idea. This protocol makes a secure two-party off-chain trades without the honest players assumption. While this protocol only allows the uni-directional coin transfer with limited channel lifetime, this design increases the cost of creating a payment channel and is not desirable in real payment scenarios. *Duplex Channel* [40] is a simple attempt of bi-directional payment channel, which is consisted of two opposite directional Spilman-styled payment channels between two channel parties. This is not only quite straightforward, but also inherits the design of limited channel lifetime. While the direction of designing a bi-direction payment channel protocol is worth to do a depth-in research. In 2016, Poon-Dryja channel [30] realizes untrusted bi-directional payment channel without limited lifetime on Bitcoin. The well-known payment channel network project, Lightning Network, is implemented based on the Poon-Dryja channel protocol. The core technique of Poon-Dryja channel, Hash-Time Lock Contract (HTLC), is widely adopted in many protocols. HTLC is a useful technique among on-chain, off-chain and cross-chain applications, and requires a programmable language supporting by its underlying blockchain. After Poon-Dryja channel, there are two worth mentioned payment channel protocols, *Eltoo* [15] and *Generalized Bitcoin-Compatible Channels* [2]. These two protocols are also based on the HTLC technique.

Atomic swap is one of the cross-chain coin swap protocols, which also adopts HTLC technique as its core building block. According to its requirements of programmable language, atomic swap has the same issues for privacy-preserving blockchains. BTC-XMR atomic swaps [22] protocol enables the coins transfer between a script-enabled blockchain and a scriptless blockchain. It requires time-lock function only on Bitcoin side. Another recent cross-chain coins swap protocol for scriptless blockchain, JugglingSwap [33], introduces a trusted third party to control the time of secret releasing. Comparing to this work, AuxChannel does not require the involvement of the trusted third party for each off-chain payment by using the CVES scheme.

The anonymous multi-hop locks (AMHLs) [24], which aims to lock several atomic transactions among several users in a payment path, is similar to the our proposed CVES. To derive all the decryption keys, AMHLs requires all the on-path users' participation. Also, the number of decryption keys is fixed at the setup phase. Thus, under the bi-directional payment channel scenario, AMHLs cannot prevent a malicious participant from locking the channel and also suffers from a limited life cycle. The functionality of AMHLs dose not satisfy the requirements of AuxChannel, while CVES does.

3 PRELIMINARY

This section gives the definitions of Verifiable Encrypted Signature and One-time Verifiable Encrypted Signature, which are the two preceding concepts of our new primitive Consecutive Verifiable Encrypted Signature.

3.1 Verifiably Encrypted Signature

Verifiably encrypted signature (VES) [9, 13] scheme allows a signer's signature to be encrypted using a third party's encryption key (the third party is also known as adjudicator), and the resulting ciphertext can be publicly verified that it is really an encryption of the signer's signature.

Note that VES is different from *signcryption* (sign-and-encrypt), though they share a similar name. In VES scheme, a signature is encrypted under the signer's encryption key or a third party's public key. We refer the third party as the adjudicator, which follows the term adopted in Boneh's paper [9]. The encrypted signature can be verified by others, and the corresponding signature can only be revealed by the signer or the trusted third party. A similar cryptographic primitive is signcryption, where a signer encrypts its signature under the receiver's public key. However, differ from VES scheme, only the receiver can decrypt and verify this encrypted signature.

An original VES scheme is defined with an ordinary underlying signature scheme (Gen, Sign, Verify) and four additional algorithms: EncGen is an algorithm to generate a (public key encryption) key pair for the adjudicator; EncSign is an algorithm for generating an encrypted signature from a message, the encryption key of the adjudicator and the signing key of the signer; EncVerify is an algorithm for convincing the verifier that the encrypted signature of a message is valid by using the verification key of the signer and the encryption key of the adjudicator; DecSig is an algorithm for recovering the encrypted signature by using the decryption key of the adjudicator. The formal definition of the verifiably encrypted signature scheme is given as follows.

Definition 1. (Verifiably Encrypted Signature Scheme). A verifiably encrypted signature (VES) scheme is defined with an ordinary underlying signature scheme (Gen, Sign, Verify) and four additional algorithms EncGen, EncSign, EncVerify, and DecSig:

- $\mathsf{Gen}(\lambda) \rightarrow (sk, pk)$: a probabilistic key generation algorithm which takes the input security parameter λ and outputs a key pair (sk, pk), where sk is the signing key and pk is its corresponding verification key.
- $\mathsf{Sign}(sk, m) \rightarrow \sigma$: a probabilistic signing algorithm that, given a signing key sk and a message m, produces a signature σ.
- $\mathsf{Verify}(pk, \sigma, m) \rightarrow \{0, 1\}$: a deterministic verification algorithm that given a verification key pk, a signature σ and a message m, and returns 1 or 0.
- $\mathsf{EncGen}(\lambda) \rightarrow (dk, ek)$: a probabilistic encryption key generation algorithm for adjudicator with an input, the security parameter λ, and outputs a valid key-pair (dk, ek), such that dk is the decryption key and ek is the corresponding encryption key.
- $\mathsf{EncSign}(sk, ek, m) \rightarrow \hat{\sigma}$: a probabilistic encrypted signing algorithm, which on the input of the signer's signing key sk, adjudicator's encryption key ek and a message m, outputs an encrypted signature $\hat{\sigma}$.
- $\mathsf{EncVerify}(pk, ek, \hat{\sigma}, m) \rightarrow \{0, 1\}$: a deterministic verification algorithm that given the signer's verification key pk, adjudicator's encryption key ek, an encrypted signature $\hat{\sigma}$ and a message m, returns 1 or 0.

- DecSig$(dk, \hat{\sigma}) \to \{\sigma, \perp\}$: a deterministic signature decryption algorithm that given the ajudicator's decryption key dk and an encrypted signature $\hat{\sigma}$, outputs a decrypted signature σ or $\perp$. This process is also know as adjudication.

VES should satisfy three security properties: validity, unforgeability, and opacity, which are described as follows:

- **Validity**: It requires that for any message, if the encrypted signature is generated by the EncSign, it should pass through the verification of EncVerify. It also requires that for any message, if the encrypted signature generated by the EncSign is decrypted by DecSig, the resulting signature should pass through the verification of Verify.
- **Unforgeability**: It is hard to forge a valid encrypted signature for any message from the verification key and encryption key only.
- **Opacity**: The signature for a message cannot be recovered from an encrypted signature without the decryption key.

3.2 One-Time Verifiable Encrypted Signature

The one-time verifiably encrypted signature (one-time VES), as known as *Adapator Signature*, is formalized by Fournier [18]. The main difference for one-time VES over normal VES is that the decryption key of one-time VES scheme can be recovered once the original signature is revealed. Therefore the encryption-decryption key pairs can be used once (and thus called “one-time" VES). We also remark that in many context, one-time VES is more known as adaptor signature (e.g. [3, 17, 31, 35]). Although the name of the signature (and the names of the algorithms) are different, they are exactly the same. We use the notion of one-time VES in the rest of this paper, as this name reflects more the nature of its properties in our context.

One-time VES scheme is defined under the basis of the original VES scheme by adding two algorithms: RecKey and Rec. RecKey is an algorithm for extracting the “recovery” key from the encrypted signature, which is later used in Rec to recover the decryption key from the decrypted signature. Here, we only give the two additional algorithms, while referring readers to Section 3.1 for the rest of the other 7 algorithms which are exactly the same as VES:

Definition 2. (One-time Verifiably Encrypted Signature). A one-time verifiably encrypted signature (one-time VES) scheme is a VES scheme (Gen, Sign, Verify, EncGen, EncSign, EncVerify, DecSig) with two additional algorithms RecKey and Rec:

- RecKey$(ek, \hat{\sigma}) \to rk$: a deterministic recovery key extraction algorithm, which given the encryption key ek and an encrypted signature $\hat{\sigma}$ returns a recovery key rk.
- Rec$(\sigma, rk) \to dk$: a deterministic decryption key recovery algorithm, which given a signature σ and the recovery key rk returns the decryption key dk.

The one-time VES scheme has the security properties, *validity*, *unforgeability* and *recoverability*. The first two are the same as VES and we omit here. Informally speaking, *recoverability* requires that it is easy to recover a decryption key by knowing the encrypted signature and its original signature.

4 CONSECUTIVE VERIFIABLY ENCRYPTED SIGNATURE (A.K.A. CONSECUTIVE ADAPTOR SIGNATURE)

4.1 Introduction of CVES

We propose a new cryptographic primitive, called *Consecutive Verifiably Encrypted Signature* (CVES) (a.k.a *Consecutive Adaptor Signature*), which is a generalization of one-time VES such that CVES enables a sequence of one-time encryption-decryption key pairs. Given the initial decryption key, CVES is able to generate the next encryption-decryption key pair and so on. But this process cannot be reverted. That is, given the most updated decryption key only, no one can compute its “ancestors”. More importantly, CVES allows anyone to publicly verify that the new encryption key is correctly generated by its immediate ancestor, from the proof given by the owner of the corresponding decryption key.

CVES inherits all the algorithms defined in one-time VES and introduces three additional algorithms: KeyUpdate generates a new encryption-decryption key-pair by taking the previous decryption key; ConsProof generates a proof from the knowledge of two consecutive decryption keys which can convince any verifier that the corresponding encryption keys are consecutive (that is, the new encryption key is well-formed or generated from KeyUpdate); ConsVerify verifies the proof generated by ConsProof.

We now present the formal definition of CVES scheme. We use (dk^0, ek^0) to denote the initiate decryption-encryption key pair, while (dk^i, ek^i) to denote the i-th key pair, after executing KeyUpdate i times for any $i > 0$. We also use KeyUpdate$^{\mathrm{i}}$ to denote recursively executing KeyUpdate i times.

Definition 3 (Consecutive Verifiably Encrypted Signature)**.** A consecutive verifiably encrypted signature (CVES) scheme is a one-time VES scheme (Gen, Sign, Verify, EncGen, EncSign, EncVerify, DecSig, RecKey, Rec) with three additional algorithms KeyUpdate, ConsProof and ConsVerify:

- KeyUpdate$(dk^{i-1}) \to (dk^i, \hat{dk}^i, ek^i)$: a deterministic algorithm with an input, a decryption key dk^{i-1} and returns its successive decryption key dk^i, a truncated decryption key $\hat{dk}^i$ and the corresponding encryption key ek^i.
- ConsProof$(dk^{i-1}, dk^i) \to P^i$: a probabilistic algorithm, which on inputs two decryption keys dk^{i-1} and dk^i, outputs a proof P^i.
- ConsVerify$(ek^{i-1}, ek^i, P^i) \to \{0, 1\}$: a deterministic algorithm, which on inputs two consecutive encryption keys ek^{i-1}, ek^i and a proof of consecutiveness P^i, outputs 0 or 1.

We also note that there are some minor differences with one-time VES for the other algorithms. In CVES, we only require Rec to output the truncated decryption key $\hat{dk}^i$ instead of the full decryption key dk^i, as in one-time VES. Furthermore, EncGen is executed only once to generate the initial decryption-encryption key-pair $((dk^0, \hat{dk}^0), ek^0)$ including the truncated decryption key $\hat{dk}^0$, and the subsequent decryption-encryption key-pairs can be generated by executing KeyUpdate.

4.2 Security of CVES

CVES scheme inherits all the secure properties from one-time VES scheme, i.e., *validity*, *unforgeability*, *recoverability*, along with three additional properties, *consecutiveness*, *consecutive verifiablity* and *one-wayness*. The informal definitions of the first three have been given in the previous section, and the later three definitions are described informally as follows:

- **Consecutiveness**: It requires that the decryption-encryption key-pair used in the i-th session of CVES is derived from the decryption key of the $(i-1)$-th session from KeyUpdate, where $i > 0$.
- **Consecutive verifiablity**: Given two encryption keys and the corresponding proof, anyone can be convinced if the two keys are consecutive, meaning that the new encryption key is generated from KeyUpdate taking the input of the previous corresponding decryption key.
- **One-wayness**: given the i-th decryption key, no one can derive any of the j-th decryption key, where $0 \leq j < i$.

We then give the formal definition of CVES security properties. We first include the concept of *validity*, *recoverability* and *consecutiveness* into a single definition of *correctness*, as defined below:

Definition 4 (Correctness of CVES)**.** A CVES is correct if the following properties hold. For any $i > 0$ and any message m^i, we require

$$
\begin{aligned}
(sk, pk) &\leftarrow \mathsf{Gen}(\lambda) \\
(dk^0, \hat{dk}^0, ek^0) &\leftarrow \mathsf{EncGen}(\lambda) \\
(dk^{i-1}, \hat{dk}^{i-1}, ek^{i-1}) &\leftarrow \mathsf{KeyUpdate}^{i-1}(dk^0) \\
(dk^i, \hat{dk}^i, ek^i) &\leftarrow \mathsf{KeyUpdate}(dk^{i-1}) \\
\sigma^i &\leftarrow \mathsf{Sign}(sk, m^i) \\
\hat{\sigma}^i &\leftarrow \mathsf{EncSign}(sk, ek^i, m^i) \\
P_i &\leftarrow \mathsf{ConsProof}(dk^{i-1}, dk^i) \\
1 &\leftarrow \mathsf{ConsVerify}(ek^{i-1}, ek^i, P^i) \\
1 &\leftarrow \mathsf{EncVerify}(pk, ek^i, \hat{\sigma}^i, m^i) \\
\hat{dk}^i &\leftarrow \mathsf{Rec}(\sigma^i, \mathsf{RecKey}(\hat{\sigma}^i)) \\
1 &\leftarrow \mathsf{Verify}(pk, \mathsf{DecSig}(\hat{dk}^i, \hat{\sigma}^i), m^i)
\end{aligned}
\tag{1}
$$

Next, before giving the definition of unforgeability, we first introduce two oracles below, which can be accessed by adversaries in the *unforgeability* experiment.

Signing Oracle: Let O be a signing oracle, which takes the verification key pk and the message m as inputs, and outputs a signature σ such that $1 \leftarrow \mathsf{Verify}(pk, \sigma, m)$.

Encsigning Oracle: Let $\mathcal{E}$ be an encsigning oracle which takes the verification key pk, the encryption key ek^i for session i and a message m as inputs, and outputs an encrypted signature $\hat{\sigma}^i$ such that $1 \leftarrow \mathsf{EncVerify}(pk, ek^i, \hat{\sigma}^i, m)$.

Definition 5 (Unforgeability of CVES)**.** A CVES is *unforgeable*, if the advantage $Adv = Pr[Exp_{\mathcal{A}}^{\text{unforgeability}} = 1]$, where $Exp_{\mathcal{A}}^{\text{unforgeability}}$ is defined in Figure 1, of any PPT adversary in forging a signature σ^i given access to signing oracle O and encsigning oracle $\mathcal{E}$ is negligible. The probability is taken over the coin tosses of the key generation algorithm Gen and EncGen, the oracles O and $\mathcal{E}$. The adversary is additionally constrained that $m^* \neq m$, where m is an input to $\mathcal{E}$ and O accessed by the adversary.

Definition 6 (One-wayness of CVES)**.** A CVES is *one-way*, if the advantage $Adv = Pr[Exp_{\mathcal{A}}^{\text{one-way}} = 1]$, where $Exp_{\mathcal{A}}^{\text{one-way}}$ is defined in Figure 1, of any PPT adversary in recovering dk^{i-1} from dk^i is negligible. The probability is taken over the coin tosses to the reversion algorithm of the adversary.

Definition 7 (Consecutive Verifiability)**.** A CVES is *consecutive verifiable*, if the advantage $Adv = Pr[Exp_{\mathcal{A}}^{\text{cverifiability}} = 1]$, where $Exp_{\mathcal{A}}^{\text{cverifiability}}$ is defined in Figure 1, of any PPT adversary in yielding a proof of consecutiveness on two unconsecutive decryption keys is negligible. The probability is taken over the coin tosses of the ConsProof algorithm and of the adversary.

The *opacity* property defined in VES guarantees that a decrypted signature is unforgeable. Similar to one-time VES, CVES does not have the *opacity* property as well, as the *recoverability* property makes it possible to recover the i-th session decryption key once the j-th session decryption key is revealed, where $i > j \geq 0$.

A CVES is secure, if it satisfies the correctness, unforgeabiltiy, one-wayness, consecutive verifiability defined in Definition 4, 5,6,7.

4.3 Schnorr CVES Construction

Overview Idea: Now we present a Schnorr-based CVES construction. The overview idea is like that. Our construction is motivated by the Schnorr-based one-time VES of [18]. The main difference is the consecutiveness of our construction, which requires two additional elements to achieve our purpose: 1) The design of a one-way function is to derive the next decryption key from the previous one; 2) The proof to prove the correctness of the key generated, such that the verification only takes the public information (encryption keys).

Since our system is based on the Schnorr ecosystem, which is in discrete logarithm (DL) setting, the first naive idea of constructing such a one-way function and the corresponding proof, is to use another DL proof system. Exponentiation function over a cyclic group is already a popular one-way function, while it also provides an efficient zero-knowledge proof for its input given the output. Unfortunately, in our case it cannot be adopted in an efficient way. The main reason is the difference of the domains of the input and output of the exponentiation function. The input domain is usually an integer, while the output domain is a group element (which is not necessarily to be an integer). In our setting, we require the domain of the input and output for the one-way function to be the same. This is necessary because the one-way function is used to generate the decryption key of the next session (from the decryption key of the previous session), which obviously should lie within the same domain of the previous decryption key. Although there are some techniques to convert the output of the exponentiation function to the same domain of the input element, they are not efficient when a proof is also required.

To be exact, we need a one-way *permutation* with the ability to construct an efficient proof for our scheme. Instead of using an exponentiation function, we in turn use a square function over

```
**Exp** $Exp_{\mathcal{A}}^{\text{unforgeability}}(\lambda)$ :
$(sk, pk) \leftarrow \mathsf{Gen}(\lambda)$
$(dk^0, \hat{dk}^0, ek^0) \leftarrow \mathsf{EncGen}(\lambda)$
$(i, st) \leftarrow \mathcal{A}_0(pk, ek^0)$
$(dk^{i-1}, \hat{dk}^{i-1}, ek^{i-1}) \leftarrow \mathsf{KeyUpdate}^{i-1}(dk^0)$
$(dk^i, \hat{dk}^i, ek^i) \leftarrow \mathsf{KeyUpdate}(dk^{i-1})$
$P^i \leftarrow \mathsf{ConsProof}(dk^{i-1}, dk^i)$
$m^* \leftarrow \mathcal{A}_1^{O(\cdot),\mathcal{E}(\cdot)}(st, pk, ek^i)$
$\hat{\sigma}^{i*} \leftarrow \mathcal{E}(pk, ek^i, m^*)$
$\sigma^{i*} \leftarrow \mathcal{A}_2(st, pk, ek^{i-1}, ek^i, P^i, \hat{\sigma}^{i*})$
If Verify$(pk, \sigma^{i*}, m^*) = 1$:
        Return 1
Else Return 0
```

```
**Exp** $Exp_{\mathcal{A}}^{\text{one-way}}(\lambda)$ :
$(sk, pk) \leftarrow \mathsf{Gen}(\lambda)$
$(dk^0, \hat{dk}^0, ek^0) \leftarrow \mathsf{EncGen}(\lambda)$
$(i, st) \leftarrow \mathcal{A}_0(pk, ek^0)$
$(dk^i, \hat{dk}^i, ek^i) \leftarrow \mathsf{KeyUpdate}^i(dk^0)$
$(dk^{i-1^*}, m^*) \leftarrow \mathcal{A}_1(st, dk^i)$
$(dk^{i^*}, \hat{dk}^{i^*}, ek^{i^*}) \leftarrow \mathsf{KeyUpdate}(dk^{i-1^*})$
$\hat{\sigma}^i \leftarrow \mathsf{EncSign}(sk, ek^i, m^*)$
$\sigma^{i*} \leftarrow \mathsf{DecSig}(\hat{dk}^{i^*}, \hat{\sigma}^i)$
If Verify$(pk, \sigma^{i*}, m^*) = 1$:
        Return 1
Else Return 0
```

```
**Exp** $Exp_{\mathcal{A}}^{\text{cverifiability}}(\lambda)$ :
$(sk, pk) \leftarrow \mathsf{Gen}(\lambda)$
$(dk^0, \hat{dk}^0, ek^0) \leftarrow \mathsf{EncGen}(\lambda)$
$(i, st) \leftarrow \mathcal{A}_0(pk, ek^0)$
$(dk^i, \hat{dk}^i, ek^i) \leftarrow \mathsf{KeyUpdate}^i(dk^0)$
$(dk^{i+1^*}, \hat{dk}^{i+1*}, ek^{i+1^*}, m^*, P^*) \leftarrow \mathcal{A}_1(st, dk^i)$
$\hat{\sigma}^* \leftarrow \mathsf{EncSign}(sk, ek^{i+1^*}, m^*)$
$(dk^{i+1}, \hat{dk}^{i+1}, ek^{i+1}) \leftarrow \mathsf{KeyUpdate}(dk^i)$
$\sigma \leftarrow \mathsf{DecSign}(\hat{dk}^{i+1}, \hat{\sigma}^*)$
If ConsVerify$(ek^i, ek^{i+1^*}, P^*) = 1$ &&
   Verify$(pk, \sigma, m^*) = 0$
      Return 1
Else Return 0
```

Figure 1: Experiments used to define unforgeability, one-wayness, and consecutive verifiability properties of CVES.

a modulus of a composite number n. If the factorization of n is unknown, the square function is one-way. It is also nice that the domain of both input and output is $\mathbb{Z}_n$.

The next and most challenging step is to construct an efficient zero-knowledge proof over the square function modulus n, such that *both the input and output of the function are the secret to be proven.* We resolve this issue by using another group of unknown order (modulus another composite integer N without knowing its factorization).

The last step is to link all these groups together. Currently we have three groups: the Schnorr prime order group $\mathbb{Z}_p$ (for a cyclic group with prime order p), the one-way function group $\mathbb{Z}_n$ and another group $\mathbb{Z}_N$ with unknown group order which is used in the zero-knowledge proof. We link these together in our proof used in ConsProof.

Concrete Construction: Our scheme is constructed over a cyclic group $\mathbb{G}_p$ with prime order p. Let $N = p'q'$, where p', q' are primes with length λ_1 which is a security parameter. It can be generated in a trusted manner (e.g. using MPC such that no one knows the factorization of N). $\mathbb{Z}_N^*$ is thus a group with unknown order $\phi(N)$. Let g, h be the group generators of $\mathbb{G}_p$ and $\mathbb{Z}_N^*$ respectively, and $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ be a hash function. Suppose that it is hard to solve a discrete logarithm problem in the group $\mathbb{Z}_N^*$.

We now construct the one-way function for KeyUpdate. We use the following settings: Let $n = p''q''$ for some primes p'', q'' with length λ_2 which is another security parameter. The order of the group $\mathbb{Z}_n^*$ is with unknown order $\phi(n)$ (again we can use MPC to generate n such that no one knows its factorization). The one-way function can be set as $x \mapsto x^2 \bmod n$.

The complete construction of Schnorr-based CVES scheme is shown in Figure 2. (The detailed explanation and underlying instantiation of the NIZK in the algorithms ConsVerify and ConsProof is given in Appendix B.)

We also remark that N can be used as a system parameter (e.g. used by a group of users) while n is used between the prover and the verifier. In practice, we simply set $N = n$, if this number is generated using MPC or other trusted method so that no one knows its factorization. We then give the intuition of why we can set $N = n$:

According to the algorithm KeyUpdate in Figure 2, the public key used in zero-knowledge proof is constructed as $\tilde{Y}' = h^{\tilde{y}'} \bmod N$, where $\tilde{y}' = \tilde{y}^2 \bmod n$. Thus, the distribution of $\tilde{y}'$, where $\tilde{y}' \in [0, n)$, would affect the distribution of $\tilde{Y}'$. Ideally, we hope $\tilde{Y}'$ is uniformly distributed from $\mathbb{Z}_N^*$, which can be guaranteed when $n = \phi(N)$, where $\phi(N)$ is the group order of $\mathbb{Z}_N^*$. As $\phi(N)$ is unknown, to ensure that $(h^{\tilde{y}'} \bmod N)$ can cover all group elements in $\mathbb{Z}_N^*$, n should be greater than $\phi(N)$. Thus if we set $n = N > \phi(N)$, and when $\tilde{y}'$ is from $[0, n)$, and $n = N$ is large enough, $\tilde{Y}' = (h^{\tilde{y}'} \bmod N)$ is statistically indistinguishable from $\tilde{Y}'^*$, which is uniformly in $\mathbb{Z}_N^*$.

Theorem 1. *The Schnorr CVES is correct, and satisfies the unforgeability, one-wayness, and consecutive verifiability, if the DL problem and factorization problem are hard, and the underlying NIZK system is correct and sound.*

The proof is given in Appendix A.

4.4 Performance Evaluation of Schnorr CVES

We have instantiated and implemented the algorithms KeyUpdate(·), ConsVerify(·) and ConsProof(·) (in Figure 2). The detailed explanation and underlying instantiation of the NIZK in the algorithms ConsVerify and ConsProof are given in Appendix B.

Our implementation is based on a zero-knowledge library, emmy [41]. For simplicity, we set $n = N = p'q'$, and adopt an unknown group $\mathbb{Z}_N^*$ and a cyclic group $\mathbb{G}_p$ with the prime order p, where $|N| = 2048$ and $p = 2^{252} + 27742317777372353535851937790883648493$.

Suppose a signer has generated an encryption-decryption keypair (ek^{i-1}, dk^{i-1}) and shared ek^{i-1} with a verifier, where $i \in \mathbb{N}^+$. He then performs $(dk^i, \hat{dk}^i, ek^i) \leftarrow \mathsf{KeyUpdate}(dk^{i-1})$ to updates the encryption and decryption keys. To convince the verifier that dk^{i-1} and dk^i are consecutive, the signer creates a proof $P^i \leftarrow \mathsf{ConsProof}(dk^{i-1}, dk^i)$, and shares P^i and ek^i with the verifier. Our experiments are run on a macOS with processor 2.6 GHz 6-Core Intel Core i7 and memory 16GB 2400 MHz DDR4. Each algorithm is executed for 100 times. The result shows that on average it requires about 33 ms to create the proof, and the corresponding verification

```
$\mathsf{Gen}(\lambda)$
$x \leftarrow \mathbb{Z}_p; X := g^x;$
$sk := (x, X);$
$pk := X;$
$\textbf{return } (sk, pk)$

$\mathsf{EncGen}(\lambda)$
$\tilde{y} \leftarrow \mathbb{Z}_n;$
$y := \tilde{y} \bmod p;$
$Y := g^y;$
$\tilde{Y} := h^{\tilde{y}} \bmod N;$
$dk^0 := (\tilde{y}, y, Y);$
$\hat{dk}^0 := (y, Y)$
$ek^0 := (\tilde{Y}, Y)$
$\textbf{return } (dk^0, \hat{dk}^0, ek^0)$

$\mathsf{DecSign}(\hat{dk}^i, \hat{\sigma}^i)$
$(\hat{R}, \hat{s}) := \hat{\sigma}^i$
$(y, Y) := \hat{dk}^i$
$R := \hat{R}Y$
$s := (\hat{s} + y) \bmod p$
$\textbf{return } \sigma^i := (R, s)$

$\mathsf{KeyUpdate}(dk^{i-1})$
$(\tilde{y}, y, Y) := dk^{i-1};$
$\tilde{y}' := \tilde{y}^2 \bmod n;$
$y' = \tilde{y}' \bmod p;$
$\tilde{Y}' := h^{\tilde{y}'} \bmod N;$
$Y' := g^{y'};$
$dk^i := (\tilde{y}', y', Y');$
$ek^i := (\tilde{Y}', Y');$
$\hat{dk}^i := (y', Y');$
$\textbf{return } (dk^i, \hat{dk}^i, ek^i)$

$\mathsf{Sign}(sk, m^i)$
$(x, X) := sk;$
$r \leftarrow \mathbb{Z}_p; R := g^r;$
$c := H(m^i||R||X);$
$s := (r + cx) \bmod p;$
$\textbf{return } \sigma^i := (R, s)$

$\mathsf{EncSign}(sk, ek^i, m^i)$
$(x, X) := sk;$
$(\tilde{Y}, Y) := ek^i$
$\hat{r} \leftarrow \mathbb{Z}_p; \hat{R} := g^{\hat{r}}$
$R := \hat{R}Y$
$c := H(m^i||R||X)$
$\hat{s} := (\hat{r} + cx) \bmod p$
$\textbf{return } \hat{\sigma}^i := (\hat{R}, \hat{s})$

$\mathsf{RecKey}(\hat{\sigma}^i)$
$(\hat{R}, \hat{s}) := \hat{\sigma}^i$
$\textbf{return } rk^i := \hat{s}$

$\mathsf{ConsProof}(dk^{i-1}, dk^i)$
$(\tilde{y}, y, Y) := dk^{i-1};$
$(\tilde{y}', y', Y') := dk^i;$
$Y := g^y; Y' := g^{y'};$
$P^i \leftarrow PoK((\tilde{y}, \tilde{y}', y, y', k):$
    $\{Y = g^y \wedge Y' = g^{y'} \wedge$
    $\tilde{Y} = h^{\tilde{y}} \bmod N \wedge$
    $\tilde{Y}' = h^{\tilde{y}'} \bmod N \wedge$
    $\tilde{Y}'(h^n)^k = \tilde{Y}^{\tilde{y}} \bmod N \wedge$
    $Y = g^{\tilde{y}} \wedge Y' = g^{\tilde{y}'} \wedge$
    $0 < \tilde{y}' < n\})$
$\textbf{return } P^i$

$\mathsf{Verify}(pk, \sigma^i, m^i)$
$X := pk;$
$(R, s) := \sigma^i;$
$c := H(m^i||R||X);$
$\textbf{return } R \overset{?}{:=} g^s X^{-c}$

$\mathsf{EncVerify}(pk, ek^i, \hat{\sigma}^i, m^i)$
$X := pk; (\tilde{Y}, Y) := ek^i$
$(\hat{R}, \hat{s}) := \hat{\sigma}^i$
$R := \hat{R}Y$
$c := H(m^i||R||X)$
$\textbf{return } \hat{R} \overset{?}{=} g^{\hat{s}} X^{-c}$

$\mathsf{Rec}(\sigma^i, rk^i)$
$(R, s) := \sigma^i$
$\hat{s} := rk^i$
$y := (s - \hat{s}) \bmod p$
$Y := g^y$
$\textbf{return } \hat{dk}^i := (y, Y)$

$\mathsf{ConsVerify}(ek^{i-1}, ek^i, P^i)$
$(\tilde{Y}, Y) := ek^{i-1};$
$(\tilde{Y}', Y') := ek^i;$
$\chi := (\tilde{Y}, \tilde{Y}', Y, Y');$
$\textbf{return } \{0, 1\} \leftarrow \mathsf{NIZK.Verify}(\chi, P^i)$
```

Figure 2: The construction of Schonrr-based CVES algorithms.

requires about 348 ms. The communication latency is transferring 16.9 KB data, including the required proof and encryption key (P^i, ek^i).

5 OVERVIEW OF AUXCHANNEL

We propose AuxChannel, a bi-directional payment channel protocol with unlimited life span for scriptless blockchains. We present the basic idea of AuxChannel with a step by step approach, before presenting it formally in Section 6.

5.1 Problem Statement

Recall the scenario we considered: two parties, Alice and Bob, establish a bi-directional payment channel over a scriptless blockchain. Initially, each of them deposits 5 coins in the channel. After several sessions of payments, Alice has 10 coins and Bob exhausted his balance. To close the channel, the final state (Alice has 10 coins and Bob has no coin) should be posted on the blockchain.

Depending on the rule of closing a channel, two misbehaviours could happen: (1) if closing a channel requires the permission from both parties, Bob can ignore the request as he loses nothing but Alice would lose all 10 coins; and (2) if it only requires a single side permission, Bob can close the channel by using a revoked state, where Bob can claim more than his actual balance. To protect a channel from being attacked by the two misbehaviours above, the protocol should provide two guarantees, *guaranteed channel closing* and *guaranteed balance payout* for channel parties [16].

Please note that the above challenge does not exist on uni-directional channels, where a payer makes incremental payment to a payee: allowing the payee to close a channel alone is sufficient as the payee is incentivised to always close the channel (i.e., *guaranteed channel closing*) with the latest actual balance (i.e., *guaranteed balance payout*).

As in other layer 2 solutions supporting bi-directional payments (such as Lightning Network [30]), the key to address the above challenges is to provide the ability to resolve dispute upon observing the misbehaviours. Resolving disputes on the blockchain is easy for blockchains supporting script language, but extremely challenging, if not impossible, for scriptless blockchains.

5.2 Strawman Design

One natural step is to leverage a script-enable blockchain (denoted $\mathcal{L}_2$) to help the scriptless blockchain (denoted $\mathcal{L}_1$) to resolve any dispute.

First attempt. This leads to the first attempt of our design, where both channel parties lock enough funds on the script-enabled blockchain as "insurance". Upon the detection of any misbehavior, the victim can resolve the dispute by claiming all insurance on $\mathcal{L}_2$. As this only provides compensate to the victim on $\mathcal{L}_2$ rather than correcting the misbehavior on $\mathcal{L}_1$, the victim will loss its fund on $\mathcal{L}_1$ (regardless of requiring permissions from one or both parties to close a channel) and the malicious attacker still gets the money on $\mathcal{L}_1$ (if closing a channel only requires the permission from one party). Thus, to deincentivise such misbehaviors, each party should lock a fund as an insurance on $\mathcal{L}_2$, such that the locked amount is no smaller than the channel capacity on $\mathcal{L}_1$ to cover the maximum potential loss of a victim.

This approach has two issues. First, it requires (the smart contract of) $\mathcal{L}_2$ to monitor all events on $\mathcal{L}_1$ and retrieve evidence to resolve the dispute. While there are existing attempts to enable communication across blockchains [1], they introduce significant additional cost and with open challenges [42]. Second, it requires additional deposit on $\mathcal{L}_2$ as insurance, where the total deposit is at least twice as large as the channel capacity. In other words, channel parties have to lock $3C$ to establish a channel with capacity C, which greatly reduces the liquidity and makes it impractical.

Second attempt. In our second attempt, rather than locking the deposit for covering the potential loss and deincentivise misbehaviors, we require both parties' involvement to close a channel. In this way, closing a channel requires the permission from both parties, eliminating the possibility of claiming balances with a revoked state and providing guaranteed balance payout. In addition, we envision a distributed and verifiable "key escrow" service on $\mathcal{L}_2$ to keep and release some secret (for example, by using [6]) required to close the channel.

In particular, each payment in the channel is created through a verifiable encrypted signature scheme, where the "key escrow" service is leveraged to keep a secret to recover the full signature for validating the payment. In the normal case, when both parties are honest, they can cooperate to close the channel with the latest state. In the case where one party is not responding to the request of closing a channel, the "key escrow" service can help releasing the secret to close the channel with latest channel state. Note that each encrypted signature should use a unique encryption key, as otherwise when an honest party releases the decryption key, the malicious counterparty can decrypt and validate any past revoked payment.

This approach still has some issues. While both parties only need to pay a small fee to the distributed and verifiable key escrow service (rather than locking additional significant amount of insurance), channel parties need to interact with the key escrow service for each payment. This not only incurs $O(n)$ communication complexity with the key escrow service, but also requires the channel parties to update the secret for each new payment through the smart contract on $\mathcal{L}_2$, making it expensive, slow, and non-scalable.

5.3 AuxChannel

We address the remaining challenges in our second attempt by leveraging our newly introduced CVES as a core building block. In particular, thanks to the consecutiveness property, both channel parties only need to provide the initial secret to the key escrow service (reducing the communication complexity to $O(1)$) when establishing a channel, processing all payments without interacting with the key escrow service, and only asking the key escrow service to release the secret to close the channel when one party is not cooperating.

For the ease of understanding, we only present the basic idea of AuxChannel below and leave the formal and detailed presentation to the next section. Loosely speaking, AuxChannel has four phases: *channel establishment, channel update, channel closure*, and *channel dispute*. For simplicity, we assume all the messages are correctly verified before being accepted.

In the channel establishment phase, each channel party executes $EncGen(\cdot)$ to generate a pair of decryption key and encryption key, and share the output with the distributed and verifiable key escrow service. The escrow service will include a commitment on the received secret in its smart contract with the two channel parties on $\mathcal{L}_2$.

Each of the channel parties performs $\mathsf{EncSign}(\cdot)$ to create an encrypted signature on a mutually agreed transaction, such that the full signature can be recovered by using the associated decryption key kept by the escrow service on $\mathcal{L}_2$. To validate a transaction, one recovers the full signature on the transaction from the encrypted signature of its counterparty, and creates a mutually signed transaction by adding its own signature to the recovered full signature.

Each new payment redistributes the channel balance as agreed by both parties. Payments can be performed via channel update phase, where for each payment the payer updates the decryption and encryption key pair by performing $\mathsf{KeyUpdate}(\cdot)$, and creates an encrypted signature on the new payment by performing $\mathsf{EncSign}(\cdot)$ with the signing key and the updated encryption key.

To close a channel, channel parties share their latest decryption key with each other, perform $\mathsf{DecSig}(\cdot)$ with the counterparty's decryption key, and validate the transaction representing the latest state of the channel balance by creating a multi-signature on the transaction.

In the case where one party is not responding to the channel closure request, the other party can resolve the dispute by asking the key escrow service to release the secret required to close the channel. Thanks to the consecutiveness property of our introduced CVES, if the initial secret shared with the escrow service by one party is released, the other party can derive the latest secret recover the full signature to close the channel, presenting a neat and efficient solution to the guaranteed channel closing.

As a malicious party may request the secret to be released for a revoked state in order to get more coins than its actual balance, it is important for the key escrow service to give enough time for the counterparty to verify and react to the secret releasing request on $\mathcal{L}_2$. This can be achieved by using standard solutions, such as a time-lock contract. We defer more details to the next section.

6 FORMAL AUXCHANNEL DESCRIPTION

6.1 System Model

We consider two blockchains, $\mathcal{L}_1$, which does not support script language, and $\mathcal{L}_2$, which is script-enabled. For simplicity, we assume that if a valid transaction is propagated to the blockchain network, it cannot be censored and will be immediately included in the "permanent" part of the blockchain.

Two participants want to establish a payment channel on $\mathcal{L}_1$. Once a channel is established, the two participants become "channel parties". For simplicity, this paper always uses channel parties to refer to them. We assume that both channel parties have accounts with sufficient coins on $\mathcal{L}_1$ and $\mathcal{L}_2$ in participating AuxChannel. We assume that they are always online, even though this assumption can be removed by deploying watchtower-like services [5, 21, 26]. We assume a distributed and verifiable key escrow service (KES), such as [6], on $\mathcal{L}_2$. The KES keeps secret keys for its clients and releases the key when a pre-defined condition is met. We assume that the communication channels among channel parties and the Key Escrow Service (KES) are authenticated e.g. through cryptographically secure digital signature.

6.2 Formalized AuxChannel

Let $\mathcal{P}_A$ and $\mathcal{P}_B$ be the two channel parties with signing-verification key-pairs (sk_A, pk_A) and (sk_B, pk_B) on $\mathcal{L}_1$, and (sk'_A, pk'_A) and (sk'_B, pk'_B) on $\mathcal{L}_2$, respectively, and they have exchanged public keys with each other. Let $Tx := In||Out$ be the construction of the transaction on $\mathcal{L}_1$, where In denotes the input of Tx and Out denotes the output of Tx. We present AuxChannel with four phases, namely establishment, update, closure, and dispute.

Establishment To establish a channel, two channel parties collectively create a funding transaction Tx_f to make their deposit on $\mathcal{L}_1$. To initialise the establishment, each of the parties generates initial decryption keys $(dk^0_{\mathcal{P}}, \hat{dk}^0_{\mathcal{P}})$ and encryption key $ek^0_{\mathcal{P}}$ by executing $\mathsf{EncGen}(\lambda)$, where $\mathcal{P} \in \{A, B\}$. They then interact with KES to share their $(cid, dk^0_{\mathcal{P}}, ek^0_{\mathcal{P}})$, where cid is a unique identifier of the channel. KES verifies the received messages and records (cid, ek^0_A, ek^0_B) on $\mathcal{L}_2$ as a confirmation of providing key escrow service to both parties. The conditions to release decryption keys by the KES will be presented in the dispute phase. Upon seeing the confirmation from KES:

(1) both channel parties create a funding transaction

$$Tx_f := (pk_A : bal^0_A, pk_B : bal^0_B)||(pk_M : bal_M)$$

to transfer their balance (bal^0_P of address pk_P) to a multi-signature account (identified by using pk_M), where the balance bal_M is the capacity of the to be established channel. They also create an encrypted signature $\hat{\sigma}^0_{\mathcal{P}} \leftarrow \mathsf{EncSign}(sk_{\mathcal{P}}, ek^i_{\mathcal{P}}, Tx^0_c)$ over a newly created commitment transaction

$$Tx^0_c := (pk_M : bal_M)||(pk_A : bal^0_A, pk_B : bal^0_B).$$

They keep the funding transaction private and exchange $\hat{\sigma}^0_{\mathcal{P}}$ with each other.

(2) If the received $\hat{\sigma}^0_{\mathcal{P}}$ passed the verification $\mathsf{EncVerify}(pk_{\mathcal{P}}, ek^0_{\mathcal{P}}, \hat{\sigma}^0_{\mathcal{P}}, Tx^0_c)$, they exchange locally signed Tx_f and post the cross-signed Tx_f on $\mathcal{L}_1$ indicating the completion of channel establishment.

Update To update a channel from the channel state index $i - 1$ to i, where $i \in \mathbb{N}^+$, both channel parties agree on the commitment transaction Tx^i_c and their encryption keys (ek^i_A, ek^i_B). The update phase is presented as follows:

(1) Both channel parties update their decryption keys $(dk^i_{\mathcal{P}}, \hat{dk}^i_{\mathcal{P}})$ and encryption key $ek^i_{\mathcal{P}}$ by executing $(dk^i_{\mathcal{P}}, \hat{dk}^i_{\mathcal{P}}, ek^i_{\mathcal{P}}) \leftarrow \mathsf{KeyUpdate}(dk^{i-1}_{\mathcal{P}})$. They also create the proof of encryption keys' consecutiveness $P^i_{\mathcal{P}} \leftarrow \mathsf{ConsProof}(dk^{i-1}_{\mathcal{P}}, dk^i_{\mathcal{P}})$, an encrypted signature $\hat{\sigma}^i_{\mathcal{P}} \leftarrow \mathsf{EncSign}(sk_{\mathcal{P}}, ek^i_{\mathcal{P}}, Tx^i_c)$ over a newly created commitment transaction

$$Tx^i_c := (pk_M : bal_M)||(pk_A : bal^i_A, pk_B : bal^i_B),$$

and exchange message $(ek^i_{\mathcal{P}}, \hat{\sigma}^i_{\mathcal{P}}, P^i_{\mathcal{P}})$;

(2) If the received $(ek^i_{\mathcal{P}}, \hat{\sigma}^i_{\mathcal{P}}, P^i_{\mathcal{P}})$ passed the two verifications $\mathsf{EncVerify}(pk_{\mathcal{P}}, ek^i_{\mathcal{P}}, \hat{\sigma}^i_{\mathcal{P}}, Tx^i_c)$ and $\mathsf{ConsVerify}(ek^{i-1}_{\mathcal{P}}, ek^i_{\mathcal{P}}, P^i_{\mathcal{P}})$, where $ek^{i-1}_{\mathcal{P}}$ is received and stored from the counter-party at the previous state, they create and exchange a signature $\sigma'^i_{\mathcal{P}}$ over the channel state index i and both channel parties' encryption keys (ek^i_A, ek^i_B) by using $sk'_{\mathcal{P}}$.

Closure The channel can be closed collectively by the two parties. Both parties exchange their latest decryption keys, dk^i_A and dk^i_B, and derive the corresponding signature $\sigma^i_{\mathcal{P}} \leftarrow \mathsf{DecSign}(\hat{dk}^i_{\mathcal{P}}, \hat{\sigma}^i_{\mathcal{P}})$ on Tx^i_c, where $\hat{dk}^i_{\mathcal{P}}$ is the truncated $dk^i_{\mathcal{P}}$ (defined in Section 4). Thus, either of them can close the channel with a latest state by posting a cross signed Tx^i_c.

Dispute Both channel parties resolve their dispute through $\mathcal{L}_2$ and the KES will release the secret per their predefined condition, as follows.

In the predefined condition, any channel party can request the KES to release the counter-party's initial decryption key by uploading a trigger transaction to $\mathcal{L}_2$, which starts a timer for the counter party to react before timeout. Assuming that $\mathcal{P}_B$ triggers the timer by posting a trigger transaction $Tx^i_{tr,B}$, which includes the decryption key dk^i_B and the message $(\sigma'^i_A, \sigma'^i_B, i, ek^i_A, ek^i_B)$.

Before the timeout occurs, if the index i represents the latest state of the channel, $\mathcal{P}_A$ releases her decryption key dk^i_A by posting a release transaction $Tx^i_{r,A}$; otherwise, $\mathcal{P}_A$ uploads an update transaction $Tx^j_{up,A}$, where $j > i$, and KES releases dk^0_B on $\mathcal{L}_2$. Upon timeout, if $\mathcal{P}_A$ did not have any response, KES releases dk^0_A.

If $\mathcal{P}_A$ has released her decryption key dk^i_A within the timer, $\mathcal{P}_B$ can perform $\mathsf{DecSign}(\hat{dk}^i_{\mathcal{P}}, \hat{\sigma}^i_{\mathcal{P}})$ to derive the commitment transaction Tx^i_c signed by $\mathcal{P}_A$. Similarly, as dk^i_B is already posted on $\mathcal{L}_2$, $\mathcal{P}_A$ can also derive the commitment transaction Tx^i_c signed by $\mathcal{P}_B$. Thus, either of them can close the channel with the latest state by posting a cross signed Tx^i_c.

If the KES has released $dk^0_{\mathcal{P}}$ (for $\mathcal{P} \in \{A, B\}$) on $\mathcal{L}_2$ according to the conditions above, then the corresponding counter party can derive the decryption keys $(dk^{i'}_{\mathcal{P}}, \hat{dk}^{i'}_{\mathcal{P}})$ of any state $i' \in [1, j]$ through

$(dk_{\mathcal{P}}^{i'}, \hat{dk}_{\mathcal{P}}^{i'}, ek_{\mathcal{P}}^{i'}) \leftarrow \mathsf{KeyUpdate}(dk_{\mathcal{P}}^{(i'-1)})$, and then derive the commitment transaction $Tx_c^{i'}$ signed by $\mathcal{P}$. This enables the counter party to close the channel with any channel state. In other words, when a party is not following the protocol, the counter party is able to close the channel at any chosen state (to its advantage) with the help of KES. This is considered a punishment to the misbehaved party.

6.3 Security Analysis

Similar to Perun [16], we also employ the following ***security goals of AuxChannel***:

1. *Guaranteed channel closing*: a channel can be closed by any of the channel parties;
2. *Guaranteed balance payout for end users*: when a channel is closed, either both channel parties retrieve their latest balances or the honest party obtains no less than his latest balance.

As AuxChannel employs Schnorr CVES scheme to update channel party's decryption keys and encrypted signatures at each state, we reduce the security of AuxChannel to the security of CVES scheme in Theorem 2.

Theorem 2. *AuxChannel achieves the two security goals if the underlying CVES scheme is secure.*

We present the proof sketch of Theorem 2 as follows:
Proof. We prove it by contradiction. Let $\mathcal{P}_A$ be the adversary $\mathcal{A}$, who has the ability to break the security goals of AuxChannel, we can make use of $\mathcal{P}_A$ to break the security of Schnorr-based CVES scheme.

For the first security goal: guaranteed channel closing. As we assume that $\mathcal{P}_A$ has the ability to prevent the channel from closing, $\mathcal{P}_A$ would not release any of her decryption keys. Thus, KES sends the initial decryption keys dk_A^0 to $\mathcal{P}_B$. As $\mathcal{P}_A$ has the ability to prevent the channel from closing, the dk_A^0 cannot help $\mathcal{P}_B$ to recover a valid $\sigma_A^{i'}$ at any intermediate state i', where $i' \in [0, i]$. Thus, the channel cannot be closed without $\mathcal{P}_A$'s cooperation.

We can make use of $\mathcal{P}_A$ to generate a proof $P_A^{i'+1*}$ on two decryption keys $dk_A^{i'}$ and $dk_A^{i'+1*}$, where:

$$(dk_A^{i'+1*}, \hat{dk}_A^{i'+1*}, ek_A^{i'*}) \leftarrow \mathcal{A}_0(dk_A^{i'}),$$
$$(dk_A^{i'+1}, \hat{dk}_A^{i'+1}, ek_A^{i'}) \leftarrow \mathsf{KeyUpdate}(dk_A^{i'}),$$
$$\sigma_A^{i'+1*} \leftarrow \mathsf{EncSign}(sk_A, ek_A^{i'+1*}, Tx_c^{i'+1*}),$$
$$P_A^{i'+1*} \leftarrow \mathcal{A}_1(dk_A^{i'}, dk_A^{i'+1*}),$$
$$1 \leftarrow \mathsf{EncVerify}(pk_A, ek_A^{i'+1*}, \sigma_A^{i'+1*}, Tx_c^{i'+1})$$
$$1 \leftarrow \mathsf{ConsVerify}(ek_A^{i'}, ek_A^{i'+1*}, P_A^{i'+1*}),$$
$$\sigma_A^{i'+1*} \leftarrow \mathsf{DecSign}(\hat{dk}^{i'+1}, \hat{\sigma}_A^{i'+1*}),$$
$$1 \not\leftarrow \mathsf{Verify}(pk_A, \sigma_A^{i'+1*}, Tx_c^{i'+1}).$$

Or we can also make use of $\mathcal{P}_A$ to create a valid encrypted signature $\hat{\sigma}_A^{i'+1*}$, but $\mathcal{P}_B$ cannot derive a valid $\sigma_A^{i'+1}$ from $\hat{\sigma}_A^{i'+1*}$, where:

$$(dk_A^{i'}, \hat{dk}_A^{i'}, ek_A^{i'}) \leftarrow \mathsf{KeyUpdate}^{i'}(dk_A^{j})_{j\in[0,i'-1]},$$
$$(dk_A^{i'+1}, \hat{dk}_A^{i'+1}, ek_A^{i'+1}) \leftarrow \mathsf{KeyUpdate}(dk_A^{i'}),$$
$$P_A^{i'+1} \leftarrow \mathsf{ConsProof}(dk_A^{i'}, dk_A^{i'+1}),$$
$$1 \leftarrow \mathsf{ConsVerify}(ek_A^{i'}, ek_A^{i'+1}, P_A^{i'+1}),$$
$$\hat{\sigma}_A^{i'+1*} \leftarrow \mathcal{A}(sk_A, ek_A^{i'+1}, Tx_c^{i'+1}),$$
$$1 \leftarrow \mathsf{EncVerify}(pk_A, ek_A^{i'+1}, \hat{\sigma}_A^{i'+1*}, Tx_c^{i'+1}),$$
$$\sigma_A^{i'+1*} \leftarrow \mathsf{DecSign}(\hat{dk}_A^{i'+1}, \hat{\sigma}_A^{i'+1*}),$$
$$1 \not\leftarrow \mathsf{Verify}(pk_A, \sigma_A^{i'+1*}, Tx_c^{i'+1}).$$

Thus, $\mathcal{P}_A$ breaks *the consecutive verifiability* and *the correctness* of Schnorr CVES scheme.

For the second security goal: guaranteed payout of channel parties. We assume that $\mathcal{P}_A$ has the highest balance at an intermediate channel state i', and she has the ability to close the channel at state i' with an non-negligible possibility. That is $\mathcal{P}_A$ can derive $\sigma_B^{i'}$, and post a cross-signed $Tx_c^{i'}$ on $\mathcal{L}_1$. Thus, she steals coins from $\mathcal{P}_B$.

Thus, we can make use of $\mathcal{P}_A$ to forge a valid signature $\sigma_B^{i'}$ by using pk_B, $ek_B^{i'}$, $\hat{\sigma}_B^{i'}$, and the proof $P_B^{i'}$, where:

$$\sigma_B^{i'} \leftarrow \mathcal{A}(pk_B, ek_B^{i'}, \hat{\sigma}_B^{i'}, P_B^{i'}),$$
$$1 \leftarrow \mathsf{Verify}(pk_B, \sigma_B^{i'}, Tx_c^{i'}).$$

We can also make use of $\mathcal{P}_A$ to reverse $dk_B^{i'}$ from dk_B^{i}, where

$$dk_B^{i'} \leftarrow \mathcal{A}(dk_B^{i}),$$
$$\sigma_B^{i'} \leftarrow \mathsf{DecSign}(\hat{dk}_A^{i'}, \hat{\sigma}_B^{i'}),$$
$$1 \leftarrow \mathsf{Verify}(pk_B, \sigma_B^{i'}, Tx_c^{i'}).$$

No matter which scenario happens, $\mathcal{P}_A$ breaks *unforgeability* or *one-wayness* of Schnorr CVES scheme.

In summary, if one of the channel parties can break the security goal, either *guaranteed channel closing* or *guaranteed payout of channel parties*, of AuxChannel, it also breaks the security of CVES scheme, which is contradict to the assumptions. Theorem 2 is proved then. □

7 DISCUSSION AND CONCLUSION

Payment channel networks (PCN) enable offchain payments between users that have not established a payment channel between them, by routing payments via a path of payment channels. This section provides an intuition on how to make a payment via multiple-hop AuxChannel.

Normally, when making a payment via multi-hop channels, two core requirements should be met. First, all payments in the path should be **atomic**, meaning that either they all succeed or they all failed. For Bitcoin-compatible blockchains, atomicity is usually guaranteed by using Hash-time Lock Contract (HTLC) [30], the coins locked in HTLC can only be released when some conditions are met. Second, all the on-path channels are **unlockable** even when the parties involved are crashed or malicious [27, 39], This is normally guaranteed by making penalty for the malicious [25], which is used for compensating parties who incurred loss by locking funds.

Assuming that Alice (A) has a channel with Bob (B), who has a channel with Carol (C), who has a channel with Dave (D), the path of these channels could be considered like: $A \leftrightarrow B \leftrightarrow C \leftrightarrow D$. The multi-hop payment from A to D requires that each intermediate channel, namely the channels between A and B, B and C, and C and D, to lock a payment. Such that once D unlocks the payment from C, C can unlock the payment from B, and B can unlock the payment from A. Thus, A obtains a witness as receipt of paying to D. Otherwise, B and C can unlock their payments (between (A and B) and (B and C)) after a pre-defined time, and A obtain the receipt as well. Under this scenario, A can still prove that she has paid to D by using the receipt, and D has motivation to unlock the payment from C, or he never get paid. Thus, all the intermediate payments are *atomic* and *unlockable*.

Supporting payment channel network in AuxChannel requires addressing two challenges: first, how to lock an encrypted signature within a channel; second, for B and C, how to unlock the locked signatures when D does not cooperate to unlock his payment for whatever the reason.

For the first challenge, we give the informal definition of *locked signature* below. Using Schnorr signature as an example, let $\sigma = (R, s)$ be the *full signature* in AuxChannel, where $s = x \cdot H(X||R||m) + (\hat{r}^* + y + z)$, x is the signing key, $X = g^x$ is the corresponding verification key, $\hat{r}^*$ is a random value, y is the decryption key, z is the locking key, and $R := g^{\hat{r}^*+y+z}$. Let $\hat{\sigma} = (\hat{R}, \hat{s})$ be the *encrypted signature*, where $\hat{s} = x \cdot H(R||m) + (\hat{r}^* + z)$ and $\hat{R} := g^{\hat{r}^*+z}$. Let $\hat{\sigma}^* := (\hat{R}^*, \hat{s}^*)$ be the *locked signature*, where $\hat{s}^* = x \cdot H(R||m) + \hat{r}^*$ and $\hat{R}^* := g^{\hat{r}^*}$. Unlocking a locked signature means revealing the encrypted signature $\hat{s}$.

For the second challenge, in a scriptless blockchain, each intermediate unlocking key is locked by a timed commitment [36, 37], such that the receiver of each payment can force the opening of the commitment and learn the unlocking key only after a pre-specified time.

Suppose that Alice wants to pay x coins to Dave via the path of AuxChannel: $A \leftrightarrow B \leftrightarrow C \leftrightarrow D$. The multi-hop payment has two rounds. First, all intermediate channels are locked from Dave to Alice sequentially. For the channel between Dave and Carol:

- Dave produces an unlocking-locking key pair (z_D, Z_D), where $Z_D = g^{z_D}$ and a timed commitment $Comm_D$ on z_D, and forwards $(Z_D, Comm_D)$ to Carol, such that Carol can force open $Comm_D$ and obtain z_D after a pre-specified time t_D.
- Dave and Carol make locked signatures $\hat{\sigma}^*_D$ and $\hat{\sigma}^*_C$ on their newest channel balances by using same locking key Z_D respectively.
- Dave and Carol make their signatures σ'_D and σ'_C on the message $i||ek^i_C||ek^i_D$, where i is the newest channel state index, by using their signing keys on the script-enabled blockchain respectively, and exchange $Enc_{Z_D}(\sigma'_D)$ and $Enc_{Z_D}(\sigma'_C)$, where $Enc_{Z_D}(\cdot)$ can be any encryption algorithms w.r.t to the encryption key Z_D, such that $\sigma'_D = Dec_{z_D}(Enc_{Z_D}(\sigma'_D))$, where Dec_{z_D} is the corresponding decryption key w.r.t the decryption key z_D.

Carol picks z_C and produces the locking key $Z_C = g^{z_D} \cdot g^{z_C}$, and repeats the above procedure with Bob, who then picks z_B and produces the locking key $Z_B = g^{z_D} \cdot g^{z_C} \cdot g^{z_B}$, and repeats the above procedure with Alice.

Second, Dave releases his encrypted signature $\hat{\sigma}_D$, and Carol calculates $z_C = \hat{\sigma}^*_D - \hat{\sigma}_D$ and produces $\hat{\sigma}_C = \hat{\sigma}^*_C + z_D$. Dave and Carol obtain $\sigma'_C = Dec_{z_D}(Enc_{Z_D}(\sigma'_C))$ and $\sigma'_D = Dec_{z_D}(Enc_{Z_D}(\sigma'_D))$ respectively, and the channel between Dave and Carol is updated. Carol then unlocks the payment and updates the channel with Bob, who then unlocks the payment and updates the channel with Alice.

Under the worst case that Dave does not release his encrypted signature $\hat{\sigma}_D$, Carol can force open $Comm_D$ and obtain z_D after the time t_D. As Carol is paid from Bob, she is motivated to release her encrypted signature $\hat{\sigma}_{C'}$ to Bob; Otherwise, Bob and Alice can force open $Comm_C$ and $Comm_B$ and obtain z_C and z_B after the time t_C and t_B respectively to unlock their payment and update their channel.

However, there are still some issues when running payment channel network for scriptless blockchain. For example, some privacy issues on Lightning Network [30], the payment channel network of Bitcoin, has been exploited by some recent works [7, 23], while most scriptless blockchains have privacy functionality, such as Monero, these issues may prevent payment channel network from been deployed on the privacy-preserving blockchains. These issues are non-trivial and require depth-in analysis. We put them as open questions for future work.

In conclusion, we propose AuxChannel, the first bi-directional payment channel protocol for scriptless blockchains. It supports duplex payments in a channel with unlimited lifespan. The design concept is also a new direction for enabling payment channel with scriptless blockchains. We also see CVES as an independent interest as this may be applied to other applications.

REFERENCES

[1] Ermyas Abebe, Yining Hu, Allison Irvin, Dileban Karunamoorthy, Vinayaka Pandit, Venkatraman Ramakrishna, and Jiangshan Yu. 2021. Verifiable Observation of Permissioned Ledgers. In *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2021, Sydney, Australia, May 3-6, 2021*. IEEE, 1–9.

[2] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. 2020. Generalized Bitcoin-Compatible Channels. *IACR Cryptol. ePrint Arch.* (2020), 476. https://eprint.iacr.org/2020/476

[3] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostakova, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. 2020. Generalized Channels from Limited Blockchain Scripts and Adaptor Signatures. Cryptology ePrint Archive, Report 2020/476. https://ia.cr/2020/476.

[4] Lukas Aumayr, Sri AravindaKrishnan Thyagarajan, Giulio Malavolta, Pedro Monero-Sánchez, and Matteo Maffei. 2021. Sleepy Channels: Bitcoin-Compatible Bi-directional Payment Channels without Watchtowers. Cryptology ePrint Archive, Report 2021/1445. https://ia.cr/2021/1445.

[5] Zeta Avarikioti, Orfeas Stefanos Thyfronitis Litos, and Roger Wattenhofer. 2020. Cerberus Channels: Incentivizing Watchtowers for Bitcoin. In *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers (Lecture Notes in Computer Science, Vol. 12059)*, Joseph Bonneau and Nadia Heninger (Eds.). Springer, 346–366. https://doi.org/10.1007/978-3-030-51280-4_19

[6] Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. 2020. Can a Public Blockchain Keep a Secret?. In *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12550)*, Rafael Pass and Krzysztof Pietrzak (Eds.). Springer, 260–290.

[7] Alex Biryukov, Gleb Naumenko, and Sergei Tikhomirov. 2021. Analysis and Probing of Parallel Channels in the Lightning Network. *IACR Cryptol. ePrint Arch.* 2021 (2021), 384.

[8] Manuel Blum, Paul Feldman, and Silvio Micali. 1988. Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois,*

USA, Janos Simon (Ed.). ACM, 103–112. https://doi.org/10.1145/62212.62222

[9] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. 2003. Aggregate and verifiably encrypted signatures from bilinear maps. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 416–432.

[10] Fabrice Boudot. 2000. Efficient proofs that a committed number lies in an interval. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 431–444.

[11] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 315–334.

[12] Jan Camenisch, Rafik Chaabouni, and Abhi Shelat. 2008. Efficient Protocols for Set Membership and Range Proofs. In *Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 5350)*, Josef Pieprzyk (Ed.). Springer, 234–252. https://doi.org/10.1007/978-3-540-89255-7_15

[13] David Chaum and Torben Pryds Pedersen. 1992. Wallet databases with observers. In *Annual International Cryptology Conference*. Springer, 89–105.

[14] I. Damgård and E. Fujisaki. 2001. An Integer Commitment Scheme based on Groups with Hidden Order. *IACR Cryptol. ePrint Arch.* 2001 (2001), 64.

[15] C. Decker and R. Russell. 2018. Eltoo : a simple layer 2 protocol for Bitcoin.

[16] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. 2017. PERUN: Virtual Payment Channels over Cryptographic Currencies. *IACR Cryptol. ePrint Arch.* 2017 (2017), 635.

[17] Muhammed F. Esgin, Oguzhan Ersoy, and Zekeriya Erkin. 2020. Post-Quantum Adaptor Signatures and Payment Channel Networks. In *Computer Security - ESORICS 2020 - 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14-18, 2020, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 12309)*, Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider (Eds.). Springer, 378–397. https://doi.org/10.1007/978-3-030-59013-0_19

[18] Lloyd Fournier. 2019. One-Time Verifiably Encrypted Signatures AKA Adaptor Signatures. https://github.com/LLFourn/one-time-VES/blob/master/main.pdf

[19] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1989. The knowledge complexity of interactive proof systems. *SIAM Journal on computing* 18, 1 (1989), 186–208.

[20] Matthew Green and Ian Miers. 2017. Bolt: Anonymous Payment Channels for Decentralized Currencies. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 473–489. https://doi.org/10.1145/3133956.3134093

[21] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. 2020. SoK: Layer-two blockchain protocols. In *International Conference on Financial Cryptography and Data Security*. Springer, 201–226.

[22] Joel Gugger. [n. d.]. Bitcoin-Monero Cross-chain Atomic Swap. https://github.com/h4sh3d/xmr-btc-atomic-swap 2020.

[23] George Kappos, Haaroon Yousaf, Ania Piotrowska, Sanket Kanjalkar, Sergi Delgado-Segura, Andrew Miller, and Sarah Meiklejohn. 2021. An empirical analysis of privacy in the lightning network. In *International Conference on Financial Cryptography and Data Security*. Springer, 167–186.

[24] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. 2019. Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/anonymous-multi-hop-locks-for-blockchain-scalability-and-interoperability/

[25] Subhra Mazumdar, Prabal Banerjee, and Sushmita Ruj. 2020. Time is Money: Countering Griefing Attack in Lightning Network. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 1036–1043. https://doi.org/10.1109/TrustCom50675.2020.00138

[26] Arash Mirzaei, Amin Sakzad, Jiangshan Yu, and Ron Steinfeld. 2021. FPPW: A Fair and Privacy Preserving Watchtower For Bitcoin. Cryptology ePrint Archive, Report 2021/117. https://eprint.iacr.org/2021/117.

[27] Ayelet Mizrahi and Aviv Zohar. 2020. Congestion Attacks in Payment Channel Networks. *CoRR* abs/2002.06564 (2020). arXiv:2002.06564 https://arxiv.org/abs/2002.06564

[28] Pedro Moreno-Sanchez, Arthur Blue, Duc V Le, Sarang Noether, Brandon Goodell, and Aniket Kate. 2020. DLSAG: non-interactive refund transactions for interoperable payment channels in Monero. In *International Conference on Financial Cryptography and Data Security*. Springer, 325–345.

[29] Christopher Natoli, Jiangshan Yu, Vincent Gramoli, and Paulo Jorge Esteves Veríssimo. 2019. Deconstructing Blockchains: A Comprehensive Survey on Consensus, Membership and Structure. *CoRR* abs/1908.08316 (2019). arXiv:1908.08316 http://arxiv.org/abs/1908.08316

[30] Joseph Poon and Thaddeus Dryja. 2016. The Bitcoin lightning network: scalable off-chain instant payments.

[31] Xianrui Qin, Handong Cui, and Tsz Hon Yuen. 2021. Generic Adaptor Signature. *IACR Cryptol. ePrint Arch.* 2021 (2021), 161. https://eprint.iacr.org/2021/161

[32] Mike Hearn Satoshi Nakamoto. 2013. Anti DoS for tx Repleasement - Nakamoto's Idea of High-frequency Trades. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002417.html

[33] Omer Shlomovits and Oded Leiba. 2020. JugglingSwap: scriptless atomic cross-chain swaps. *CoRR* abs/2007.14423 (2020). arXiv:2007.14423 https://arxiv.org/abs/2007.14423

[34] Jeremy Spilman. 2013. Anti DoS for tx Repleasement - Spilman-styled Payment Channel. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html

[35] Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. 2021. Post-Quantum Adaptor Signature for Privacy-Preserving Off-Chain Payments. In *Financial Cryptography and Data Security - 25th International Conference, FC 2021, Virtual Event, March 1-5, 2021, Revised Selected Papers, Part II (Lecture Notes in Computer Science, Vol. 12675)*, Nikita Borisov and Claudia Diaz (Eds.). Springer, 131–150. https://doi.org/10.1007/978-3-662-64331-0_7

[36] Sri Aravinda Krishnan Thyagarajan, Tiantian Gong, Adithya Bhat, Aniket Kate, and Dominique Schröder. 2021. OpenSquare: Decentralized Repeated Modular Squaring Service. (2021), 1273. https://eprint.iacr.org/2021/1273

[37] Sri Aravinda Krishnan Thyagarajan, Giulio Malavolta, Fritz Schmidt, and Dominique Schröder. 2020. PayMo: Payment Channels For Monero. *IACR Cryptol. ePrint Arch.* (2020), 1441. https://eprint.iacr.org/2020/1441

[38] Sri Aravinda Krishnan Thyagarajan, Giulio Malavolta, Fritz Schmidt, and Dominique Schröder. 2020. PayMo: Payment Channels For Monero. Cryptology ePrint Archive, Report 2020/1441. https://eprint.iacr.org/2020/1441.

[39] Sergei Tikhomirov, Pedro Moreno-Sanchez, and Matteo Maffei. 2020. A Quantitative Analysis of Security, Anonymity and Scalability for the Lightning Network. In *IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2020, Genoa, Italy, September 7-11, 2020*. IEEE, 387–396. https://doi.org/10.1109/EuroSPW51379.2020.00059

[40] Florian Tschorsch and Björn Scheuermann. 2016. Bitcoin and beyond: A technical survey on decentralized digital currencies. *IEEE Communications Surveys & Tutorials* 18, 3 (2016), 2084–2123.

[41] Xlab. 2020. Library for zero-knowledge proof based applications (like anonymous credentials). https://github.com/xlab-si/emmy.

[42] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J. Knottenbelt. 2019. SoK: Communication Across Distributed Ledgers. *IACR Cryptol. ePrint Arch.* 2019 (2019), 1128.

[43] Zcash. [n. d.]. Zcash. https://z.cash/zh/

[44] Yuncong Zhang, Yu Long, Zhen Liu, Zhiqiang Liu, and Dawu Gu. 2019. Z-Channel: Scalable and efficient scheme in Zerocash. *Comput. Secur.* 86 (2019), 112–131. https://doi.org/10.1016/j.cose.2019.05.012

A SUPPLEMENTARY OF CVES

This section gives the security analysis of our proposed Schnorr-based CVES scheme in Section 4.3.

Lemma A.1. *The Schnorr CVES is correct.*

***Proof*.** In our Schnorr CVES (Figure 2), we have that sk is (x, X), and the corresponding verification key pk is $X := g^x$. A decryption key dk^i is $(\tilde{y}, y, Y)$ w.r.t the encryption key $ek^i := (\tilde{Y}, Y)$, where $y := \tilde{y} \bmod p$ and $\tilde{Y} := h^{\tilde{y}} \bmod N, Y := g^y$. Since $g^{\hat{s}}X^{-c} = g^{\hat{s}-cx} = g^{(\hat{r}+cx)-cx} = g^{\hat{r}} = \hat{R}$, $c := H(m_i||\hat{R}Y||X)$, and $\hat{R} = g^{\hat{r}}$, it implies that $\mathsf{EncVerify}(pk, ek^i, \hat{\sigma}^i, m^i) = 1$. Similarly, as $R = g^r$, $c := H(m_i||R||X)$ and $g^sX^{-c} = g^{s-cx} = g^{(r+cx)-cx} = g^r = R$, $\mathsf{Verify}(pk, \sigma_i, m_i) = 1$ always holds. Thus the encrypted signature $\hat{\sigma}_i := (\hat{R}, \hat{s})$ and the original signature $\sigma_i := (R, s)$ are valid.

As an encrypted signature $\hat{\sigma}^i$ is $(\hat{R}, \hat{s})$, the corresponding truncated decryption key $\hat{dk}^i$ is (y, Y), and the original signature σ_i is $(R, s) = (\hat{R}Y, \hat{s} + y)$, we have $s = \hat{s} + y$, $y = s - \hat{s} = \hat{s} + y - \hat{s}$, and $Y = g^y = g^{\hat{s}+y-\hat{s}}$. Thus $\mathsf{Rec}(\sigma, \hat{\sigma})$ outputs $\hat{dk}^i = (y, Y)$ always holds. That is the Schnorr CVES is recoverable.

Also, the i-th decryption key dk^i is $(\tilde{y}, y, Y)$ and $(i + 1)$-th decryption key dk^{i+1} is $(\tilde{y}', y', Y') = (\tilde{y}^2 \bmod N), \tilde{y}' \bmod p, g^{y'})$ and encryption key ek^{i+1} is $(\tilde{Y}', Y') = ((h^{\tilde{y}'} \bmod N), (g^{y'}))$. According to our Schnorr CVES construction in Figure 2, the NIZK system proves the following proof-of-knowledge:

$$
\begin{aligned}
PoK((\tilde{y}, \tilde{y}', y, y', k) : \{Y = g^y \wedge Y' = g^{y'} \wedge \tilde{Y} = h^{\tilde{y}} \bmod N \wedge \\
\tilde{Y}' = h^{\tilde{y}'} \bmod N \wedge \tilde{Y}'(h^n)^k = \tilde{Y}^{\tilde{y}} \bmod N \wedge \\
Y = g^{\tilde{y}} \wedge Y' = g^{\tilde{y}'} \wedge 0 < \tilde{y}' < n\})
\end{aligned}
$$

in Appendix B, our instantiation simply sets $n = N = p'q'$.

Since $\tilde{y}' = \tilde{y}^2 - kN$ (or $\tilde{y}' = \tilde{y}^2 \bmod N$) and

$$
\begin{aligned}
\tilde{Y}'(h^n)^k &= \tilde{Y}'(h^{Nk}) = \tilde{Y}'(h^{\tilde{y}^2-\tilde{y}'}) = \tilde{Y}'(h^{\tilde{y}^2-(\tilde{y}^2 \bmod N)}) \\
&= (h^{\tilde{y}'} \bmod N)(h^{\tilde{y}^2-(\tilde{y}^2 \bmod N)}) \\
&= h^{\tilde{y}^2} \bmod N = (h^{\tilde{y}} \bmod N)^{\tilde{y}} \bmod N = \tilde{Y}^{\tilde{y}} \bmod N
\end{aligned}
$$

, it implies that $\mathsf{ConsVerify}(ek^i, ek^{i+1}, \mathsf{ConsProof}(dk^i, dk^{i+1})) = 1$. That is the Schnorr CVES is consecutive.

Lemma A.2. *Assuming that the discrete logarithm (DL) problem is hard and the underlying NIZK system is secure, the Schnorr CVES scheme is unforgeable under the random oracle model.*

***Proof*.** Assuming that there exists an adversary $\mathcal{A}$ who can break the unforgeability, then given a DL problem instance $(X, g) \in \mathbb{G}_p$, we can construct a simulator $\mathcal{S}_3$ who can make use of $\mathcal{A}$ to output an integer x such that $X = g^x$.

$\mathcal{S}_3$ picks X from the problem instance (while its DL x is unknown) and sets $pk := X$. For the initial and later stage decryption and encryption key-pairs and the proof on two consecutive decryption keys, $\mathcal{S}_3$ generates according to the algorithms EncGen, $\mathsf{KeyUpdate}$ and $\mathsf{ConsProof}$. $\mathcal{S}_3$ gives (pk, ek^i, P^i, g) to $\mathcal{A}$.

The game of forging a valid signature runs in three stages by $\mathcal{A}$: first $\mathcal{A}_0$ chooses a certain session and records the public keys (pk, ek^i, P^i, g), and then $\mathcal{A}_1$ chooses a message m, finally $\mathcal{A}_2$ outputs a forged signature $\sigma := (R, s)$ on m, such that $R = g^sX^{-H(m||R||X)}$, where X is the pk given to $\mathcal{A}$. Remark that m was not queried to $\mathcal{O}$ before and that signature is valid.

$\mathcal{S}_3$ simulates the random oracle H as normal (keep a table to make the consistency of input and output), and the encsigning oracle $\mathcal{E}$ (with input the public keys pk, ek^i, and the chosen message m) as follows:

- Extract X and Y from pk and ek^i;
- Pick $c \in \mathbb{Z}_p$ and $\hat{s} \in \mathbb{Z}_p$ randomly;
- Set $\hat{R} := g^{\hat{s}}X^{-c}$ and $R = \hat{R}Y$;
- Assign c to the value of the output of the random oracle query $H(m||R||X)$;
- Output $\hat{\sigma} = (\hat{R}, \hat{s})$.

Similar to the simulation of $\mathcal{E}$, $\mathcal{S}_3$ also simulates the signing oracle $\mathcal{O}$ (with input the public keys pk and the chosen message m) as follows:

- Extract X and Y from pk and ek^i;
- Pick $c \in \mathbb{Z}_p$ and $\hat{s}$, $s \in \mathbb{Z}_p$ randomly;
- Set $\hat{R} := g^{\hat{s}}X^{-c}$, $Y := g^{s-\hat{s}}$ and $R = \hat{R}Y$;
- Assign $\hat{s}$ to the value of output encsigning oracle query $\mathcal{E}(X, Y, m)$ and c to the value of the output of the random oracle query $H(m||R||X)$, if $H(m||R||X)$ has not been queried or assigned any value before in other oracles. Otherwise, repeat the process by choosing another different c;
- Output $\sigma = (R, s)$.

$\mathcal{A}$ can query the random oracle for q_H times, the encsigning oracle for q_E times, and the signing oracle for q_S times. At the end, $\mathcal{A}$ outputs a forged signature $\sigma^* := (R^*, s^*)$ on its chosen message m^*, where m^* is inputted to encsigning oracle, but not inputted to the signing oracle.

$\mathcal{S}_3$ then rewinds the random tape to the point that $\mathcal{A}$ is making the random oracle query for the value $c^* = H(m^*||R^*||X)$, and use another random tape to supply a different value $\tilde{c}^* = H(m^*||R^*||X)$, so that $\mathcal{A}$ outputs a different signature $\tilde{\sigma}^* = (R^*, \tilde{s}^*)$. [This is called the Forking Lemma.]

Now there are two forged signatures of m^* outputted by $\mathcal{A}$: $\sigma^* = (R^*, s^*)$ and $\tilde{\sigma}^* = (R^*, \tilde{s}^*)$ and they both satisfy the equation $R^* = g^{s^*}X^{-c^*}$, $R^* = g^{\tilde{s}^*}X^{-\tilde{c}^*}$ respectively. Then we have the following equations: $r^* = s^* + c^*x \bmod p$ and $r^* = \tilde{s}^* + \tilde{c}^*x \bmod p$ for two unknowns r^* and x, where $\hat{R}^* = g^{\hat{r}^*}$. By solving these two equations, $\mathcal{S}_3$ can output x which is the solution for the DL problem instance. □

Lemma A.3. *Assuming that the SQROOT problem is hard, the Schnorr CVES scheme satisfies one-wayness.*

***Proof*.** We prove this lemma by contradiction. Assuming that there exists an adversary $\mathcal{A}$ who can break the one-wayness of Schnorr CVES scheme. Given a SQROOT problem instance (φ, N), where $\varphi \in Q_N$, we can construct a simulator $\mathcal{S}_4$ who can make use of $\mathcal{A}$ to output an integer $\tilde{y}$ such that $\varphi = \tilde{y}^2 \bmod N$.

$\mathcal{S}_4$ picks φ from the SQROOT problem instance (φ, N) and sets $dk^i := (\varphi, y', Y')$, where $y' = \varphi \bmod p$ and $Y' = h^{y'} \bmod p$, and the corresponding encryption key is $ek^i := (\tilde{Y}', Y')$. For the signing key and verification key-pair (sk, pk), where $sk := (x, X)$ and $pk := X$, the initial and later stage decryption and encryption keys, $\mathcal{S}_4$ generates according to the algorithms Gen, EncGen and $\mathsf{KeyUpdate}$. $\mathcal{S}_4$ then gives (sk, dk^i, g, h, N) to $\mathcal{A}$.

The game of reversing a decryption key from it previous session runs in two stages: first, $\mathcal{A}_0$ chooses a certain session to attack and records the decryption key $dk^i = (\varphi, y', Y')$ and encryption key $ek^i := (\tilde{Y}', Y')$ (while the decryption key $dk^{i-1} = (\tilde{y}, y, Y)$ is unknown, such that $\tilde{y} := \varphi \bmod N$, $y := \tilde{y} \bmod p$, and $Y := g^y$), where $\tilde{Y}' := h^{\varphi} \bmod N$, then $\mathcal{A}_1$ outputs $dk^{i-1^*} := (\tilde{y}^*, y^*, Y^*)$, which satisfies:

$$(dk^{i*}, \hat{dk}^{i*}, ek^{i*}) \leftarrow \mathsf{KeyUpdate}(dk^{i-1*}) \tag{2}$$

$$\hat{\sigma}_i \leftarrow \mathsf{EncSign}(sk, ek^i, m^*) \tag{3}$$

$$\sigma_i^* \leftarrow \mathsf{DecSign}(\hat{dk}^{i*}, \hat{\sigma}_i) \tag{4}$$

$$1 \leftarrow \mathsf{Verify}(pk, m^*, \sigma_i^*) \tag{5}$$

where $dk^{i*} := (\tilde{y}^{'*}, y^{'*}, Y^{'*})$, $\hat{dk}^{i*} := (y^{'*}, Y^{'*})$, $ek^{'*} := (\tilde{Y}^{'*}, Y^{'*})$, $\hat{\sigma}_i := (\hat{R}, \hat{s})$ and $\sigma_i^* := (R^*, s^*)$. Let $\hat{R} = g^{\hat{r}}$. We can imply that $y'^* = \tilde{y}'^* \bmod p$ **(from 2)** $\Rightarrow y'^* = (\tilde{y}^{*^2} \bmod N) \bmod p$.
Let $R = g^r$, then $R = \hat{R}Y'$ **(from 3)** $\Rightarrow r = (\hat{r} + y') \bmod p, \hat{s} = (\hat{r} + cx) \bmod p$ **(from 3)** $\Rightarrow \hat{s} = (r - y' + cx) \bmod p$, where $c = H(m^*||R||X)$.

As all the inputs of equation 3 are generated according to the algorithms Gen, EncGen and KeyUpdate, according to the correctness of Schnorr we have $1 \leftarrow \mathsf{EncVerify}(pk, ek^i, \hat{\sigma})$, which implies that $\hat{R} = g^{\hat{s}}X^{-c}$. Thus, $R^* = \hat{R}Y'^*$ **(from 4)** $\Rightarrow R^* = g^{\hat{s}}X^{-c}g^{y'^*}$ On the other side, $R^* = g^{s^*}X^{-c^*}$ **(from 5)** $\Rightarrow R^* = g^{\hat{s}+y'^*}X^{-c^*}$ From the above equation, we have $R^* = g^{\hat{s}}X^{-c}g^{y'^*} = g^{\hat{s}+y'^*}X^{-c^*} \Rightarrow c = c^* \Rightarrow H(m^*||R||X) = H(m^*||R^*||X)$.

Remark that, for the same input, the hash function $H()$ produces a same output. Thus, if $c = c^*$, it implies that $m^*||R||X = m^*||R^*||X$. We then have:

$$R = R^* \Rightarrow \hat{R}Y' = \hat{R}Y'^* \Rightarrow y' = y'^*$$

$$\Rightarrow \varphi \bmod p = \tilde{y}'^* \bmod p = (\tilde{y}^{*^2} \bmod N) \bmod p$$

That is $\mathcal{S}_4$ can output $\tilde{y}^*$, which is the solution for the SQROOT problem instance (φ, N). □

Lemma A.4. *Assuming that the underlying NIZK system used in our* ConsProof *and* ConsVerify *is secure, the Schnorr CVES is consecutive verifiable.*

***Proof*.** We prove this lemma by contradiction. Assuming that, for a session i with the decryption key dk^i and encryption key ek^i, there exists an adversary $\mathcal{A}$ who can break the consecutive verifiability of Schnorr CVES scheme.

We can construct a simulator $\mathcal{S}_5$ who can make use of $\mathcal{A}$ to output a proof on a chosen NIZK problem instance (χ, w), which includes a chosen NP problem instance $\chi := (\tilde{Y}, \tilde{Y}'^*, Y, Y'^*)$ and a five-element tuple $w := (\tilde{y}, \tilde{y}'^*, y, y'^*, k)$, such that $\mathsf{NIZK.Verify}(\chi, P) = 1$, where $\tilde{y}'^*, y'^*, \tilde{Y}'^*, Y'^*$ are chosen by $\mathcal{A}$.

$\mathcal{S}_5$ picks $\tilde{y}, y, \tilde{Y}, Y$ from an NIZK problem instance (χ, w), where $\chi := (\tilde{Y}, \tilde{Y}'^*, Y, Y'^*)$, and $w := (\tilde{y}, \tilde{y}'^*, y, y'^*, k)$, then sets decryption key $dk^i := (\tilde{y}, y, Y)$ and the corresponding encryption key $ek^i := (\tilde{Y}, Y)$. For the signing and verification key-pair (sk, pk), where $sk := (x, X)$ and $pk := X$, the initial and later decryption keys , $\mathcal{S}_5$ generates according to the algorithms Gen, EncGen and KeyUpdate. $\mathcal{S}_5$ then gives (dk^i, ek^i, k) to $\mathcal{A}$.

The game of breaking the consecutive verifiablility of the Schnorr CVES scheme is defined in three stages: first $\mathcal{A}_0$ chooses a certain session and records the useful intermediate variable, then $\mathcal{A}_1$ outputs a decryption and encryption key-pair $(\{dk^{i+1*}\}, \{\hat{dk}^{i+1*}\}, \{ek^{i+1*}\})$, where $dk^{i+1*} := (\tilde{y}'^*, y'^*, Y'^*)$, $\hat{dk}^{i+1*} := (y'^*, Y'^*)$, $ek^{i+1*} := (\tilde{Y}'^*, Y'^*)$, $y'^* = \tilde{y}'^* \bmod p$, $Y'^* = g^{y'^*}$, $\tilde{Y}'^* = h^{\tilde{y}'^*} \bmod N$, on the input the decryption key dk^i, finally $\mathcal{A}_2$ outputs a proof P^* on dk^i and dk^{i+1*}. These outputs satisfy:

$$(dk^{i+1}, \hat{dk}^{i+1}, ek^{i+1}) \leftarrow \mathsf{KeyUpdate}(dk^i) \tag{6}$$

$$\hat{\sigma}^* \leftarrow \mathsf{EncSign}(sk, ek^{i+1*}, m^*) \tag{7}$$

$$\sigma \leftarrow \mathsf{DecSign}(\hat{dk}^{i+1}, \hat{\sigma}^*) \tag{8}$$

$$1 \leftarrow \mathsf{ConsVerify}(ek^i, ek^{i+1*}, P^*) \tag{9}$$

$$0 \leftarrow \mathsf{Verify}(pk, m^*, \sigma) \tag{10}$$

where $dk^{i+1} := (\tilde{y}', y', Y')$, $\hat{dk}^{i+1} := (y', Y')$, $ek^{i+1} := (\tilde{Y}', Y')$, $\hat{\sigma}^* := (\hat{R}^*, \hat{s}^*)$, $\sigma := (R, s)$, and m^* is an arbitrary message.

We can imply that $\tilde{y}' = \tilde{y}^2 \bmod N, y' = \tilde{y}' \bmod p, Y' = g^{y'}$; **(from 6)**. Let $\hat{R}^* = g^{\hat{r}^*}$, we can also imply that $R^* = \hat{R}^*Y'^*, c^* = H(m^*||R^*||X), \hat{s}^* = (\hat{r}^* + c^*x) \bmod p$, **(from 7)**, $R = \hat{R}^*Y'$, $s = (\hat{s}^* + y') \bmod p$ **(from 8)**, $1 \leftarrow \mathsf{NIZK.Verify}(ek^i, ek^{i+1*}, P^*)$ **(from 9)** and $c = H(m^*||R||X)$, $R \neq g^sX^{-c}$ **(from 10)**.

From the above equations, the following always hold: $R = \hat{R}^*Y' \neq g^sY^{-c} \Rightarrow \hat{R}^*Y' \neq g^{\hat{s}^*+y'}Y^{-c} \Rightarrow \hat{r}^* + y' \neq \hat{s}^* + y' - cx \Rightarrow \hat{r}^* \neq (\hat{r}^* + c^*x) \bmod p - cx \Rightarrow cx \neq c^*x \bmod p \Rightarrow c \neq c^* \Rightarrow H(m^*||R||X) \neq H(m^*||R^*||X)$. Remark that, for the same input, the hash function $H()$ produces a same output, otherwise, it yields the different output. Thus, if $c \neq c^*$, it implies that $m^*||R||X \neq m^*||R^*||X$. We then have: $R \neq R^* \Rightarrow \hat{R}^*Y' \neq \hat{R}^*Y'^* \Rightarrow Y' \neq Y'^* \Rightarrow y' \neq y'^* \Rightarrow (\tilde{y}^2 \bmod N) \bmod p = \tilde{y}' \bmod p \neq \tilde{y}'^* \bmod p$, which implies that, $\tilde{y}'^* \neq \tilde{y}^2 \bmod N$.

According to the above inference, for the NIZK instance (χ, w), where $\chi := (\tilde{Y}, \tilde{Y}'^*, Y, Y'^*)$ and $w := (\tilde{y}, \tilde{y}'^*, y, y'^*, k)$ and the relation R, where R is defined as: $R = PoK_{CVES} = \{(\tilde{y}, \tilde{y}'^*, y, y'^*, k) : \tilde{Y} = h^{\tilde{y}} \bmod N \wedge \tilde{Y}'^* = h^{\tilde{y}'^*} \bmod N \wedge \tilde{Y}'(h^N)^k = \tilde{Y}^{\tilde{y}} \bmod N \wedge Y'^* = g^{\tilde{y}'^*} \wedge 0 < \tilde{y}'^* < N\}$, which implies that $\tilde{y}'^* = \tilde{y}^2 - kN$ or $(\tilde{y}'^* = \tilde{y}^2 \bmod N)$, there exist a proof P^* satisfies the equation $\mathsf{NIZK.Verify}(\chi, P^*) = 1$, where $(\chi, w) \notin R$, and

$$Pr[\mathsf{NIZK.Verify}(\chi, P^*) = 1] > negl(\xi),$$

It is contradict to the NIZK soundness [8, 19]. That is $\mathcal{S}_5$ can output a proof P^* on the chosen NIZK instance (χ, w), which breaks the soundness property of the NIZK system. □

According to Lemma A.1, A.2, A.3, A.4, Theorem 1 is proved.

B INSTANTIATION AND IMPLEMENTATION OF SCHNORR-BASED CVES

B.1 Instantiation

We give the instantiation of the algorithms $\mathsf{KeyUpdate}(\cdot)$, $\mathsf{ConsVerify}(\cdot)$ and $\mathsf{ConsProof}(\cdot)$ (in Figure 2). For simplicity, we set $n = N = p'q'$, and let $dk^i := (u_i, t_i, T_i)$ and $ek^i = (U_i, T_i)$ in the following proof, where i is the session number. Let $u_i \in [0, N-1]$ and (u_{i-1}, u_i) satisfy: $u_i = u_{i-1}^2 \bmod N$.

Observe that $U_{i-1}, U_i, T_{i-1}, T_i$ are public elements. We denote PoK_{CVES} the proof-of-knowledge for CVES scheme. Then we construct a zero-knowledge proof-of-knowledge system for u_{i-1},

u_i, t_{i-1}, t_i and k, which can be abstracted as follow:

$$PoK^1_{CVES} = \{(u_{i-1}, u_i, t_{i-1}, t_i, k) : U_{i-1} = h^{u_{i-1}} \bmod N \wedge$$
$$U_i = h^{u_i} \bmod N \wedge U_i(h^N)^k = U_{i-1}^{u_{i-1}} \bmod N \wedge T_i = g^{u_i}\}.$$

PoK^1_{CVES} implies that $u_i = u_{i-1}^2 - kN$ (or $u_i = u_{i-1}^2 \bmod N$) and $t_i = u_i \bmod p$.

Also, we need to prove u_i is from 0 to $N-1$. This can be done with any range proof technique. Thus, we have the second part for PoK_{CVES}:

$$PoK^2_{CVES} = \{(u_i) : U_i = h^{u_i} \bmod N \wedge 0 < u_i < N\}, \tag{11}$$

The construction of PoK_{CVES} has two parts as described above. They are constructed separately, but can be executed in parallel in its actual implementation. First, we present the proof system for PoK^1_{CVES}. Let λ_3, λ_4 be some security parameters which determine the soundness and (statistical) zero-knowledgeness of the proof. Define a cryptographic hash function $H : \{0,1\}^* \to \{0,1\}^{\lambda_3}$. The first part of the zero-knowledge proof is constructed as follows:

- (*Commitment*). In a session i, the prover chooses $\rho_{u_{i-1}}, \rho_{u_i}, \rho_k$ uniformly at random from $[0, 2^{|N|+\lambda_3+\lambda_4-1}]$, and computes four commitments:
  $Y_{i,1} = g^{\rho_{u_i}}$, $Y_{i,2} = h^{\rho_{u_{i-1}}} \bmod N$, $Y_{i,3} = h^{\rho_{u_i}} \bmod N$, and $Y_{i,4} = U_{i-1}^{\rho_{u_{i-1}}}(h^{-N})^{\rho_k} \bmod N$.
- (*Challenge*). The verifier picks a λ_3-bit challenge uniformly at random. In the non-interactive form, the λ_3-bit challenge can be computed as the hash of the commitments and the public parameters, i.e, $\omega_i = H(g, h, T_{i-1}, T_i, U_{i-1}, U_i, Y_{i,1}, Y_{i,2}, Y_{i,3}, Y_{i,4})$.
- (*Response*). The prover computes the response: $(z_{u_{i-1}} = \rho_{u_{i-1}} - \omega_i \cdot u_{i-1}, z_{u_i} = \rho_{u_i} - \omega_i \cdot u_i, z_k = \rho_k - \omega_i \cdot k)$ as the proof.
- (*Verify*). The verifier outputs 1 if it holds that:
  $$Y_{i,1} = h^{z_{u_{i-1}}} U_{i-1}^{\omega_i} \bmod N; Y_{i,2} = g^{z_{u_i}} T_i^{\omega_i};$$

$$Y_{i,3} = h^{z_{u_i}} U_i^{\omega_i} \bmod N; Y_{i,4} = (h^{-N})^{z_k} U_{i-1}^{z_{u_{i-1}}} U_i^{\omega_i} \bmod N.$$

Completeness: if $z_{u_{i-1}} = \rho_{u_{i-1}} - \omega_i \cdot u_{i-1}$, $z_{u_i} = \rho_{u_i} - \omega_i \cdot u_i$, and $z_k = \rho_k - \omega_i \cdot k$, then $Y_{i,1} = g^{\rho_{u_i}} = g^{z_{u_i}} T_i^{\omega_i}$, $Y_{i,2} = h^{\rho_{u_{i-1}}} \bmod N = h^{z_{u_{i-1}}} U_{i-1}^{\omega_i} \bmod N$, $Y_{i,3} = h^{\rho_{u_i}} \bmod N = h^{z_{u_i}} U_i^{\omega_i} \bmod N$ and $Y_{i,4} = U_{i-1}^{\rho_{u_{i-1}}}(h^{-N})^{\rho_k} \bmod N = U_{i-1}^{z_{u_{i-1}}} U_i{}^{\omega_i}(h^{-N})^{z_k} \bmod N$.

Zero Knowledge: for every $g \in \mathbb{G}_p$ and $h \in \mathbb{Z}_N^*$, let $\mathcal{P}$ be the proof and $\mathcal{V}$ be the verify algorithm, the output of the simulator needs to be indistinguishable from the distribution of the transcripts:

$$\begin{aligned}
&View_V(\mathcal{P}(u_{i-1}, u_i) \leftrightarrow \quad \mathcal{V}(g, h, U_{i-1}, U_i, T_{i-1}, T_i)) \\
&= \{(g^{\rho_{u_i}}, h^{\rho_{u_{i-1}}} \bmod N, h^{\rho_{u_i}} \bmod N, U_{i-1}^{\rho_{u_{i-1}}}(h^{-N})^{\rho_k}, \\
&\qquad \omega_i, \rho_{u_{i-1}} - \omega_i \cdot u_{i-1}, \rho_{u_i} - \omega_i \cdot u_i, \rho_k - \omega_i \cdot k) : \\
&\quad \omega_i = H(g, h, U_{i-1}, U_i, T_{i-1}, T_i, Y_{i,1}, Y_{i,2}, Y_{i,3}, Y_{i,4}), \\
&\quad (\rho_{u_{i-1}}, \rho_{u_i}, \rho_k) \leftarrow [0, 2^{|N|+\lambda_3+\lambda_4} - 1]\} \\
&= \{(Y_i, Y_{i,2}, Y_{i,3}, Y_{i,4}, \omega_i, z_{u_{i-1}}, z_{u_i}, z_k) : \\
&Y_{i,1} = g^{z_{u_i}} T_i^{\omega_i}, Y_{i,2} = h^{z_{u_{i-1}}} U_{i-1}^{\omega_i} \bmod N, \\
&Y_{i,3} = h^{z_{u_i}} U_i^{\omega_i} \bmod N, Y_{i,4} = U_{i-1}^{z_{u_{i-1}}} U_i{}^{\omega_i}(h^{-N})^{z_k} \bmod N\}
\end{aligned} \tag{12}$$

We construct a simulator S that outputs the same distribution by running the protocol "in reverse":

```
$S(g, h, U_{i-1}, U_i, T_{i-1}, T_i)$:
$z_{u_{i-1}}, z_{u_i}, z_k \leftarrow [0, 2^{|N|+\lambda_3+\lambda_4} - 1]$;
$\omega_i \leftarrow [0, 2^{\lambda_3} - 1]$;
$Y_{i,1} \leftarrow g^{z_{u_i}} T_i^{\omega_i}$;
$Y_{i,2} \leftarrow h^{z_{u_{i-1}}} U_{i-1}^{\omega_i} \bmod N$
$Y_{i,3} \leftarrow h^{z_{u_i}} U_i^{\omega_i} \bmod N$
$Y_{i,4} \leftarrow U_{i-1}^{z_{u_{i-1}}} U_i{}^{\omega_i}(h^{-N})^{z_k} \bmod N$
```

Since ω_i is chosen from at random $[0, 2^{\lambda_3}]$, and $z_{u_{i-1}}, z_{u_i}$ and z_k are chosen at random from $[0, 2^{|N|+\lambda_3+\lambda_4}]$, then the resulting $Y_{i,1}$, $Y_{i,2}$, $Y_{i,3}$ and $Y_{i,4}$ are random, and the output is distributed statistically close to the real transcript.

Soundness: Let I^* be a (possible malicious) prover that convinces the verifier with probability $1 - negl(\xi')$, where $negl()$ is a negligible probability with ξ'-bit inputs. We construct the extractor as follows:

```
$\epsilon^{P^*}(h)$:
1: Run the prover $I^*$ to obtain an initial message tuple $(Y_{i,1}, Y_{i,2}, Y_{i,3}, Y_{i,4})$ and compute the first challenge $\omega_{i,a}$ using the two tuple. Then we have a response tuple $(z^a_{u_{i-1}}, z^a_{u_i}, z^a_k) \leftarrow [0, 2^{|N|+\lambda_3+\lambda_4} - 1]$;
2: Rewind the prover $I^*$ to its state after the first message;
3: Run the prover $I^*$ to obtain the second challenge $\omega_{i,b}$ by using $(Y_{i,1}, Y_{i,2}, Y_{i,3}, Y_{i,4})$, and get second response tuple $(z^b_{u_{i-1}}, z^b_{u_i}, z^b_k)$;
4: Compute and output $u_i = \frac{z^a_{u_i} - z^b_{u_i}}{\omega_{i,b} - \omega_{i,a}}$ $(in\ \mathbb{Z})$, $u_{i-1} = \frac{z^a_{u_{i-1}} - z^b_{u_{i-1}}}{\omega_{i,b} - \omega_{i,a}}$ $(in\ \mathbb{Z})$.
It remains to argue that $\omega_{i,b} - \omega_{i,a} | z^a_{u_i} - z^b_{u_i}$ (resp. $\omega_{i,b} - \omega_{i,a} | z^a_{u_{i-1}} - z^b_{u_{i-1}}$) so that the extractor can compute this without knowing $\phi(N)$.
```

We sketch the proof that $\omega_{i,b} - \omega_{i,a} | z^a_{u_i} - z^b_{u_i}$ under the strong RSA assumption. First, we have $Y_{i,3} = h^{z^a_{u_i}} U_i^{\omega_{i,a}} \bmod N$ and $Y_{i,3} = h^{z^b_{u_i}} U_i^{\omega_{i,b}} \bmod N$. Thus, we have $h^{z^a_{u_i}} U_i^{\omega_{i,a}} = h^{z^b_{u_i}} U_i^{\omega_{i,b}} (\bmod N)$ Denote by α and β the values $z^a_{u_i} - z^b_{u_i}$ and $\omega_{i,b} - \omega_{i,a}$, thus, we have $h^\alpha = U_i^\beta (\mathrm{mod} N)$.

Suppose on a contrary, β does not divide α. Let $f = \gcd(\alpha, \beta)$ and $\alpha = f\alpha'$ and $\beta = f\beta'$ for some $\alpha', \beta' \neq 1$.

This implies $h^{\alpha'} = U_i^{\beta'} (\mathrm{mod} N)$ (otherwise we can setup N as a product of two safe-primes and f would help factorising N. In other words, we should choose N such that it is a product of two safe primes.)

Since $\gcd(\alpha', \beta') = 1$, there exists integers X, Y such that $X\alpha' + Y\beta' = 1$. Therefore, $h = h^{X\alpha' + Y\beta'} (\mathrm{mod} N) = (U_i^X h^Y)^{\beta'} (\mathrm{mod} N)$.

Therefore, $(U_i^X h^Y), \beta')$ is a solution to the strong RSA problem on (N, h). In other words, if $\omega_{i,b} - \omega_{i,a} | z^a_{u_i} - z^b_{u_i}$, we can solve the strong RSA problem.

The proof that $\omega_{i,b} - \omega_{i,a} | z^a_{u_{i-1}} - z^b_{u_{i-1}}$ is exactly the same and is thus omitted.

The extraction then fails if $z^a_{u_{i-1}} = z^b_{u_{i-1}}$ or $z^a_{u_i} = z^b_{u_i}$ which happens with probability $\frac{1}{2^{\lambda_3}}$. Therefore, the knowledge error here is $negl(\xi') = \frac{1}{2^{|N|+\lambda_3+\lambda_4}}$.

The second part PoK^2_{CVES} requires that at each session, the prover proves that his/her updated decryption key $u_{\mathcal{P},i}$ lies in the interval $[0, N-1]$. This proof can be done by various existing range proof systems, for example [10–12, 14]. As the second part PoK^2_{CVES} is not the contribution of the paper, we omit the inistantiation of proving $u_{\mathcal{P},i}$ lies in the interval $[0, N-1]$.