MONASH University

Payment Channel Network for Scriptless Blockchain

Doctor of Philosophy Degree

By ***Zhimei Sui***

Supervisor: Dr. Jiangshan Yu

Co-supervisor: Prof. Joseph K. Liu

A thesis submitted for the degree of *Doctor of Philosophy*
at Monash University in *2023*,
Faculty of Information Technology,
Department of Software Systems & Cybersecurity

Copyright notice

Abstract

Payment channels are considered a promising solution to address blockchain scalability, but their development for scriptless blockchains like Monero has presented significant challenges. One of the open challenges is enabling bidirectional payments on scriptless blockchains.

Therefore, this work aims to answer the question of how bi-directional payment channels can be developed and implemented for scriptless blockchains, specifically Monero, to effectively address the scalability challenges.

We introduce AuxChannel, the first bi-directional payment channel protocol without limited lifetime for scriptless blockchains. This means that building payment channels only requires the support of verifiably encrypted signatures, also known as adaptor signatures, on the underlying blockchain.

To cater specifically to Monero, we adapt AuxChannel and name it MoChannel. Additionally, we introduce MoNet, a multi-hop payment framework built on top of MoChannel, designed to ensure anonymity and fungibility for the underlying blockchain.

Furthermore, our proposed solutions require a distributed third party to effectively resolve potential channel disputes. To facilitate this, we propose FPSS, a flexible proactive secret share scheme, which enables the provision of such a distributed service.

Finally, to enable efficient construction of payment channels and multi-hop payments over scriptless blockchains, we introduce several primitives, including Consecutive Verifiably Encrypted Signature (CVES), Verifiable Consecutive One-Way Function (VCOF), Generalized Consecutive Adaptor Signature (GCAS), and FPSS. These versatile building blocks can be applied to other applications beyond payment channels.

Declaration

This declaration is to be included in a standard thesis. Students should reproduce this section in their thesis verbatim.

This thesis is an original work of my research and contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Signature: ……

Print Name: Zhimei Sui

Date: 11 May 2023

Publications during enrolment

The following papers containing contents in this thesis have been published.

- Zhimei Sui, Joseph K. Liu, Jiangshan Yu, Man Ho Au, Jia Liu. "AuxChannel: Enabling Efficient Bi-Directional Channel for Scriptless Blockchains." AsiaCCS 2022: 138-152 (Core Ranking A). https://dl.acm.org/doi/10.1145/3488932.3524126.

- Zhimei Sui, Joseph K. Liu, Jiangshan Yu, Xianrui Qin. "MoNet: A Fast Payment Channel Network for Scriptless Cryptocurrency Monero." ICDCS 2022: 280-290 (Core Ranking A). https://ieeexplore.ieee.org/document/9912301.

The following paper(s) contained in this thesis is in submission.

- Zhimei Sui, Joseph K. Liu, Jiangshan Yu. "FPSS: Flexible Proactive Secret Share Scheme".

The following publications are related but not included in this thesis:

- Runchao Han, Zhimei Sui, Jiangshan Yu, Joseph K. Liu, Shiping Chen. "Fact and Fiction: Challenging the Honest Majority Assumption of Permissionless Blockchains." AsiaCCS 2021: 817-831 (Core Ranking A). https://dl.acm.org/doi/10.1145/3433210.3453087.

- Xianrui Qin, Shimin Pan, Arash Mirzaei, Zhimei Sui, Oguzhan Ersoy, Amin Sakzad, Muhammed F. Esgin, Joseph K. Liu, Jiangshan Yu, Tsz Hon Yuen: BlindHub: Bitcoin-Compatible Privacy-Preserving Payment Channel Hubs Supporting Variable Amounts. IEEE S&P 2023 (Core Ranking A□). https://www.computer.org/csdl/proceedings-article/sp/2023/933600c020/1Js0EznDFew

Acknowledgements

I express my heartfelt gratitude to my supervisors, Dr. Jiangshan Yu and Prof. Joseph K. Liu, for providing me with the invaluable opportunity to pursue my Ph.D. studies and for their guidance, support, and understanding throughout my research journey. Their constant encouragement, insightful feedback, and expertise have been instrumental in shaping my academic career and personal growth.

My panel members, Dr. Amin Sakzad and Assoc. Prof. Shifeng Sun, as well as my Chair, Assoc. Prof. Ron Steinfeld, also deserve my gratitude for their insightful comments, feedback, and guidance during my Ph.D. journey. Their expertise and support have been instrumental in shaping my research and academic development.

Furthermore, I extend my appreciation to Monash University for providing financial support that enabled me to focus on my research and achieve my academic goals. I am also grateful to Dr. Jia Liu and Prof. Man Ho Allen Au for their assistance in deepening my understanding of cryptography, and to my co-authors, Xianrui Qin, Dr. Runchao Han, Dr. Shiping Chen, and Dr. Muhammed Esgin, for their invaluable contributions to my research.

Finally, I am deeply grateful for the unwavering love, understanding, and companionship provided by my family, including my parents, relatives, and partner, throughout my Ph.D. journey. Their support and encouragement have been a constant source of inspiration and motivation, and I could not have made it this far without them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Payment channel 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 the 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 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.

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, 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 a 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 channels, as a payer performs incremental payment to a payee (by creating a transaction signed by the payer), the payee always has the incentive to close the channel with the latest state (where the payee has the largest balance) by cross-signing the received payment. However, it is possible that the payee is not responding the request to close the channel. 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) de nes a timer, such that upon timeout the payer will get a full refund of its channel balance unless the payee has reacted to it by closing the channel before the timeout.

However, time lock is not su cient to resolve disputes 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 nal 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 nal 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, such as Monero [1] and ZCash [2], and therefore cannot implement such dispute mechanism.

We call such blockchains *scriptless blockchains*. Scriptless blockchains are well known to the blockchain community and also represent billions of dollars. For example, Monero is the top one privacy-preserving cryptocurrency with a market cap of over 2.6 Billion US dollars [1]. However, the scalability of Monero is still an open challenge in the blockchain community. The only deployed effort so far is the adjustable block size in Monero; however, increasing the block size limit has caused controversy. Despite the current research effort on payment channels, designing a bi-directional payment channel for scriptless blockchains remains an open challenge.

We then conclude the specific research questions that this thesis aims to solve as follows:

1. Q1: How can a bi-directional payment channel with an infinite lifespan be introduced for scriptless blockchains?

2. Q2: How can the payment channel mentioned in Q1 be used in Monero, and how to build a payment channel network on top of this adapted payment channel, enabling multi-hop payments while ensuring anonymity and fungibility for the underlying blockchain?

3. Q3: How can potential channel disputes be effectively resolved?

We solve the first research question, Q1, by presenting an innovative solution called AuxChannel, which introduces a bi-directional payment channel with an infinite lifespan for scriptless blockchains.

Next, we answer the second research question, Q2, by adapting AuxChannel to be used in Monero, named MoChannel. Additionally, we also propose MoNet, a framework built on top of MoChannel, that enables multi-hop payments. We

[1] Data collected on 16 March 2023 from https://coinmarketcap.com/.

demonstrate that MoNet ensures anonymity and fungibility for the underlying blockchain.

Finally, to resolve any potential channel disputes, which is the third research question Q3, we leverage a distributed third-party service. This is achieved by using the proactive secret share scheme, a well-studied cryptographic primitive. The secret share scheme allows a secret owner to distribute a secret among n parties, called a committee, any $t + 1$ subset of the n parties can reconstruct the secret.

We propose FPSS, a exible secret share scheme. It adds an additional phase, F-HandO , to enable e ective membership updates. By using FPSS, an update requires only the interaction of pairs of shareholders who are leaving and joining. FPSS enables the provision of such a distributed service required in channel disputes.

1.1 Contributions

This thesis contains the following merits:

1. We propose **AuxChannel** in enabling e cient bi-directional payment channel for scriptless blockchain with unlimited lifespan.

    - AuxChannel guarantees that *neither parties can prevent the channel from closing or stealing 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. We leverage a newly introduced cryptographic primitive, called CVES (see below), to minimise

the involvement of the script-enabled blockchain, i.e., with communication complexity O(1). The introduced cost of using a second chain is negligible in comparison to the saved transaction fees.

- 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 veri able, and more importantly the encryption-decryption key-pairs are consecutive, which means the latest key-pairs are derived from their previous session (consecutive). A veri er 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 ow. (We remark that CVES can be also regarded as the *consecutive* version of adaptor signature [3], in which a new \adapted secret" can be generated in a veri able 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. We propose **MoChannel** to enable bi-directional payment channels (with no limit on the lifetime) between both channel parties and build our **MoNet** on top of the MoChannel to enable multi-hop payments. Both MoChannel and MoNet also preserve the anonymity and fungibility of Monero.

    - **Verifiable Consecutive One-way function (VCOF) and Generic Consecutive Adaptor Signature (CAS).** We propose the veri able consecutive one-way function (VCOF), which generates

a sequence of statement-witness pairs in a single direction, and provides the one-wayness property, where it is easy to compute on the input, but hard to invert given the image of a random input, can be publicly veri ed. We further construct the generic construction of consecutive adaptor signature (CAS) by using VCOF, which is an important building block of MoNet.

- **MoNet**. We construct an e cient payment channel network, MoNet, for Monero. In particular, we propose MoChannel to enable bi-directional payment channels (with no limit on the lifetime) between both channel parties, and build our MoNet on top of the MoChannel to enable multi-hop payments. Both MoChannel and MoNet also preserve the anonymity and fungibility of Monero. We also construct the security model for MoNet, and prove that MoNet is UC-secure.

- **Implementation and Optimization**. We develop and evaluate a proof-of-concept implementation of MoNet. Our evaluation shows that MoChannel enables Monero to process approximately $2{:}34D$ transactions per second (TPS), where D is the number of established channel on Monero. For example, as of Jan 2022, there are more than $80{;}000$ channels in Lightning Network for Bitcoin. If MoChannel is of the same scale, then it can support Monero to process over $180{;}000$ transactions per second. We further optimize MoChannel's performance by using pre-computation to calculate and verify a batch of statement-witness pairs, improving the throughput to approximately $1{;}100{;}000$ TPS, which reaches the same throughput level of lightning network[1]. In terms of multi-hop payment, MoNet can process a multi-hop pay-

[1]Data collected from https://medium.com/coinmonks/how-does-bitcoin-get-scalable-with-the-lightning-network-63591040462c

ment within 68:68ms n_h via n_h hops with 4G WAN latency of 60ms.

3. Our proposed solutions necessitate a distributed third party to e ectively resolve potential channel disputes. To facilitate this, we propose FPSS, a exible proactive secret share scheme, which enables the provision of such a distributed service. It adds an additional phase, F-HandO , to enable e ective membership updates. F-HandO requires only the interaction of pairs of shareholders who are leaving and joining. The core innovations of F-HandO are: 1) F-HandO adds *exibility* for shareholders to hand their shares o to another newly joint one. 2) when multiple shareholders update their shares concurrently, only the pair of the leaving and joining shareholders needs to be interactively involved. The hand-o of F-HandO has two cases, *single F-HandO* and *batched F-HandO* .

   - *Single F-HandO* . It allows a shareholder to hand-o his share to a new shareholder. The single F-HandO involves only two shareholders, the one who is leaving and the other who is joining. The single F-HandO can be executed by any shareholder at any time without interactively communicating with the remaining committee.
   - *Batched F-HandO* . It allows multiple shareholders to leave at the same time without interacting with each other. Leaving shareholders can be any subset of the committee. Communication is only required between the pair of leaving and joining shareholders. Thus, the single F-HandO cannot be directly used in the batch F-HandO , since each involved shareholder should have a consensus on the new committee members.

Roughly speaking, FPSS employs a blockchain to verify each F-HandO request and as the public accessible storage. The involved shareholders also broadcast

messages o -chain to each shareholder in the old and new committees, respectively. Since involved shareholders will communicate via blockchain, any malicious behavior can be detected. The underlying system can incentivize the honest and punish the malicious. While such an incentive mechanism is beyond the scope of this thesis, we will not discuss it further. Notice that the two cases of F-HandO are not tiered. Namely, they are not countermeasures to each other when the fault is detected, but rather the two functionalities for di erent use cases.

1.2 Related Work

This thesis focuses on the payment channel scheme and the channel dispute solution for scriptless blockchain. Furthermore, we consider using proactive secret share schemes to resolve the potential channel dispute. We then summarize some works related to payment channel schemes and proactive secret share schemes below.

Existing payment channel protocols for scriptless blockchain. While providing bi-directional payment channels on scriptless blockchains remains unsolved, ways to provide unidirectional payment channels 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 [4] and Z-Channel [5]) and timed commitment (e.g. PayMo [6]), as all of them provide the time-lock function without requiring the support of script language.

Both the DLSAG channel for Monero and Z-Channel for Zerocash use time proof as a veri able commitment for a prede ned timelock. This timelock is included in the input of a transaction, and the output of the transaction can only

be spent after the prede ned time has passed. To avoid repetition, we will use the DLSAG channel as an example to demonstrate how the time commitment works in a payment channel.

DLSAG channel makes it possible for Monero to support payment channels, while it has a limited lifespan which means a channel has to be closed within a prede ned 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 quicky (even before the prede ned 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 veri cation. Thus it makes it di cult for being adopted in 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-speci ed time. PayMo has extra computation requirement for opening the timed commitment, while it can be mitigated by a third-party service [7]. Thus PayMo is fully compatible with the transaction in Monero. However, it still has limited lifespan and supports one-way payment only.

Exisiting bi-directional payment channel protocols for blockchains with limited scripts. Bolt [8] is a bi-directional payment channel hub protocol designed to address linkability issues in channels, and it can be built on ZCash. To deploy Bolt, it is necessary to verify zero-knowledge proofs on the underlying blockchains, such as Monero. Aumayr et al. [9] have formalized the generalized payment channel for blockchains with limited scripts. This method requires the underlying blockchain to support the adaptor signature scheme, relative time-

lock, and a constant number of Boolean operations. Another approach, Sleepy Channels [10], supports bi-directional payment channels with a limited lifespan and also requires on-chain time-lock support. If running Sleepy Channels between untrusted parties, additional collateral is needed, which may be equal to the channel capacity for each channel party, to ensure secure and quick channel closure. AuxChannel focuses on building bi-directional payment channels with an unlimited lifespan for scriptless blockchains. It allows building payment channels by only requiring the underlying blockchain to support veri ably encrypted signatures (also known as adaptor signatures).

Other related works of payment channel schemes. In 2013, Satoshi Nakamoto, the author and creator of the Bitcoin whitepaper [11] and project, described a technique for high-frequency transactions in a personal email. This technique allowed participants to close the channel with any intermediate state and double-spend the o -chain funding input. However, according to its design, this technique only worked under the strong assumption that all participants followed the protocol honestly. Despite its disadvantages, this idea is considered the prototype of payment channel schemes and the rst o -chain transaction idea on blockchain.

In the same year, the Spilman-styled payment channel [12] was proposed to ll the insecure design in Satoshi's high-frequency transactions technique idea. This protocol allowed for secure two-party o -chain trades without relying on the honest players assumption. However, it only allowed for uni-directional coin transfer with limited channel lifetime, which increased the cost of creating a payment channel and was not desirable in real payment scenarios.

To address this limitation, the Duplex Channel [13] was introduced, which consisted of two opposite-directional Spilman-styled payment channels between two channel parties. This approach was straightforward and inherited the design

of limited channel lifetime. However, there is still a need for deeper research into bi-directional payment channel protocol design.

In 2016, the Poon-Dryja channel [14] was introduced, which realized an untrusted bi-directional payment channel without a limited lifetime on Bitcoin. This protocol became the basis for the well-known payment channel network project, Lightning Network. The core technique of Poon-Dryja channel, Hash-Time Lock Contract (HTLC), is widely adopted in many protocols, making it useful among on-chain, o -chain, and cross-chain applications. However, it requires a programmable language supported by its underlying blockchain.

After the Poon-Dryja channel, two other payment channel protocols were introduced, Eltoo [15] and Generalized Bitcoin-Compatible Channels [9], which are also based on the HTLC technique. These advancements have contributed to the development of secure and e cient payment channel networks that have the potential to improve blockchain scalability and transaction speed.

One of the cross-chain coin swap protocols is atomic swap, which also utilizes HTLC technique as its core building block. However, due to its requirements of a programmable language, atomic swap faces the same challenges as scriptless blockchains. To address this issue, the BTC-XMR atomic swaps protocol [16] has been developed, which enables coin transfers between a script-enabled blockchain and a scriptless blockchain. It requires time-lock function only on the Bitcoin side. Another recent cross-chain coin swap protocol for a scriptless blockchain is JugglingSwap [17], which introduces a trusted third party to control the time of secret releasing. However, unlike JugglingSwap, AuxChannel does not require the involvement of a trusted third party for each o -chain payment by using the CVES scheme.

The anonymous multi-hop locks (AMHLs) [18], which aim to lock several atomic transactions among several users in a payment path, are similar to our

proposed CVES. However, AMHLs require all the on-path users' participation to derive all the decryption keys, and the number of decryption keys is xed at the setup phase. Therefore, under the bi-directional payment channel scenario, AMHLs cannot prevent a malicious participant from locking the channel and also su er from a limited life cycle. The functionality of AMHLs does not meet the requirements of AuxChannel, while CVES does.

Secret share schemes on Blockchain. In recent years, Blockchain technology has gained a lot of attention for its permissionless, public accessibility, and immutability functionalities. Many researchers are adopting the blockchain as the underlying infrastructure to improve their techniques, including dynamic secret share schemes [19; 20; 21]. For example, instead of sending a message directly to a receiver, some works consider using the blockchain as a communication channel. This means that once a participant in the scheme posts their message to the blockchain, everyone in the network can receive it. Other works use the blockchain for player replacement [19; 20].

Benhamouda et al. [19] use PoS blockchain to construct the committee selection scheme and the anonymous public key encryption primitive. Each member of the new committee is anonymous to the old committee, and their player selection will incur on-chain cost. This scheme can tolerate at most 1/4 malicious parties under the mobile adversary setting. Goyal et al. [20] enable the dynamic secret share scheme to be deployed on either PoS or PoW blockchain and can tolerate at most 1/2 malicious parties under the adaptive adversary model. This scheme requires that all members from the old and new committees participate in the hand-o phase.

CHURP [21] uses the bivariant polynomial to share secrets, and they introduce a new proactivization protocol with 1/2 resilience. They prefer to use blockchain as the communication channel, or o -chain as an alternative, due to the high cost

of writing to the blockchain. All participants in their protocol should participate in the hand-o phase. CALYPSO [22] introduces a data management system that enables data owners to manage their data access control rights via a secret share scheme, while among a static committee.

Overall, blockchain technology has shown great potential for improving dynamic secret share schemes, allowing for greater security, privacy, and e ciency in data management and player replacement.

Chapter 2

Preliminary

This Chapter presents some preliminary knowledge of this thesis.

2.1 Blockchain

Permissionless Blockchain is a distributed, tamper-evident, and public accessible ledger. We assume that once the blockchain network receives a transaction, it will be recorded on-chain within time interval. A on-chain transaction will never be modi ed. The new generated block will be sent to the network and received by other nodes in the network within $_b$ time interval. For the sake of security, the adopted blockchain satis es the properties of (1) *persistence* | if a transaction *tx* is con rmed by an honest party, no honest party will ever disagree about the position of *tx* in the ledger, and (2) *liveness* | if a valid transaction *tx* is broadcast, it will eventually become con rmed by all honest parties.

2.2 Layer 2 Solutions for Scalability

Blockchain is the distributed ledger technology, and the unchangeability is the main feature. However, the scalability is hindered to satisfy the characteristic. Nowadays, there are more and more attempts to put many applications on blockchains, but there are problems in actual use, for example, the low number of processing transactions per second (tps).

Brie y, protocols that perform with minimal interaction with the blockchain. Typically, for opening, paying, closing and disputing a channel. Two major examples of layer 2 solutions are the Bitcoin Lightning Network [14] and the Ethereum Plasma. Despite having their own working mechanisms and particularities, both solutions are striving to provide increasing throughout to blockchains systems. This layer encompasses much more protocols and algorithms than the channel construction alone, as we review in later sections

In this context, the term \layer 2" refers to the multiple solutions being proposed to the blockchain scalability problem. In a broader sense, layer 2 protocols create a secondary framework, where blockchain transactions and processes can take place independently of the layer 1 (main chain). For this reason, these techniques may also be referred to as \o -chain" scaling solutions.

According to Jourenko's work [23], systematization of knowledge for layer 2 protocols, they conclude the taxonomy and the levels in 3 levels as follows:

- *O* -chain Channels: Here are the constructions for the channels themselves. In other words, how the nodes relying on the transaction design of a cryptocurrency, can construct channels. Protocols in this level contains: Z-Channel[5], Perun [24], NoCUST [25], Raiden [26], Lightning [14], BOLT [8], Teechan [27] etc.

- *N*etwork: Here are the techniques employed to create the network them-

selves. Typically, by concatenating existing pairwise channels, or establishing a network of more than two nodes right from start. Protocols in this level contains: HTLC [14], Sprites [28], Virtual Channels [29] and Counterfactual [30], etc.

- *N*etwork Management: This the early mentioned family which embraces the protocol and algorithms that maintain the channels and network of channels. Typically they are responsible for keeping them \alive" and usable. Protocols in this level contains: Silent Whisper [31], Spider Routing [32], Atomic Multi-path [33], etc.

2.3 Payment Channel

In 2013, Satoshi Nakamoto, the author and creator of Bitcoin whitepaper and project, described a *high-frequency transactions technique* in a personal email [11]. This is the first o -chain transaction idea on blockchain with obvious disadvantages. In this protocol, there is no punishment and on-chain deposit designed, so participants can close the channel with any intermediate state and double-spend the o -chain funding input. According to its design, this technique only works on a strong assumption that all participants follow the protocol honestly. This email is considered as the prototype of payment channel schemes. At the same year, Spilman-styled style payment channel [12] was proposed to fill the insecure design in Satoshi's *high-frequency transactions technique* idea. This protocol makes a secure two-party o -chain trades without the honest players assumption. While this protocol only allows a unidirectional coin transfer and limited channel lifetime, this design increases the cost of creating a payment channel and is not desireable in real payment scenarios. A succeeding bidirectional channel scheme, *Duplex Channel* [13], is a simple attempt. The idea is making two opposite di-

rection spilman-styled payment channel between two channel users. This idea 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 [14] realizes untrusted bi-directional payment channel without limited lifetime on Bitcoin, it got a research breakthrough on payment channel. 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, o -chain and cross-chain applications, while one of the core design timelock in this technique 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* [34]. While these two protocols are also based on the HTLC technique.

As the requirement of supporting programmable language, the above payment channel protocols are not compatible with privacy-preserving blockchains. Then some payment channel schemes designed for the speci c privacy-preserving blockchains. Yuncong Zhang proposed the Z-channel [5] protocol for ZCash, and Pedro Moreno-Sanchez proposed DLSAG [4] for Monero. These two protocols have the similar solution, *time proof*, to realize the timelock technique on its underlying privacy-preserving blockchains. Time proof is a commitment contains in the input of a transaction, this is a prede ned timelock, which can be veri ed without uncover the time value. The output of in the transaction can only be spent after the prede ned time proof. While to support the time proof technique, a hard fork is required on the underlying blockchains. Also, these two protocols, Z-channel and DLSAG, are unidirectional, and the channel lifespan is limited.

Atomic swap is a technique widely used in cross-chain coin swap between two blockchains. This is another common scenario of using HTLC. According to its requirements of programmable language, atomic swap has the same issues for privacy-preserving blockchains. BTC-XMR atomic swaps [16] protocol proposed a coin swap protocol between script and scriptless blockchains. In this protocol, time lock only requires on Bitcoin side, on Monero side, there is a only one on-chain transaction. As this timelock scheme is not compatible with payment channel technique, we only give a brief introduction of this work. Another recent work, JugglingSwap [17], realized a more general timelock scheme in a cross-chain application. This work introduces a trusted third party to control the time of secret releasing. Comparing to this work, our work does not require a trusted third party to force the secret releasing, rather than a reward mechanism. The reward mechanism does not compromise the privacy of the channel users.

2.3.1 Anonymous Multi Hop Locks (AMHL)

Anonymous Multi Hop Locks (AMHL) proposed by Malavolta et al. [18] allows $n + 1$ channel parties to setup n payment locks, denoted by $l_1;::::;l_n$, in a path. AMHL guarantees that the lock l_i can be \unlocked" if and only if l_{i+1} is also released, where $i\ 2\ [1;::::;n\ \ 1]$. It enforces the atomicity of multi-hop payments and also o ers better on-chain privacy. A recent work [35] by Thyagarajan et al, constructs a LRS-compatible AMHL scheme.

2.4 Secret Share Scheme

Secret Share (SS). Assuming that a secret $s = f(0)$, where $f(\ )$ is a polynomial of degree t with random coe cients (aside from the constant term) over a nite eld. $f(i)$ is known by each node i, where $i \neq 0$ is the unique identi er for nodes.

Proactive Secret Share (PSS). PSS scheme allows a committee C who holds shares w.r.t to a secret to hand-o their shares to another committee C^0. The members in C and C^0 may not always be the same.

Security Definition. A secret share scheme usually should satisfy two properties, *robustness* and *secrecy*. We refer to the description of these two properties in the work by Goyal et al [20]. *Robustness* requires that it should always be possible to recover the secret. *Secrecy* requires that an adversary should not learn any further information about the secret beyond what has been learned before running the protocol.

2.5 Signature Scheme and Adaptor Signature Scheme

Signature scheme SIG allows a signer to authenticate messages, and adaptor signature $aSIG$ is a signature concealed by a secret, it allows a signer to pre-sign a message under her secret key, such that this pre-signature can be *adapted* into a valid signature for anyone who knows the pre-signature and the corresponding secret, or anyone who knows the pre-signature and signature can *extract* a secret from them. Figure 2.1 is the generic construction of digital signature, where

$\underline{\mathsf{Gen}_{sk}(\ )}$	$\underline{\mathsf{Sign}_{sk}(m)}$	$\underline{\mathsf{Vrfy}_{pk}(m;\ )}$
return $(sk; pk)$	$(R; st) \quad P_1(sk)$	parse $(h; s)$
	$h := \mathsf{H}_{SIG}(R; m)$	$R := V_0(pk; h; s)$
	$s := P_2(sk; R; h; st)$	return $h \stackrel{?}{=} \mathsf{H}_{SIG}(R; m)$
	$:= (h; s)$	
	return	

Figure 2.1: Generic Construction of Digital Signature

H_{SIG} is the challenge hash function, P_1 and P_2 are two algorithms employed by the signing algorithm producing the corresponding signature, and V_0 is the veri er

algorithm employed by the veri cation algorithm. Erwig et al. [36] further constructs a generic transformation from signature schemes into adaptor signatures schemes, as shown in Figure 2.2, where f_{shift}, f_{adapt} and f_{ext} are a randomness shift function, an adapt operation function, and a witness extraction function respectively.

$\mathsf{PSign}_{sk}(m; Y)$
$(R_{pre}; St) \quad P_1(sk);$
$R_{sign} := f_{shift}(R_{pre}; Y);$
$h := \mathsf{H}_{SIG}(R_{sign}; m)$
$\hat{s} \quad P_2(sk; R_{pre}; h; St)$
$\hat{} := (h; \hat{s})$
return $\hat{}$

$\mathsf{PVrfy}_{pk}(m; Y; \hat{})$
parse $(h; \hat{s}) \quad \hat{};$
$\hat{R}_{pre} := V_0(pk; h; \hat{s});$
$\hat{R}_{sign} := f_{shift}(\hat{R}_{pre}; Y);$
return $h \stackrel{?}{=} \mathsf{H}_{SIG}(\hat{R}_{sign}; m)$

$\mathsf{Adapt}_{pk}(\hat{}; y)$
parse $(h; \hat{s}) \quad \hat{};$
$s = f_{adapt}(\hat{s}; y);$
$:= (h; s)$
return ;

$\mathsf{Ext}_{pk}(\; ; \hat{}; Y)$
parse $(h; s) \quad ;$
parse $(h; \hat{s}) \quad \hat{};$
return $y := f_{ext}(s; \hat{s})$

Figure 2.2: Generic Adaptor Signature aSIG.

2.6 KZG Commitment

KZG commitment [37] allows a prover to generate a commitment to a private polynomial. Let F be a nite eld, s 2 F be a vector, where $s = (s_1; :::; s_n)$ satis es that there is a polynomial f 2 F[X] of degree at most t such that $f(0) = s$ and $f(i) = s_i$ for i 2 $f1; :::; ng$. Each party (shareholder) P_i holds a share s_i and the whole sharing is denoted as $[s]_t$. We present the four algorithms below and in Fig 2.3 that we will require in our protocol:

- $fhp; G; G_T; e; gi; cpkg \quad \mathsf{KZG.Setup}(t; 1)$: this algorithm initializes some public parameters, which will be used in the following commitment scheme. It inputs the degree bound t and the unary form of the security parameters, and outputs a commitment key pair $(csk; cpk)$ and a billinaer

group $(hp; G; G_T; e; g\,i)$. While the commitment secret key *csk* is safely destroyed, and the commitment public key will be used for the following calculation.

- KZG.cmt$_f$ KZG.Commit$(f(x); cpk)$: this algorithm generates a commitment for a polynomial. It inputs the polynomial $f(x)$ and the commitment public key *cpk*, and outputs a commitment KZG.cmt$_f$ on f.

- f cmt$_{f(i)}$; w$_{f(i)}$ g KZG.CrtWtn$(cpk; f(x); i)$: this algorithm generates the witness of the polynomial f on the variable i. It takes and the commitment public key *cpk*, the polynomial f, and the variable i as inputs, and outputs a witness $w_{f(i)}$ and the commitment of $f(i)$.

- 0=1 KZG.VrfyEval$(cpk;$ KZG.cmt$_f; i; f(i); w_{f(i)})$: this algorithm veri es the commitment on $f(i)$. It takes the commitment public key *cpk*, the polynomial commitment KZG.cmt$_f$, the variable i, the commitment of $f(i)$, and the witness $w_f(i)$ as inputs, and outputs 0 if the evaluation is correct, or 1 otherwise.

Remark. We input $C_{f(i)} = g^{f(i)}$ instead of $f(i)$ in the algorithm KZG.VrfyEval. Since in our work, we require a public evaluation on $(i; f(i))$, where $f(i)$ should be con dential to the veri er.

Especially, when a prover claims that the committed polynomial $f(\,)$ passes through the point $(0; 0)$, veri ers can verify this statement by simply checking $e(C; g) \stackrel{?}{=} e(w_0; c_0)$.

The DL-based KZG commitment proposed by [37] is homomorphic. Namely, given two KZG.Commit commitments KZG.cmt$_{f_1}$ and KZG.cmt$_{f_2}$ w.r.t polynomials $f_1(x)$ and $f_2(x)$ respectively, one can compute the commitment KZG.cmt$_f$ for $f(x) = f_1(x) + f_2(x)$ as KZG.cmt$_f$ = KZG.cmt$_{f_1}$ KZG.cmt$_{f_2}$.

KZG.Setup($1^{param_1}; t$)
Generate group G;
$G \quad G \; ! \; G_T; G := he; G; G_T i; g \; 2_R \; G, \quad 2_R \; Z_p;$
$\langle g; g \; ; g \; ^2; : : : ; g \; ^t \; 2 \; G^{t+1};$
KZG.PK:= $\langle G; g; g \; ; g \; ^2; : : : ; g \; ^t \rangle$
return KZG.PK

KZG.Commit(KZG.PK; f(x))
$\langle G; g; g \; ; g \; ^2; : : : ; g \; ^t \rangle \quad$ KZG.PK;
$C := \prod_{j=0}^{deg(f)} (g \; ^j)^{f_j};$
return C

KZG.CrtWtn(KZG.PK; f(x); i)
$_i(x) := \frac{f(x) \quad f(i)}{(x \quad i)}; w_i := g \; ^{i(\quad)}; C_{f(i)} := g^{f(i)};$
return ($i; f(i); C_{f(i)}; w_i$)

KZG.VrfyEval (KZG.PK; C; i; $C_{f(i)}$; w_i)
return $e(C; g) \stackrel{?}{=} e(w_i; g \; = g^i) e(C_{f(i)}; g)$

Figure 2.3: DL-based KZG Commitment. Let $f(x)$ 2 $Z_p[x]$: $(x \quad i)$ perfectly divide the polynomial $f(x) \quad f(i)$ for i 2 Z_p.

2.7 Verifiable Encryption

The verifiable encryption is a cryptographic scheme that provides both confidentiality and authenticity for encrypted messages. The scheme works by encrypting a message using a public key and then generating a proof that can be verified to prove the correctness of the encryption.

A verifiable encryption scheme is a encryption scheme with additional proof system.

- (Enc.PK; Enc.SK) Enc.KGen($1^{\;2}$). The key generation algorithm of the verifiable encryption scheme. It takes the parameter $1^{\;2}$ as input and outputs a pair of private and public key (Enc.PK; Enc.SK).

- ct Enc(Enc.PK; m). The encryption algorithm for encrypting the message m. It takes the public key Enc.PK and message m as inputs, and

outputs a ciphertext *ct*.

- *Enc:* Enc.Prf(Enc.PK; *ct*; *m*). The proof algorithm for demonstrating that the ciphertext *ct* was generated correctly. It takes the public key Enc.PK, the ciphertext *ct*, and the message *m* as inputs, and outputs a proof *Enc:* .

- 0=1 Enc.Vrf(*Enc:* ; Enc.PK; *ct*). The corresponding proof system for convincing the veri er that the ciphertext *ct* was generated correctly and has not been modi ed since encryption. This algorithm takes the proof *Enc:* , public key Enc.PK, and ciphertext *ct* as inputs, and outputs 0 for rejecting a ciphertext or 1 for accepting it.

Let be a cyclic group of with prime order generated by . We assume that and are publicly known, and k and k' be further security parameters, where 2^{k} and $2^{k'}$ are the negligible function ($f0;1g^{k}$ is the \challenge space" of the veri er and k' controls the quality of the zero-knowledge property), such that $2^{k} < min\{p'; q'; g$. Let $h_c : f0;1g \; ! \; f0;1g^{k}$ be the hash function. Let $R := f(w;) 2 W : {}^{w} = g$ be the relation de ned by a generator algorithm G' which on 1^{3} outputs a description $= [R; W;]$ of a binary relation R on W . We present the DL-based veri able encryption by Camenisch et al [38], which will be used in this paper, in Figure 2.4.

The veri able encryption algorithm provides both con dentiality and authenticity for encrypted messages and is a useful building block for many cryptographic applications, including e-voting, electronic cash, and secure data storage.

$\underline{\text{Enc.KGen}(1^{param_2})}$
$(p'; q') \quad P;$
$p := (2p' + 1);\ q := (2q' + 1);$
$n := pq,\ n' := p'q';$
$x_1; x_2; x_3 \ 2_R \ [n^2{=}4];$
$g' \ 2_R \ Z_{n^2};\ g := (g')^{2n};$
$y_1 := g^{x_1};\ y_2 := g^{x_2};\ y_3 := g^{x_3};$
$hk \ 2_R \ H \ ;$
$h := (1 + n \bmod n^2) \ 2 \ Z_{n^2};$
$\text{Enc.SK} := (hk; n; x_1; x_2; x_3);$
$\text{Enc.PK} := (hk; n; g; h; y_1; y_2; y_3);$
return (Enc.SK; Enc.PK)

$\underline{\text{Enc}(\text{Enc.PK}; m; L)}$
$r \ 2_R \ [n{=}4];$
$(hk; n; g; h; y_1; y_2; y_3) \quad \text{Enc.PK};$
$u := g^r;$
$o := y_1^r h^m;$
$v := abs((y_2 y_3^{H_{hk}(u;o;L)})^r);$
$ct := (u; o; v);$
return ct

$\underline{\text{Enc.Prf}(\text{Enc.PK}; ct; m)}$
$(hk; n; g; h; y_1; y_2; y_3) \quad \text{Enc.PK};$
$(u; e; v) \quad ct;$
$G\ ,\ := \ ^m;$
$s \ 2_R \ [n{=}4],\ l := g^m h^s;$
$r'; s' \ 2_R \ Z_{\ n2^{k+k'}\ ^2; n2^{k+k'}\ ^2};\ m' \ 2_R \ Z_{\ 2^{k+k'}; \ 2^{k+k'}};$
$u' := g^{2r'},\ e' := y_1^{2r'} h^{2m'},\ v' := (y_2 y_3^{H_{hk}(u;e;L)})^{2r'},$
$' := \ ^{m'},\ l' := g^{m'} h^{s'};$
$c := h_c(ctjj\text{Enc.PK});$
$\not{r} := r' \quad cr,\ \not{s} := s' \quad cs,\ \not{m} := m' \quad cm;$
$Enc: \ := (ct; \ ; L; \ ; l; u'; e'; v'; \ '; l'; \not{r}; \not{s}; \not{m});$
return $Enc: \ ;$

$\underline{\text{Enc.Vrf}(Enc: \ ; \text{Enc.PK}; \text{Enc.PK})}$
$(ct; \ ; L; \ ; l; u'; e'; v'; \ '; l'; c; \not{r}; \not{s}; \not{m}) \quad Enc: \ ;$
$(u; e; v) \quad ct;\ (hk; n; g; h; y_1; y_2; y_3) \quad \text{Enc.PK};$
$c := h_c(ctjj\text{Enc.PK});$
If $u' \not{=} u^{2c} g^{2\not{r}}$ or $e' \not{=} e^{2c} y_1^{2\not{r}} h^{2\not{m}}$ or $v' \not{=} v^{2c} (y_2 y_3^{H_{hk}(u;e;L)})^{2\not{r}}$ or
$' \not{=} \ ^c \ ^{\not{m}}$ or $l' \not{=} l^c g^{\not{m}} h^{\not{s}}$ or $n{=}4 < \not{m} < n{=}4$:
return 0;
return 1

Figure 2.4: DL-based Verifiable Encryption.

Chapter 3

AuxChannel

This chapter proposes AuxChannel in enabling e cient bi-directional payment channel for scriptless blockchains with unlimited lifespan. In order to construct AuxChannel, we further propose a new cryptographic primitive called Consecutive Veri able Encrypted Signature (CVES) (as known as Consecutive Adaptor Signature).

Brie y, AuxChannel guarantees that neither parties can prevent the channel from closing or stealing 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. We leverage CVES to minimise the involvement of the script-enabled blockchain, i.e., with communication complexity $O(1)$.

3.1 Consecutive Verifiably Encrypted Signature (a.k.a. Consecutive Adaptor Signature)

3.1.1 Introduction of CVES

We propose a new cryptographic primitive, called *Consecutive Veri ably 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 de ned 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 veri er that the corresponding encryption keys are consecutive (that is, the new encryption key is well-formed or generated from KeyUpdate); ConsVerify veri es the proof generated by ConsProof.

We now present the formal de nition 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^i to denote recursively executing KeyUpdate i times.

Definition 1 (Consecutive Veri ably Encrypted Signature)**.** *A consecutive veriably 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*$(dk^{i-1}) \rightarrow (dk^i; \hat{dk}^i; ek^i)$*: 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*$(dk^{i-1}; dk^i) \rightarrow P^i$*: a probabilistic algorithm, which on inputs two decryption keys* dk^{i-1} *and* dk^i*, outputs a proof* P^i*.*

- *ConsVerify*$(ek^{i-1}; ek^i; P^i) \rightarrow \{0,1\}$*: a deterministic algorithm, which on inputs two consecutive encryption keys* ek^{i-1}*,* ek^i *and a proof of consecutiveness* P^i*, outputs* 0 *to reject or* 1 *to accept the inputs.*

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.

3.1.2 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 verifiability* 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 $(i \quad 1)$-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 $0 \quad j < i$.

We then give the formal de nition of CVES security properties. We rst include the concept of *validity*, *recoverability* and *consecutiveness* into a single de nition of *correctness*, as de ned below:

Definition 2 (Correctness of CVES). *A CVES is correct if the following properties hold. For any $i > 0$ and any message m^i, we require*

$$
\begin{aligned}
(sk; pk) &\quad \text{Gen}(\ ) \\
(dk^0; \hat{dk}^0; ek^0) &\quad \text{EncGen}(\ ) \\
(dk^{i-1}; \hat{dk}^{i-1}; ek^{i-1}) &\quad \text{KeyUpdate}^{i-1}(dk^0) \\
(dk^i; \hat{dk}^i; ek^i) &\quad \text{KeyUpdate}(dk^{i-1}) \\
\sigma^i &\quad \text{Sign}(sk; m^i) \\
\hat{\sigma}^i &\quad \text{EncSign}(sk; ek^i; m^i) \\
P_i &\quad \text{ConsProof}(dk^{i-1}; dk^i) \\
1 &\quad \text{ConsVerify}(ek^{i-1}; ek^i; P^i) \\
1 &\quad \text{EncVerify}(pk; ek^i; \hat{\sigma}^i; m^i) \\
\hat{dk}^i &\quad \text{Rec}(\sigma^i; \text{RecKey}(\hat{\sigma}^i)) \\
1 &\quad \text{Verify}(pk; \text{DecSig}(\hat{dk}^i; \hat{\sigma}^i); m^i)
\end{aligned}
\tag{3.1}
$$

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 $1 \leftarrow Verify(pk; \sigma; m)$.

Encsigning Oracle: Let 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 $1 \leftarrow EncVerify(pk; ek^i; \hat{\sigma}^i; m)$.

Definition 3 (Unforgeability of CVES). *A CVES is unforgeable, if the advantage $Adv = Pr[Exp_A^{unforgeability} = 1]$, where $Exp_A^{unforgeability}$ is defined in Figure 3.1, of any probabilistic polynomial time (PPT) adversary in forging a signature σ^i given access to signing oracle* O *and encsigning oracle* E *is negligible. The probability is taken over the coin tosses of the key generation algorithm Gen and EncGen, the oracles* O *and* E*. The adversary is additionally constrained*

that $m \neq m$, *where* m *is a message chosen by an adversary* A, m *is an input to* E *and* O *accessed by the adversary.*

Definition 4 (One-wayness of CVES). *A CVES is one-way, if the advantage* $Adv = Pr[Exp_{A}^{one\text{-}way} = 1]$, *where* $Exp_{A}^{one\text{-}way}$ *is defined in Figure 3.1, 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 5 (Consecutive Verifiability). *A CVES is consecutive verifiable, if the advantage* $Adv = Pr[Exp_{A}^{cverifiability} = 1]$, *where* $Exp_{A}^{cverifiability}$ *is defined in Figure 3.1, 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 secure CVES scheme should satisfy the four properties defined in Definition 2, 3,4,5.

3.1.3 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 [3]. 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).

$\underline{\mathbf{Exp}\ Exp_{\mathsf{A}}^{\text{unforgeability}}(\ ):}$
$(sk; pk) \quad \text{Gen}(\ )$
$(dk^0; \hat{dk}^0; ek^0) \quad \text{EncGen}(\ )$
$(i; st) \quad A_0(pk; ek^0)$
$(dk^{i\ 1}; \hat{dk}^{i\ 1}; ek^{i\ 1})$
$\text{KeyUpdate}^{i\ 1}(dk^0)$
$(dk^i; \hat{dk}^i; ek^i) \quad \text{KeyUpdate}(dk^{i\ 1})$
$P^i \quad \text{ConsProof}(dk^{i\ 1}; dk^i)$
$m \quad A_1^{O(\ );E(\ )}(st; pk; ek^i)$
$\hat{}^i \quad \text{E}(pk; ek^i; m\ )$
$^i \quad A_2(st; pk; ek^{i\ 1}; ek^i; P^i; \hat{}^i\ )$
If $\text{Verify}(pk;\ ^i; m\ ) = 1$:
Return 1
Else Return 0

$\underline{\mathbf{Exp}\ Exp_{\mathsf{A}}^{\text{one-way}}(\ ):}$
$(sk; pk) \quad \text{Gen}(\ )$
$(dk^0; \hat{dk}^0; ek^0) \quad \text{EncGen}(\ )$
$(i; st) \quad A_0(pk; ek^0)$
$(dk^i; \hat{dk}^i; ek^i) \quad \text{KeyUpdate}^i(dk^0)$
$(dk^{i\ 1}; m\ ) \quad A_1(st; dk^i)$
$(dk^i; \hat{dk}^i; ek^i\ ) \quad \text{KeyUpdate}(dk^{i\ 1}\ )$
$\hat{}^i \quad \text{EncSign}(sk; ek^i; m\ )$
$^i \quad \text{DecSig}(\hat{dk}^i; \hat{}^i)$
If $\text{Verify}(pk;\ ^i; m\ ) = 1$:
Return 1
Else Return 0

$\underline{\mathbf{Exp}\ Exp_{\mathsf{A}}^{\text{cveri ability}}(\ ):}$
$(sk; pk) \quad \text{Gen}(\ )$
$(dk^0; \hat{dk}^0; ek^0) \quad \text{EncGen}(\ )$
$(i; st) \quad A_0(pk; ek^0)$
$(dk^i; \hat{dk}^i; ek^i) \quad \text{KeyUpdate}^i(dk^0)$
$(dk^{i+1}; \hat{dk}^{i+1}; ek^{i+1}; m; P\ ) \quad A_1(st; dk^i)$
$\hat{} \quad \text{EncSign}(sk; ek^{i+1}; m\ )$
$(dk^{i+1}; \hat{dk}^{i+1}; ek^{i+1}) \quad \text{KeyUpdate}(dk^i)$
$\text{DecSign}(\hat{dk}^{i+1}; \hat{}\ )$
If $\text{ConsVerify}(ek^i; ek^{i+1}; P\ ) = 1 \&\& \text{Verify}(pk;\ ; m\ ) = 0$:
Return 1
Else Return 0

Figure 3.1: Experiments used to de ne unforgeability, one-wayness, and consecutive veri ability properties of CVES.

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 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 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 Z_p (for a cyclic group with prime order p), the one-way function group Z_n and another group 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 G_p with prime order p. Let $N = p'q'$, where $p'; q'$ are primes with length $_1$ 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). Z_N is thus a group with unknown order (N). Let $g; h$ be the group generators of G_p and Z_N respectively, and $H : f0; 1g \; ! \; Z_p$ be a hash function. Suppose that it is hard to solve a discrete logarithm problem in the group 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 $_2$ which is another security parameter. The order of the group Z_n is with unknown order (n) (again we can use MPC to generate n such that no one knows its factorization). The one-way function can be set as $x \; 7! \; x^2 \bmod n$.

The complete construction of Schnorr-based CVES scheme is shown in Figure 3.2. (The detailed explanation and underlying instantiation of the NIZK in the algorithms ConsVerify and ConsProof will be presented later in this section.)

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 veri er. In practice, we simply set $N = n$, if this number is generated using MPC or other trusted methods 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 3.2, the public key used in zero-knowledge proof is constructed as $Y' = h^{y'} \bmod N$, where $y' = y^2 \bmod n$. Thus, the distribution of y', where $y' \; 2 \; [0; n)$, would a ect the distribution of Y'.

Ideally, we hope Υ'^{0} is uniformly distributed from Z_N, which can be guaranteed when $n = (N)$, where (N) is the group order of Z_N. As (N) is unknown, to ensure that $(h^{\gamma'^{0}} \bmod N)$ can cover all group elements in Z_N, n should be greater than (N). Thus if we set $n = N > (N)$, and when γ'^{0} is from $[0; n)$, and $n = N$ is large enough, $\Upsilon'^{0} = (h^{\gamma'^{0}} \bmod N)$ is statistically indistinguishable from Υ'^{0}, which is uniformly in 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.*

We then provide the security proof of the Theorem 1.

Lemma 3.1.1. *The Schnorr CVES is correct.*

Proof. In our Schnorr CVES (Figure 3.2), we have that sk is $(x; X)$, and the corresponding verification key pk is $X := g^x$. A decryption key dk^i is $(\gamma'; y; Y)$ w.r.t the encryption key $ek^i := (\Upsilon'; Y)$, where $y := \gamma' \bmod p$ and $\Upsilon' := h^{\gamma'} \bmod N; Y := g^y$. 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 EncVerify$(pk; ek^i; \hat{}^i; m^i) = 1$. Similarly, as $R = g^r$, $c := H(m_i \| R \| X)$ and $g^s X^{\ c} = g^{s\ cx} = g^{(r+cx)\ cx} = g^r = R$, Verify$(pk; \ _i; m_i) = 1$ always holds. Thus the encrypted signature $\hat{}_i := (\hat{R}; \hat{s})$ and the original signature $_i := (R; s)$ are valid.

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

Also, the i-th decryption key dk^i is $(\gamma'; y; Y)$ and $(i+1)$-th decryption key dk^{i+1} is $(\gamma'^{0}; y^{0}; Y^{0}) = (\gamma'^{2} \bmod N); \gamma'^{0} \bmod p; g^{y^{0}})$ and encryption key ek^{i+1} is $(\Upsilon'^{0}; Y^{0}) = ((h^{\gamma'^{0}} \bmod N); (g^{y^{0}}))$. According to our Schnorr CVES construction in Figure 3.2,

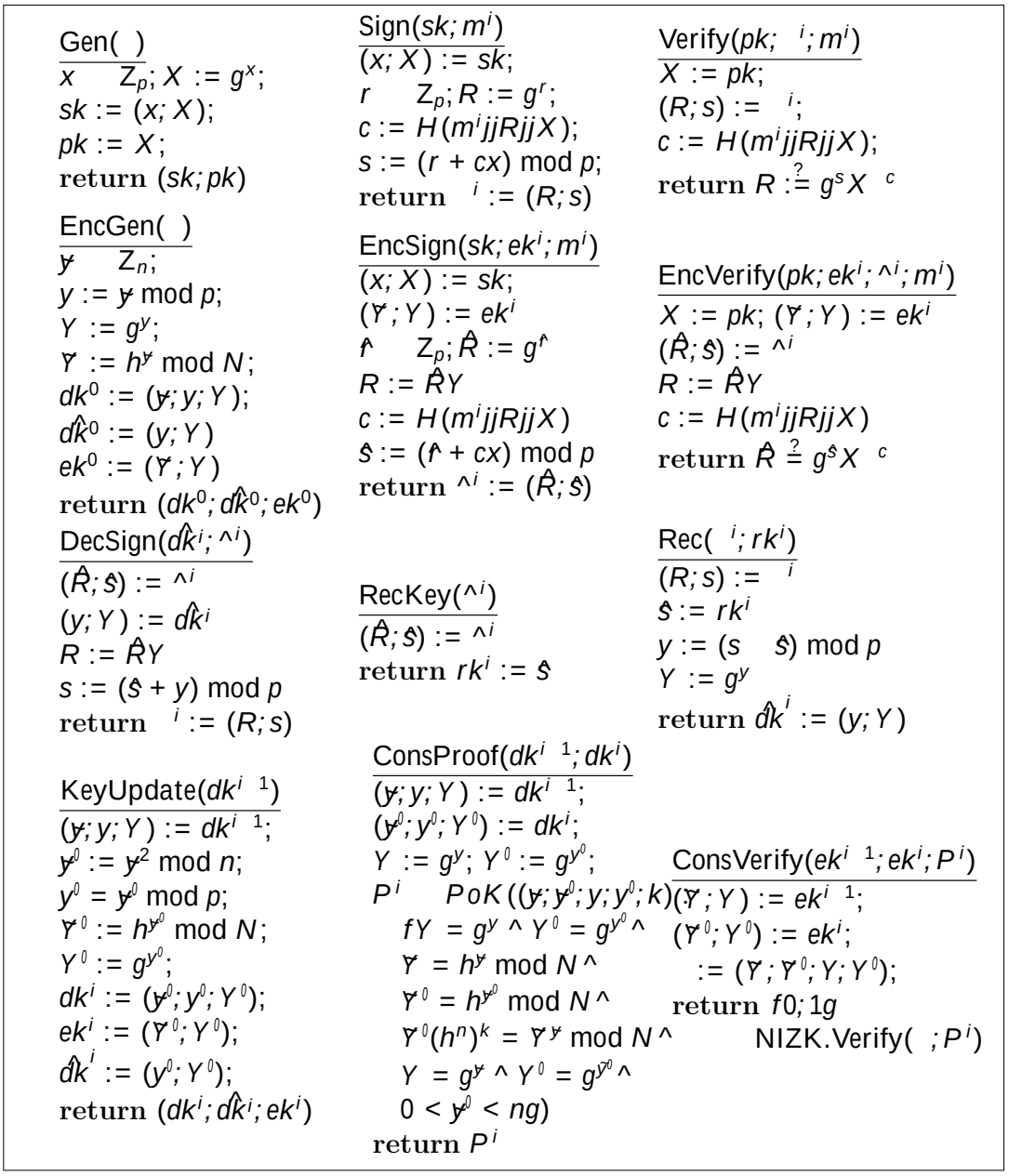

Figure 3.2: The construction of Schonrr-based CVES algorithms.

the NIZK system proves the following proof-of-knowledge:

$$PoK((\gamma;\gamma^0;y;y^0;k) : \{Y = g^y \wedge Y^0 = g^{y^0} \wedge \Gamma = h^\gamma \bmod N \wedge \Gamma^0 = h^{\gamma^0} \bmod N \wedge \Gamma^{0}(h^n)^k = \Gamma^\gamma \bmod N \wedge Y = g^\gamma \wedge Y^0 = g^{y^0} \wedge 0 < \gamma^0 < n\}$$

Our instantiation (see later this section) simply sets $n = N = p^0q^0$.

Since $\gamma^0 = \gamma^2 \quad kN$ (or $\gamma^0 = \gamma^2 \bmod N$) and

$$\begin{aligned}\Gamma^{0}(h^n)^k &= \Gamma^{0}(h^{Nk}) = \Gamma^{0}(h^{\gamma^2 \quad y^0}) = \Gamma^{0}(h^{\gamma^2 \quad (\gamma^2 \bmod N)}) \\ &= (h^{y^0} \bmod N)(h^{\gamma^2 \quad (\gamma^2 \bmod N)}) \\ &= h^{\gamma^2} \bmod N = (h^\gamma \bmod N)^\gamma \bmod N = \Gamma^\gamma \bmod N\end{aligned}$$

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

Lemma 3.1.2. *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 A who can break the unforgeability, then given a DL problem instance $(X; g)$ 2 G_p, we can construct a simulator S_3 who can make use of A to output an integer x such that $X = g^x$.

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, S_3 generates according to the algorithms EncGen, KeyUpdate and ConsProof. S_3 gives $(pk; ek^i; P^i; g)$ to A.

The game of forging a valid signature runs in three stages by A: rst A_0 chooses a certain session and records the public keys $(pk; ek^i; P^i; g)$, and then A_1 chooses a message m, nally A_2 outputs a forged signature := $(R; s)$ on m,

such that $R = g^s X^{H(m||R||X)}$, where X is the pk given to A. Remark that m was not queried to O before and that signature is valid.

S_3 simulates the random oracle H as normal (keep a table to make the consistency of input and output), and the encsigning oracle 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 $c \in Z_p$ and $\hat{s} \in Z_p$ 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 E, S_3 also simulates the signing oracle O (with input the public keys pk and the chosen message m) as follows:

- Extract X and Y from pk and ek^i;
- Pick $c \in Z_p$ and $\hat{s}, s \in Z_p$ 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 $E(X; Y; m)$ 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 $\sigma = (R; s)$.

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, A outputs a forged signature

:= $(R; s)$ on its chosen message m, where m is inputted to encsigning oracle, but not inputted to the signing oracle.

S_3 then rewinds the random tape to the point that A is making the random oracle query for the value $c = H(m \| R \| X)$, and use another random tape to supply a di erent value $\tilde{c} = H(m \| R \| X)$, so that A outputs a di erent signature $\tilde{} = (R; \tilde{s})$. [This is called the Forking Lemma.]

Now there are two forged signatures of m outputted by A: $= (R; s)$ and $\tilde{} = (R; \tilde{s})$ 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 $\hat{R} = g^r$. By solving these two equations, S_3 can output x which is the solution for the DL problem instance. ∎

Lemma 3.1.3. *Assuming that the SQROOT problem is hard, the Schnorr CVES scheme satis es one-wayness.*

Proof. We prove this lemma by contradiction. Assuming that there exists an adversary A who can break the one-wayness of Schnorr CVES scheme. Given a SQROOT problem instance $(\ ; N)$, where $\ \in Q_N$, we can construct a simulator S_4 who can make use of A to output an integer $\tilde{y}$ such that $\ = \tilde{y}^2 \bmod N$.

S_4 picks $\ $ from the SQROOT problem instance $(\ ; N)$ and sets $dk^i := (\ ; y'; Y')$, where $y' = \ \bmod p$ and $Y' = h^{y'} \bmod p$, and the corresponding encryption key is $ek^i := (\Upsilon'; Y')$. For the signing key and veri cation key-pair $(sk; pk)$, where $sk := (x; X)$ and $pk := X$, the initial and later stage decryption and encryption keys, S_4 generates according to the algorithms Gen, EncGen and KeyUpdate. S_4 then gives $(sk; dk^i; g; h; N)$ to A.

The game of reversing a decryption key from it previous session runs in two stages: rst, A_0 chooses a certain session to attack and records the decryption key $dk^i = (\ ; y'; Y')$ and encryption key $ek^i := (\Upsilon'; Y')$ (while the decryption

key $dk^{i-1} = (y', y, Y)$ is unknown, such that $y' := r' \bmod N$, $y := y' \bmod p$, and $Y := g^y$), where $Y' := h^{r'} \bmod N$, then A_1 outputs $dk^{i-1} := (y', y, Y)$, which satisfies:

$$(dk^i, \hat{dk}^i, ek^i) \quad \text{KeyUpdate}(dk^{i-1}) \tag{3.2}$$

$$\hat{}_i \quad \text{EncSign}(sk, ek^i, m) \tag{3.3}$$

$$_i \quad \text{DecSign}(\hat{dk}^i, \hat{}_i) \tag{3.4}$$

$$1 \quad \text{Verify}(pk, m, _i) \tag{3.5}$$

where $dk^i := (y'^0, y^0, Y^0)$, $\hat{dk}^i := (y^0, Y^0)$, $ek^0 := (Y'^0, Y^0)$, $\hat{}_i := (\hat{R}, \hat{s})$ and $_i := (R, s)$. Let $\hat{R} = g^{\hat{r}}$. We can imply that $y^0 = y'^0 \bmod p$ (**from 3.2**) $)$ $y^0 = (y'^2 \bmod N) \bmod p$.

Let $R = g^r$, then $R = \hat{R}Y^0$ (**from 3.3**) $)$ $r = (\hat{r} + y^0) \bmod p$; $\hat{s} = (\hat{r} + cx) \bmod p$ (**from 3.3**) $)$ $s = (r \quad y^0 + cx) \bmod p$; where $c = H(m \,||\, R \,||\, X)$.

As all the inputs of equation 3.3 are generated according to the algorithms Gen, EncGen and KeyUpdate, according to the correctness of Schnorr we have $1 \quad \text{EncVerify}(pk, ek^i, \hat{})$, which implies that $\hat{R} = g^{\hat{s}} X^c$. Thus, $R = \hat{R}Y^0$ (**from 3.4**) $)$ $R = g^{\hat{s}} X^c g^{y^0}$ On the other side, $R = g^s X^c$ (**from 3.5**) $)$ $R = g^{\hat{s}+y^0} X^c$ From the above equation, we have $R = g^{\hat{s}} X^c g^{y^0} = g^{\hat{s}+y^0} X^c$ $)$ $c = c$ $)$ $H(m \,||\, R \,||\, X) = H(m \,||\, R \,||\, X)$.

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:

$$R = R \;) \; \hat{R}Y^0 = \hat{R}Y^0 \;) \; y^0 = y^0$$

$$) \; r' \bmod p = y'^0 \bmod p = (y'^2 \bmod N) \bmod p$$

That is S_4 can output y', which is the solution for the SQROOT problem instance (r', N). ∎

Lemma 3.1.4. *Assuming that the underlying NIZK system used in our ConsProof and ConsVerify is secure, the Schnorr CVES is consecutive veri able.*

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 A who can break the consecutive veri ability of Schnorr CVES scheme.

We can construct a simulator S_5 who can make use of A to output a proof on a chosen NIZK problem instance $(; w)$, which includes a chosen NP problem instance $:= (\Upsilon ; \Upsilon^{\prime} ; Y ; Y^{\prime})$ and a ve-element tuple $w := (\gamma ; \gamma^{\prime} ; y ; y^{\prime} ; k)$, such that NIZK.Verify$(; P) = 1$, where $\gamma^{\prime} ; y^{\prime} ; \Upsilon^{\prime} ; Y^{\prime}$ are chosen by A.

S_5 picks $\gamma ; y ; \Upsilon ; Y$ from an NIZK problem instance $(; w)$, where $:= (\Upsilon ; \Upsilon^{\prime} ; Y ; Y^{\prime})$, and $w := (\gamma ; \gamma^{\prime} ; y ; y^{\prime} ; k)$, then sets decryption key $dk^i := (\gamma ; y ; Y)$ and the corresponding encryption key $ek^i := (\Upsilon ; Y)$. For the signing and veri - cation key-pair $(sk ; pk)$, where $sk := (x ; X)$ and $pk := X$, the initial and later decryption keys , S_5 generates according to the algorithms Gen, EncGen and KeyUpdate. S_5 then gives $(dk^i ; ek^i ; k)$ to A.

The game of breaking the consecutive veri ablility of the Schnorr CVES scheme is de ned in three stages: rst A_0 chooses a certain session and records the useful intermediate variable, then A_1 outputs a decryption and encryption key-pair $(fdk^{i+1} g ; f\hat{dk}^{i+1} g ; fek^{i+1} g)$, where $dk^{i+1} := (\gamma^{\prime} ; y^{\prime} ; Y^{\prime})$, $\hat{dk}^{i+1} := (y^{\prime} ; Y^{\prime})$, $ek^{i+1} := (\Upsilon^{\prime} ; Y^{\prime})$, $y^{\prime} = \gamma^{\prime} \mod p$, $Y^{\prime} = g^{y^{\prime}}$, $\Upsilon^{\prime} = h^{\gamma^{\prime}} \mod N$, on the input the decryption key dk^i, nally A_2 outputs a proof P on dk^i and dk^{i+1}. These outputs satisfy:

$$(dk^{i+1} ; \hat{dk}^{i+1} ; ek^{i+1}) \quad \text{KeyUpdate}(dk^i) \tag{3.6}$$

$$\wedge \quad \text{EncSign}(sk ; ek^{i+1} ; m) \tag{3.7}$$

$$\text{DecSign}(\hat{dk}^{i+1} ; \wedge) \tag{3.8}$$

$$1 \quad \text{ConsVerify}(ek^i; ek^{i+1}; P) \tag{3.9}$$

$$0 \quad \text{Verify}(pk; m;) \tag{3.10}$$

where $dk^{i+1} := (y'^0; y^0; Y^0)$, $\hat{dk}^{i+1} := (y^0; Y^0)$, $ek^{i+1} := (Y'^0; Y^0)$, $\hat{} := (\hat{R}; \hat{s})$, $:= (R; s)$, and m is an arbitrary message.

We can imply that $y'^0 = y'^2 \bmod N; y^0 = y'^0 \bmod p; Y^0 = g^{y^0}$; (from 3.6): Let $\hat{R} = g^{\hat{r}}$, we can also imply that $R = \hat{R} Y^0; c = H(m \,||R\,||X); \hat{s} = (\hat{r} + c\, x) \bmod p$; (from 3.7), $R = \hat{R} Y^0; s = (\hat{s} + y^0) \bmod p$ (from 3.8), $1 \quad \text{NIZK.Verify}(ek^i; ek^{i+1}; P)$ (from 3.9) and $c = H(m \,||R||X); R \stackrel{?}{=} g^s X^c$ (from 3.10).

From the above equations, the following always hold: $R = \hat{R} Y^0 \stackrel{?}{=} g^s Y^c) \hat{R} Y^0 \stackrel{?}{=} g^{\hat{s}+y^0} Y^c) \hat{r} + y^0 \stackrel{?}{=} \hat{s} + y^0 \quad cx) \hat{r} \stackrel{?}{=} (\hat{r} + c\, x) \bmod p \quad cx) cx \stackrel{?}{=} c\, x \bmod p) c \stackrel{?}{=} c) H(m \,||R||X) \stackrel{?}{=} H(m \,||R\,||X)$. Remark that, for the same input, the hash function $H()$ produces a same output, otherwise, it yields the different output. Thus, if $c \stackrel{?}{=} c$, it implies that $m \,||R||X \stackrel{?}{=} m \,||R\,||X$. We then have: $R \stackrel{?}{=} R) \hat{R} Y^0 \stackrel{?}{=} \hat{R} Y^0) Y^0 \stackrel{?}{=} Y^0) y^0 \stackrel{?}{=} y^0) (y'^2 \bmod N) \bmod p = y'^0 \bmod p \stackrel{?}{=} y'^0 \bmod p$, which implies that, $y'^0 \stackrel{?}{=} y'^2 \bmod N$.

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

$$Pr[\text{NIZK.Verify}(; P) = 1] > negl( );$$

It is contradict to the NIZK soundness [39; 40]. That is S_5 can output a proof P on the chosen NIZK instance $(; w)$, which breaks the soundness property of

the NIZK system. ∎

According to Lemma 3.1.1, 3.1.2, 3.1.3, 3.1.4, Theorem 1 is proved.

Instantiation of the algorithms ConsVerify and ConsProof. We have instantiated and implemented the algorithms KeyUpdate( ), ConsVerify( ) and ConsProof( ) (in Figure 3.2). We then give the detailed explanation and underlying instantiation of the NIZK in the algorithms ConsVerify and ConsProof.

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:

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

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}:

$$PoK^2_{CVES} = \{(u_i) : U_i = h^{u_i} \bmod N \wedge 0 < u_i < N\}; \tag{3.11}$$

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 $_3$; $_4$ be some security parameters which determine the soundness and (statistical) zero-knowledgeness of the proof. De ne a cryptographic hash function $H : \{0;1\}^* \to \{0;1\}^{_3}$. The rst part of the zero-knowledge proof is constructed as follows:

- (*Commitment*). In a session i, the prover chooses $u_{i\ 1}$, u_i, k uniformly at random from $[0; 2^{jNj+\ 3+\ 4\ 1}]$, and computes four commitments: $Y_{i;1} = g^{u_i}$, $Y_{i;2} = h^{u_{i\ 1}} \bmod N$, $Y_{i;3} = h^{u_i} \bmod N$, and $Y_{i;4} = U_i^{u_{i\ 1}}(h^{N})^{k} \bmod N$.

- (*Challenge*). The veri er picks a $_3$-bit challenge uniformly at random. In the non-interactive form, the $_3$-bit challenge can be computed as the hash of the commitments and the public parameters, i.e., $!_i = H(g; h; T_{i\ 1}; T_i; U_{i\ 1}; U_i; Y_{i;1}; Y_{i;2}; Y_{i;3}; Y_{i;4})$.

- (*Response*). The prover computes the response: $(z_{u_{i\ 1}} = {}_{u_{i\ 1}}\ !_i\ u_{i\ 1}, z_{u_i} = {}_{u_i}\ !_i\ u_i, z_k = {}_k\ !_i\ k)$ as the proof.

- (*Verify*). The veri er outputs 1 if it holds that:

$$Y_{i;1} = h^{z_{u_{i\ 1}}} U_{i\ 1}^{!_i} \bmod N; Y_{i;2} = g^{z_{u_i}} T_i^{!_i};$$

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

Completeness: if $z_{u_{i\ 1}} = {}_{u_{i\ 1}}\ !_i\ u_{i\ 1}$, $z_{u_i} = {}_{u_i}\ !_i\ u_i$, and $z_k = {}_k\ !_i\ k$, then $Y_{i;1} = g^{u_i} = g^{z_{u_i}} T_i^{!_i}$, $Y_{i;2} = h^{u_{i\ 1}} \bmod N = h^{z_{u_{i\ 1}}} U_{i\ 1}^{!_i} \bmod N$, $Y_{i;3} = h^{u_i} \bmod N = h^{z_{u_i}} U_i^{!_i} \bmod N$ and $Y_{i;4} = U_i^{u_{i\ 1}}(h^{N})^{k} \bmod N = U_{i\ 1}^{z_{u_{i\ 1}}} U_i^{!_i}(h^{N})^{z_k} \bmod N$.

Zero Knowledge: for every $g \in G_p$ and $h \in Z_N$, let P be the proof and V be the verify algorithm, the output of the simulator needs to be indistinguishable from the distribution of the transcripts:

$$
\begin{aligned}
&View_V(\mathrm{P}(u_{i-1}; u_i) \$ \mathrm{V}(g; h; U_{i-1}; U_i; T_{i-1}; T_i)) \\
&= f(g^{u_i}; h^{u_{i-1}} \bmod N; h^{u_i} \bmod N; U_{i-1}^{u_i} (h^{N})^{k}; \\
&\quad \omega_i; \omega_{u_{i-1}} \oplus \omega_{u_i-1}; \omega_{u_i} \oplus \omega_{u_i}; \omega_k \oplus \omega_i k) : \\
&\quad \omega_i = H(g; h; U_{i-1}; U_i; T_{i-1}; T_i; Y_{i;1}; Y_{i;2}; Y_{i;3}; Y_{i;4}); \\
&\quad (r_{u_{i-1}}; r_{u_i}; r_k) \in [0; 2^{jNj+\ell_3+\ell_4} - 1]g \\
&= f(Y_i; Y_{i;2}; Y_{i;3}; Y_{i;4}; \omega_i; z_{u_{i-1}}; z_{u_i}; z_k) : \\
&Y_{i;1} = g^{z_{u_i}} T_i^{\omega_i}; Y_{i;2} = h^{z_{u_{i-1}}} U_{i-1}^{\omega_i} \bmod N; \\
&Y_{i;3} = h^{z_{u_i}} U_i^{\omega_i} \bmod N; Y_{i;4} = U_{i-1}^{z_{u_i}} U_i^{\omega_i} (h^{N})^{z_k} \bmod N g
\end{aligned}
\tag{3.12}
$$

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

$\underline{S(g; h; U_{i-1}; U_i; T_{i-1}; T_i)}$:

$z_{u_{i-1}}; z_{u_i}; z_k \in [0; 2^{jNj+\ell_3+\ell_4} - 1]$;

$\omega_i \in [0; 2^{\ell_3} - 1]$;

$Y_{i;1} = g^{z_{u_i}} T_i^{\omega_i}$;

$Y_{i;2} = h^{z_{u_{i-1}}} U_{i-1}^{\omega_i} \bmod N$

$Y_{i;3} = h^{z_{u_i}} U_i^{\omega_i} \bmod N$

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

Since ω_i is chosen from at random $[0; 2^{\ell_3}]$, and $z_{u_{i-1}}; z_{u_i}$ and z_k are chosen at random from $[0; 2^{jNj+\ell_3+\ell_4}]$, 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 veri er with probability $1 - negl(\ell_0)$, where $negl()$ is a negligible probability with ℓ_0-bit inputs. We construct the extractor as follows:

^{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 $!_{i;a}$ using the two tuple. Then we have a response tuple ($z^a_{u_i\ 1}$,$z^a_{u_i}$,z^a_k) [0; $2^{jNj+\ 3+\ 4}$ 1];

2: Rewind the prover I to its state after the first message;

3: Run the prover I to obtain the second challenge $!_{i;b}$ 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 = $\frac{z^a_{u_i}\ z^b_{u_i}}{!_{i;b}\ !_{i;a}}$ (*in* Z), $u_{i\ 1}$ = $\frac{z^a_{u_i\ 1}\ z^b_{u_i\ 1}}{!_{i;b}\ !_{i;a}}$ (*in* Z). It remains to argue that $!_{i;b}\ !_{i;a}jz^a_{u_i}\ z^b_{u_i}$ (resp. $!_{i;b}\ !_{i;a}jz^a_{u_i\ 1}\ z^b_{u_i\ 1}$) so that the extractor can compute this without knowing (N).

We sketch the proof that $!_{i;b}\ !_{i;a}jz^a_{u_i}\ z^b_{u_i}$ under the strong RSA assumption. First, we have $Y_{i;3} = h^{z^a_{u_i}} U_i^{!_{i;a}} \bmod N$ and $Y_{i;3} = h^{z^b_{u_i}} U_i^{!_{i;b}} \bmod N$. Thus, we have $h^{z^a_{u_i}} U_i^{!_{i;a}} = h^{z^b_{u_i}} U_i^{!_{i;b}} (\bmod N)$ Denote by and the values $z^a_{u_i}\ z^b_{u_i}$ and $!_{i;b}\ !_{i;a}$, thus, we have h = U_i (mod$N$).

Suppose on a contrary, does not divide . Let f = gcd(;) and = $f^{\ 0}$ and = $f^{\ 0}$ for some ^{0}; ^{0} 6= 1.

This implies $h^{\ 0}$ = $U_i^{\ 0}$(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(^{0}; ^{0}) = 1, there exists integers $X;Y$ such that $X^{\ 0} + Y^{\ 0}$ = 1. Therefore, $h = h^{X\ ^{0}+Y\ ^{0}}(\bmod N) = (U_i^X h^Y)^{\ 0}(\bmod N)$.

Therefore, $(U_i^X h^Y)$; ^{0}) is a solution to the strong RSA problem on ($N;h$). In other words, if $!_{i;b}\ !_{i;a}jz^a_{u_i}\ z^b_{u_i}$, we can solve the strong RSA problem.

The proof that $!_{i;b}\ !_{i;a}jz^a_{u_i\ 1}\ z^b_{u_i\ 1}$ 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 $\frac{1}{2^{\ 3}}$. Therefore, the knowledge error here is $negl(^{0})$ = $\frac{1}{2^{jNj+\ 3+\ 4}}$.

The second part PoK^{2}_{CVES} requires that at each session, the prover proves that his/her updated decryption key $u_{P;i}$ lies in the interval $[0; N \quad 1]$. This proof can be done by various existing range proof systems, for example [41; 42; 43; 44]. As the second part PoK^{2}_{CVES} is not the contribution of the paper, we omit the inistantiation of proving $u_{P;i}$ lies in the interval $[0; N \quad 1]$.

3.1.4 Performance Evaluation of Schnorr CVES

Our implementation is based on a zero-knowledge library, emmy [45]. For simplicity, we set $n = N = p'q'$, and adopt an unknown group Z_N and a cyclic group G_p with the prime order p, where $jNj = 2048$ and $p = 2^{252} +$ 27742317777372353535851937790883648493. The setting of the prime order p adopts the order of a subgroup of Ed25519 curve, which is widely employed by several cryptocurrencies, such as Monero (the top one cryptocurrencies ranked by Coinmarketcap), Nano (the digital currency for the real world), Decred (hybridized PoW/PoS cryptocurrency), Chain Core (enterprise-grade blockchain infrastructure that enables organizations to build better nancial services from the ground up), etc [1].

Suppose a signer has generated an encryption-decryption key-pair $(ek^{i \ 1}; dk^{i \ 1})$ and shared $ek^{i \ 1}$ with a veri er, where $i \ 2 \ N^{+}$. He then performs $(dk^{i}; \hat{dk}^{i}; ek^{i})$ KeyUpdate$(dk^{i \ 1})$ to update the encryption and decryption keys. To convince the veri er that $dk^{i \ 1}$ and dk^{i} are consecutive, the signer creates a proof P^{i} ConsProof$(dk^{i \ 1}; dk^{i})$, and shares P^{i} and ek^{i} with the veri er. 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 veri cation requires about 348 ms. The commu-

[1] https://ianix.com/pub/ed25519-deployment.html

nication latency is transferring 16:9 KB data, including the required proof and encryption key ($P^i; ek^i$).

3.2 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 3.3.

3.2.1 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 [24].

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 [14]), 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.

3.2.2 Strawman Design

One natural step is to leverage a script-enabled blockchain (denoted L_2) to help the scriptless blockchain (denoted 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 L_2. As this only provides compensate to the victim on L_2 rather than correcting the misbehavior on L_1, the victim will lose its fund on L_1 (regardless of requiring permissions from one or both parties to close a channel) and the malicious attacker still gets the money on 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 L_2, such that the locked amount is no smaller than the channel capacity on L_1 to cover the maximum potential loss of a victim.

This approach has two issues. First, it requires (the smart contract of) L_2 to monitor all events on L_1 and retrieve evidence to resolve the dispute. While there are existing attempts to enable communication across blockchains [46], they introduce significant additional cost and with open challenges [47]. Second, it requires additional deposit on 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 veri able \key escrow" service on L_2 to keep and release some secret (for example, by using [48]) required to close the channel.

In particular, each payment in the channel is created through a veri able 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 veri able key escrow service (rather than locking additional signi cant 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 to update the secret for each new payment through the smart contract on L_2, making it expensive, slow, and non-scalable.

3.2.3 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 veri ed before being accepted.

In the channel establishment phase, each channel party executes $EncGen(\,)$ to generate a pair of decryption key and encryption key, and share the output with the distributed and veri able key escrow service. The escrow service will include a commitment on the received secret in its smart contract with the two channel parties on L_2.

Each of the channel parties performs EncSign( ) 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 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

KeyUpdate( ), and creates an encrypted signature on the new payment by performing EncSign( ) with the signing key and the updated encryption key.

To close a channel, channel parties share their latest decryption key with each other, perform DecSig( ) 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 e cient 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 L_2. This can be achieved by using standard solutions, such as a time-lock contract. We defer more details to the next section.

3.3 Formal AuxChannel Description

3.3.1 System Model

We consider two blockchains, L_1, which does not support script language, and 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 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 su cient coins on L_1 and L_2 in participating AuxChannel. We assume that they are always online, even though this assumption can be removed by deploying watchtower-like services [49; 50; 51]. We assume a distributed and veri able key escrow service (KES), such as [48], on L_2. The KES keeps secret keys for its clients and releases the key when a pre-de ned 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.

3.3.2 Formalized AuxChannel

Let P_A and P_B be the two channel parties with signing-veri cation key-pairs $(sk_A; pk_A)$ and $(sk_B; pk_B)$ on L_1, and $(sk'_A; pk'_A)$ and $(sk'_B; pk'_B)$ on L_2, respectively, and they have exchanged public keys with each other. Let $Tx := InjjOut$ be the construction of the transaction on 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 L_1. To initialise the establishment, each of the parties generates initial decryption keys $(dk^0_P; \hat{dk}^0_P)$ and encryption key ek^0_P by executing EncGen( ), where $P \in \{A; B\}$. They then interact with KES to share their $(cid; dk^0_P; ek^0_P)$, where cid is a unique identi er of the channel. KES veri es the received messages and records $(cid; ek^0_A; ek^0_B)$ on L_2 as a con rmation 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 con rmation from KES:

1. both channel parties create a funding transaction

$$Tx_f := (pk_A : bal_A^0; pk_B : bal_B^0)||(pk_M : bal_M)$$

to transfer their balance (bal_P^0 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 $\hat{}_P^0$ EncSign($sk_P; ek_P^i; Tx_c^0$) over a newly created commitment transaction

$$Tx_c^0 := (pk_M : bal_M)||(pk_A : bal_A^0; pk_B : bal_B^0):$$

They keep the funding transaction private and exchange $\hat{}_P^0$ with each other.

2. If the received $\hat{}_P^0$ passed the verification EncVerify($pk_P; ek_P^0; \hat{}_P^0; Tx_c^0$), they exchange locally signed Tx_f and post the cross-signed Tx_f on L_1 indicating the completion of channel establishment.

Update To update a channel from the channel state index $i-1$ to i, where $i \geq 2 \in \mathbb{N}^+$, both channel parties agree on the commitment transaction Tx_c^i and their encryption keys ($ek_A^i; ek_B^i$). The update phase is presented as follows:

1. Both channel parties update their decryption keys ($dk_P^i; \hat{dk}_P^i$) and encryption key ek_P^i by executing ($dk_P^i; \hat{dk}_P^i; ek_P^i$) $\leftarrow$ KeyUpdate(dk_P^{i-1}). They also create the proof of encryption keys' consecutiveness $P_P^i \leftarrow$ ConsProof($dk_P^{i-1}; dk_P^i$), an encrypted signature $\hat{}_P^i \leftarrow$ EncSign($sk_P; ek_P^i; Tx_c^i$) over a newly created commitment transaction

$$Tx_c^i := (pk_M : bal_M)||(pk_A : bal_A^i; pk_B : bal_B^i);$$

and exchange message ($ek_P^i; \hat{}_P^i; P_P^i$);

2. If the received $(ek^i_{\mathsf{P}}; \wedge^i_{\mathsf{P}}; P^i_{\mathsf{P}})$ passed the two verifications $\mathrm{EncVerify}(pk_{\mathsf{P}}; ek^i_{\mathsf{P}}; \wedge^i_{\mathsf{P}}; Tx^i_c)$ and $\mathrm{ConsVerify}(ek^{i\ 1}_{\mathsf{P}}; ek^i_{\mathsf{P}}; P^i_{\mathsf{P}})$, where $ek^{i\ 1}_{\mathsf{P}}$ is received and stored from the counter-party at the previous state, they create and exchange a signature σ^{0i}_{P} over the channel state index i and both channel parties' encryption keys $(ek^i_A; ek^i_B)$ by using sk^0_{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 $\sigma^i_{\mathsf{P}} \leftarrow \mathrm{DecSign}(\hat{dk}^i_{\mathsf{P}}; \wedge^i_{\mathsf{P}})$ on Tx^i_c, where $\hat{dk}^i_{\mathsf{P}}$ is the truncated dk^i_{P} (defined in Section 3.1). 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 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 L_2, which starts a timer for the counter party to react before timeout. Assuming that 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 $(\sigma^{0i}_A; \sigma^{0i}_B; i; ek^i_A; ek^i_B)$.

Before the timeout occurs, if the index i represents the latest state of the channel, P_A releases her decryption key dk^i_A by posting a release transaction $Tx^i_{r;A}$; otherwise, P_A uploads an update transaction $Tx^j_{up;A}$, where $j > i$, and KES releases dk^0_B on L_2. Upon timeout, if P_A did not have any response, KES releases dk^0_A.

If P_A has released her decryption key dk^i_A within the timer, P_B can perform $\mathrm{DecSign}(\hat{dk}^i_{\mathsf{P}}; \wedge^i_{\mathsf{P}})$ to derive the commitment transaction Tx^i_c signed by P_A. Similarly, as dk^i_B is already posted on L_2, P_A can also derive the commitment transaction Tx^i_c signed by 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_P^0 (for P 2 $fA; Bg$) on L_2 according to the conditions above, then the corresponding counter party can derive the decryption keys $(dk_P^{i'}; \hat{dk}_P^{i'})$ of any state i' 2 $[1; j]$ through $(dk_P^{i'}; \hat{dk}_P^{i'}; ek_P^{i'})$ KeyUpdate$(dk_P^{(i' \ 1)})$, and then derive the commitment transaction $Tx_c^{i'}$ signed by 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.

3.3.3 Security Analysis

Similar to Perun [24], 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 P_A be the adversary A, who has the ability to break the security goals of AuxChannel, we can make use of P_A to break the security of Schnorr-based CVES scheme.

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

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

$$(dk_A^{i'+1}; \hat{dk}_A^{i'+1}; ek_A^{i'}) \quad A_0(dk_A^{i'});$$

$$(dk_A^{i'+1}; \hat{dk}_A^{i'+1}; ek_A^{i'}) \quad \text{KeyUpdate}(dk_A^{i'});$$

$$_A^{i'+1} \quad \text{EncSign}(sk_A; ek_A^{i'+1}; Tx_c^{i'+1});$$

$$P_A^{i'+1} \quad A_1(dk_A^{i'}; dk_A^{i'+1});$$

$$1 \quad \text{EncVerify}(pk_A; ek_A^{i'+1}; \ _A^{i'+1}; Tx_c^{i'+1})$$

$$1 \quad \text{ConsVerify}(ek_A^{i'}; ek_A^{i'+1}; P_A^{i'+1});$$

$$_A^{i'+1} \quad \text{DecSign}(\hat{dk}^{i'+1}; \wedge_A^{i'+1});$$

$$1 \ 8 \ \text{Verify}(pk_A; \ _A^{i'+1}; Tx_c^{i'+1}):$$

Or we can also make use of P_A to create a valid encrypted signature $\wedge_A^{i'+1}$, but P_B cannot derive a valid $_A^{i'+1}$ from $\wedge_A^{i'+1}$, where:

$$(dk_A^{i'}; \hat{dk}_A^{i'}; ek_A^{i'}) \quad \text{KeyUpdate}^{i'}(dk_A^j)_{j \in [0; i' \ 1]};$$

$$(dk_A^{i'+1}; \hat{dk}_A^{i'+1}; ek_A^{i'+1}) \quad \text{KeyUpdate}(dk_A^{i'});$$

$$P_A^{i^0+1} \quad \text{ConsProof}(dk_A^{i^0}; dk_A^{i^0+1});$$

$$1 \quad \text{ConsVerify}(ek_A^{i^0}; ek_A^{i^0+1}; P_A^{i^0+1});$$

$$\wedge_A^{i^0+1} \quad \text{A}(sk_A; ek_A^{i^0+1}; Tx_c^{i^0+1});$$

$$1 \quad \text{EncVerify}(pk_A; ek_A^{i^0+1}; \wedge_A^{i^0+1}; Tx_c^{i^0+1});$$

$$\hat{}_A^{i^0+1} \quad \text{DecSign}(\hat{dk}_A^{i^0+1}; \wedge_A^{i^0+1});$$

$$1 \ 8 \ \text{Verify}(pk_A; \ \hat{}_A^{i^0+1}; Tx_c^{i^0+1}):$$

Thus, P_A breaks *the consecutive veri ability* and *the correctness* of Schnorr CVES scheme.

For the second security goal: guaranteed payout of channel parties. We assume that P_A has the highest balance at an intermediate channel state i^0, and she has the ability to close the channel at state i^0 with an non-negligible possibility. That is P_A can derive $\hat{}_B^{i^0}$, and post a cross-signed $Tx_c^{i^0}$ on L_1. Thus, she steals coins from P_B.

Thus, we can make use of P_A to forge a valid signature $\hat{}_B^{i^0}$ by using pk_B, $ek_B^{i^0}$, $\wedge_B^{i^0}$, and the proof $P_B^{i^0}$, where:

$$\hat{}_B^{i^0} \quad \text{A}(pk_B; ek_B^{i^0}; \wedge_B^{i^0}; P_B^{i^0});$$

$$1 \quad \text{Verify}(pk_B; \ \hat{}_B^{i^0}; Tx_c^{i^0}):$$

We can also make use of P_A to reverse $dk_B^{i^0}$ from dk_B^i, where

$$dk_B^{i^0} \quad \text{A}(dk_B^i);$$

$$\hat{}_B^{i^0} \quad \text{DecSign}(\hat{dk}_A^{i^0}; \wedge_B^{i^0});$$

$$1 \quad \text{Verify}(pk_B;\ {}^{i'}_{B};\ Tx_c^{i'}):$$

No matter which scenario happens, 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. ∎

3.4 Discussion and Conclusion

Payment channel networks (PCN) enable o -chain 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) [14], 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 [52; 53], This is normally guaranteed by making penalty for the malicious [54], 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: $A \$ B \$ C \$ D$. 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-de ned 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 gets paid. Thus, all the intermediate payments are *atomic* and *unlockable*.

Supporting payment channel network in AuxChannel requires addressing two challenges: rst, 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 rst challenge, we give the informal de nition of *locked signature* below. Using Schnorr signature as an example, let $\sigma = (R, s)$ be the *full signature* in AuxChannel, where $s = x \cdot H(X||R||m) + (r + y + z)$, x is the signing key, $X = g^x$ is the corresponding veri cation key, r is a random value, y is the decryption key, z is the locking key, and $R := g^{r+y+z}$. Let $\hat{\sigma} = (\hat{R}, \hat{s})$ be the *encrypted signature*, where $\hat{s} = x \cdot H(R||m) + (r + z)$ and $\hat{R} := g^{r+z}$. Let $\tilde{\sigma} := (\hat{R}, \tilde{s})$ be the *locked signature*, where $\tilde{s} = x \cdot H(R||m) + r$ and $\hat{R} := g^r$. 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 [7; 35], such that the receiver of each payment can force the opening of the commitment and learn the unlocking key only after a pre-speci ed time.

Suppose that Alice wants to pay x coins to Dave via the path of AuxChannel: $A \$ B \$ C \$ D$. 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-speci ed time t_D.

- Dave and Carol make locked signatures $\hat{}_D$ and $\hat{}_C$ on their newest channel balances by using same locking key Z_D respectively.

- Dave and Carol make their signatures θ_D and θ_C on the message $ijjek^i_C jjek^i_D$, where i is the newest channel state index, by using their signing keys on the script-enabled blockchain respectively, and exchange $Enc_{Z_D}(\theta_D)$ and $Enc_{Z_D}(\theta_C)$, where $Enc_{Z_D}(\ )$ can be any encryption algorithms w.r.t to the encryption key Z_D, such that $\theta_D = Dec_{z_D}(Enc_{Z_D}(\theta_D))$, 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 $Z_C = g^{z_D}\ g^{z_C}$, and repeats the above procedure with Bob, who then picks z_B and produces the locking key $Z_B = g^{z_D}\ g^{z_C}\ g^{z_B}$, and repeats the above procedure with Alice.

Second, Dave releases his encrypted signature $\hat{}_D$, and Carol calculates $z_C = \hat{}_D\ \ \hat{}_D$ and produces $\hat{}_C = \hat{}_C + z_D$. Dave and Carol obtain $\theta_C = Dec_{z_D}(Enc_{Z_D}(\theta_C))$ and $\theta_D = Dec_{z_D}(Enc_{Z_D}(\theta_D))$ 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 $\hat{}_D$, 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 $\hat{}_{C'}$ 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 [14], the payment channel network of Bitcoin, has been exploited by some recent works [55; 56], 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 the further research on them is covered in Chapter 4.

In conclusion, we are excited to present AuxChannel, the first bi-directional payment channel protocol designed specifically for scriptless blockchains. This innovative protocol enables duplex payments in a channel that has an unlimited lifespan. Our design approach represents a new direction for enabling payment channels with scriptless blockchains, which we believe has the potential to transform the industry. Moreover, we see CVES as an independent interest, as it has the potential to be applied to other applications beyond payment channels.

Chapter 4

MoNet

The section presents MoNet, which is described as the first bi-directional payment channel network with unlimited lifetime for Monero. It is fully compatible with Monero without requiring any modification of the current Monero blockchain. MoNet preserves transaction fungibility, which means that transactions over MoNet and Monero are indistinguishable, and it guarantees anonymity of Monero and MoNet users by avoiding any potential privacy leakage introduced by the new payment channel network. We also propose a new crypto primitive in this section called Verifiable Consecutive One-way Function (VCOF), which allows the generation of a sequence of statement-witness pairs in a consecutive and verifiable way. These statement-witness pairs are one-way, which means that it is easy to compute a statement-witness pair by knowing any of the pre-generated pairs, but hard in the opposite direction. Using VCOF, a signer can produce a series of consecutive adaptor signatures (CAS). Finally, we also propose the generic construction of consecutive adaptor signature as an important building block of MoNet.

4.1 Generlized Consecutive Adaptor Signature

This subsection introduces the Veri able Consecutive One-way Function (VCOF). We extend the construction of consecutive veri ably encrypted signature (CVES) to a generic construction called generalized CAS by using VCOF. Moreover, we apply CAS to 2-party linkable ring adaptor signature denoted by 2-party consecutive linkable ring adaptor signature (2P-CLRAS), which is key building block of MoNet.

4.1.1 Verifiable Consecutive One-Way Function

We propose the concept of veri able consecutive one-way function (VCOF), which produces a new statement-witness pair from the last generated statement-witness pair (*consecutive*), the last generated statement-witness pair is referred to as the \ancestors" pair, and the new statement-witness pair generation can be public veri ed (*veri able*). Also it is e cient to compute the new statement-witnesses pair from its \ancestors" pair, but is computationally hard to compute in an opposite ow (*one-wayness*). A VCOF can be e ciently used in a stateful system, such as a bi-directional payment channel protocol, where a state is revoked if the channel capacity is reallocated. VCOF can be used for some stateful schemes requiring a sequence of interactions and updates.

Let $f_R : \ _w \ ! \ _s$ be the function to produce a statement $Y \ 2 \ _s$ by inputting the witness $y \ 2 \ _w$. We assume a secure one-way function $f_c : ((\ _s; \ _w); pp) \ ! \ (\ _s; \ _w)$ (f_c is also called the consecutive function) to generate a new statement-witness pair $(Y^0; y^0)$ by taking the given statement-witness pair $(Y; y)$ and a public parameter pp, where pp would be a value or an operation, as inputs. Let P_c and V_c be a proof system, where $P_c((Y; y); (Y^0; y^0))$ generates a proof P on the two statement-witness pairs $(Y; y)$ and $(Y^0; y^0)$, and the corresponding veri cation

function $V_c((Y;Y')P)$ outputs 1 to accept the two pairs $(Y;y)$ and $(Y';y')$ or 0 to reject. The three functions f_c, P_c and V_c satisfy the following equations:

$$(Y';y') \quad f_c((Y;y);pp)$$
$$1 \quad V_c((Y;Y');P_c((Y;y);(Y';y')))$$

The proof system $(P_c;V_c)$ is secure, if the following holds:

- *Correctness*: given two statement-witness pairs $(Y;y)$ and $(Y';y')$, where $(Y';y')$ $f_c((Y;y);pp)$, anyone can produce a proof P $P_c((Y;y);(Y';y'))$, where 1 $V_c((Y;Y');P)$.
- *Soundness*: once a proof P on $(Y;y)$ and $(Y';y')$ is accepted by the veri - cation function V_c, then $(Y';y')$ $f_c((Y;y);pp)$.
- *Zero-knowledge*: the inputs of V_c, $((Y;Y');P)$, do not leak any information of $(y;y')$.

The formal de nition of veri able consecutive one-way function is presented as follows:

Definition 6 (Veri able Consecutive One-way Function (VCOF))**.** *A veri able consecutive function w.r.t a hard relation R consists of a tuple of three algorithms* $VCOF_R$*:=(SWGen, NewSW, CVrfy):*

- $(Y^0;y^0)$ *SWGen*( )*: the statement-witness generation algorithm takes the security parameter as input, and produces a statement-witness pair* $(Y^0;y^0)$.
- $((Y^{i+1};y^{i+1});P^{i+1})$ *NewSW*$((Y^i;y^i);pp)$*: the new statement-witness algorithm takes a statement-witness pair* $(Y^i;y^i)$ *and a public parameter pp as inputs, and produces a new statement-witness pair* $(Y^{i+1};y^{i+1})$ *and a proof P, where i 2* N.

- 1=0 $CVrfy(Y^i; Y^{i+1}; P^{i+1})$: *the veri cation algorithm takes two statements* $(Y^i; Y^{i+1})$ *and the proof* P^{i+1} *as inputs, and outputs* 1 *(acceptance) or* 0 *(rejection).*

SWGen( )	NewSW($(Y^i; y^i); pp$)	CVrfy($(Y^i; Y^{i+1}); P^{i+1}$)
y^0 $_w$;	$(Y^{i+1}; y^{i+1})$ $f_c((Y^i; y^i); pp)$;	return 1=0
Y^0 $f_R(y^0)$;	P^{i+1} $P_c((Y^{i+1}; y^{i+1}); (Y^i; y^i))$;	$V_c(Y^i; Y^{i+1}; P^{i+1})$
return $(Y^0; y^0)$	return $((Y^{i+1}; y^{i+1}); P^{i+1})$	

Figure 4.1: Veri able Consecutive One-way Function (VCOF)

Comparison between VCOF and Forward Secrecy. In a signature scheme, forward secrecy FS guarantees that a forward secure signature in the past remains secure even if the current key is lost. The main idea is to divided the lifetime of the public key into T intervals, and in each time interval the same public key corresponds to di erent secret keys. A current secret key can be used to derive the secret key in the future, but not the past. Thus, even a compromise of the current secret key dose not enable the adversary to forge signatures pertaining to the past. The main di erence between VCOF and FS is that VCOF updates a statement-witness pair, while FS only updates the private key and the public key remains unchanged.

Security Model of VCOF. The veri able consecutive one-way function should satisfy three properties, *consecutiveness*, *consecutive veri ability*, and *one-wayness*:

- *Consecutiveness*: a new statement-witness $(Y^{i+1}; y^{i+1})$ is derived from the given statement-witness by using the function $f_c((Y^i; y^i); pp)$.
- *Consecutive veri ability*: given two statements and an associated proof, anyone can be convinced that two witnesses associated to the given state-

ments are consecutive, meaning that one of the statement-witness pairs is generated from the other.

- *One-wayness*: given a newly generated statement-witness pair, no one can derive the previous statement-witness.

We then provide the formal de nition of three properties and the corresponding proof below.

Definition 7 (Consecutiveness). *The two statement-witness pairs* $(Y^i; y^i)$ *and* $(Y^{i+1}; y^{i+1})$ *are consecutive, if*

$$(Y^0; y^0) \quad SWGen(\ )$$
$$((Y^{i+1}; y^{i+1}); P^{i+1}) \quad NewSW((Y^i; y^i); pp)$$

Definition 8 (Consecutive Veri ability). *A VCOF is consecutive veri able, if for every probabilistic polynomial-time adversary* A*, the probability of running the experiment* $CVrfy_{A;R}$*, de ned in Fig. 4.2, is* $Pr[CVrfy_{A;R}(\ ) = 1] \quad negl(\ )$*, where the probability of* $Y^{i+1} = Y^{i+1}$ *is a negligible value* $\frac{1}{j \quad Y j}$*.*

Definition 9 (One-wayness). *A VCOF satis es one-wayness for a PPT adversary* A*, if the advantage* $Adv = Pr[cOneway_{A;R}(\ ) = 1] \quad negl(\ )]$*, where the experiment* $cOneway_{A;R}$ *is de ned in Fig 4.2.*

The consecutiveness of the VCOF is straightforward. We skip the proof of consecutiveness and present proofs of *consecutive veri ability* and *one-wayness*.

Lemma 4.1.1. *A VCOF is consecutive veri able, if the embedded proof system* $(P_c; V_c)$ *is secure.*

Proof. Suppose that there is a PPT adversary A, who can break the consecutive veri ability security of VCOF, we construct a simulator B to break the

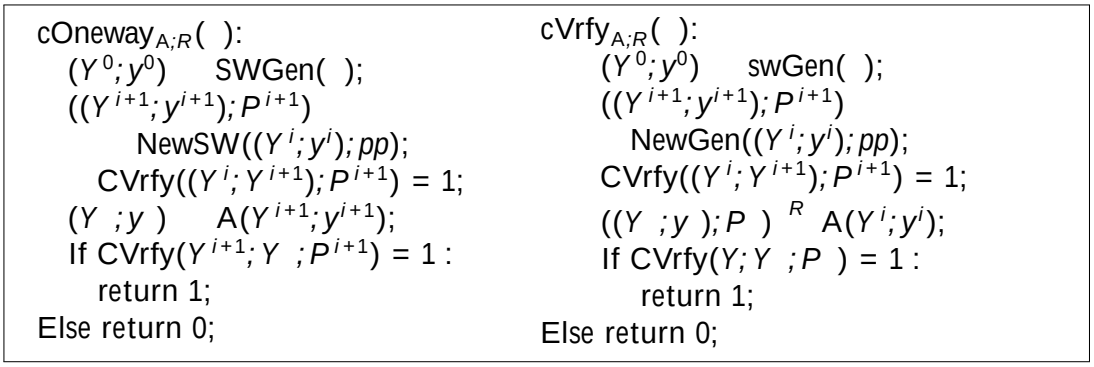

Figure 4.2: Experiments cOneway$_{A;R}$ and cVrfy$_{A;R}$( )

embedded proof system of VCOF. First, there is a given statement-witness pair $(Y;y)$ to B, and B forwards $(Y;y)$ to A.

For all the consecutiveness oracle queries of a q_o-tuple $f(Y';y')g$ from A on a statement-witness pair $(Y;y)$ 2 $f(Y';y')g$, B picks a random $(Y^0;y^0)$ from their corresponding domain and constructs a P by using $(Y;y)$ and $(Y^0;y^0)$, then returns $((Y^0;y^0);P\ )$ to A. We have $(Y^0;y^0)$ $f_c((Y;y);pp)$, and the probability of $(Y^0;y^0)$ equals to $(Y^0;y^0)$ is negligible.

If A can output $((Y^0;y^0);P\ )$, which satis es 1 $V_c((Y;Y^0);P\ )$, B can produce not only a P on $(Y;y)$ and $(Y^0;y^0)$, but also a P on $(Y;y)$ and $(Y^0;y^0)$, which breaks the soundness requirement of the embedded proof system. ∎

Lemma 4.1.2. *A VCOF is one-way, if the embedded consecutive function f_c is one-way.*

Proof. Suppose that there is a PPT adversary A, who can break the one-wayness of VCOF, we construct a simulator C to break the security of the embedded consecutive function f_c. First, given a statement-witness pair $(Y^0;y^0)$, which is generated from an pair $(Y;y)$, where y is unknown to C, who forwards $(Y^0;y^0)$ to A. C forwards $(Y^0;y^0)$ to A.

For all the consecutiveness oracle queries of a q_c-tuple $f(Y'^0;y'^0)g$ from A on a statement-witness pair $(Y^0;y^0)$ 2 $f(Y'^0;y'^0)g$, C picks $(Y\ ;y\ )$ randomly from their

corresponding domain, and return them to A.

If A outputs a $((Y\ ;y\ );P\ )$ on a given $((Y^{0};y^{0});P;\ \ )$, where $(Y^{0};y^{0})$ 2 $f(Y^{0};y^{0})g$ and 1 CVrfy$((Y\ ;Y^{0});P\ )$. It implies that C can output a solution $(Y\ ;y\ )$ for the given instance $(Y^{0};y^{0})$, where $f_c((Y\ ;y\ );pp) = (Y^{0};y^{0})$. Thus, C breaks the security of the embedded one-way function f_c. ∎

Application of VCOF. VCOF can be combined to achieve some useful functionalities. We show one of the applications - consecutive adaptor signature (CAS) in Section 4.1.2, which combines adaptor signature with VCOF. Brie y, CAS allows a signer to pre-sign some messages by using VCOF, and revealing one of the intermediate witnesses makes an exposure of the following signatures.

4.1.2 Generalized Consecutive Adaptor Signature

Consecutive adaptor signature (CAS) allows a signer to make multiple adaptor signatures $\hat{} = f\hat{}^{0};:::;\hat{}^{i_c}g$, where i_c 2 N, on some messages $\mathbf{m} = fm^{0};:::;m^{i_c}g$ by using a sequence of witnesses $\mathbf{y} = fy^{0};:::;y^{i_c}g$. Section 3.1 proposes the concept of the consecutive veri ably encrypted signature. This section provides a generic transformation from adaptor signature and VCOF to CAS, which is the generalized version of CVES.

CAS uses VCOF to update the embedded statements $(Y^{i_c};Y^{i_c+1})$ of two adaptor signatures. Once a witness is revealed, the corresponding signature and the following signatures are adaptable. Remark that, the rst statement-witness pair is generated by executing SWGen( ), and each of witnesses-statement pairs is generated from its ancestor by executing NewSW( ). Let i_c 2 N and i_c^{0} 2 $\mathbb{N}^{+}$ be the index number of signing sessions. Following the generic signature and adaptor signature construction from Section 2.5, our generic construction of consecutive adaptor signature is presented in Algorithm 1.

Since the security model of CAS follows directly from CVES in Section 3.1,

Algorithm 1: Generic Transformation from aSIG and VCOF to Consecutive Adaptor Signature (CAS).

Procedure $SetUp(\ )$:
  return $param$
Procedure $Gen(\ )$:
  $sk \quad _{sk}(param)$;
  $vk = f_{vk}(sk)$;
  return $(sk; vk)$
Procedure $PSign_{sk}(m^{i_c}; Y^{i_c})$:
  $(R_{pre}; St) \quad P_1(sk)$;
  $R_{sign} := f_{shift}(R_{pre}; Y^{i_c})$;
  $h^{i_c} := \mathsf{H}(R_{sign}; m^{i_c})$;
  $\hat{s}^{i_c} \quad P_2(sk; R_{pre}; h^{i_c}; St)$;
  $\hat{\wedge}^{i_c} := (h^{i_c}; \hat{s}^{i_c})$;
  return $\hat{\wedge}^{i_c}$
Procedure $PVrfy_{vk}(m^{i_c}; Y^{i_c}; \hat{\wedge}^{i_c})$:
  $(h^{i_c}; \hat{s}^{i_c}) := \hat{\wedge}^{i_c}$;
  $\hat{R}_{pre} := V_0(pk; h^{i_c}; \hat{s}^{i_c})$;
  $\hat{R}_{sign} := f_{shift}(\hat{R}_{pre}; Y^{i_c})$;
  return $h^{i_c} \stackrel{?}{=} \mathsf{H}(\hat{R}_{sign}; m^{i_c})$
Procedure $Vrfy_{vk}(m^{i_c}; \ ^{i_c})$:
  $(h^{i_c}; s^{i_c}) := \ ^{i_c}$;
  $R_{sign} := f_{shift}(pk; h^{i_c}; s^{i_c})$;
  return $h^{i_c} \stackrel{?}{=} \mathsf{H}(R_{sign}; m^{i_c})$
Procedure $Adapt_{vk}(\hat{\wedge}^{i_c}; y^{i_c})$:
  $(h^{i_c}; \hat{s}^{i_c}) := \hat{\wedge}^{i_c}$;
  $s^{i_c} = f_{Adapt}(\hat{s}^{i_c}; y^{i_c})$;
  return $(h^{i_c}; s^{i_c})$;
Procedure $Ext_{vk}(\ ^{i_c}; \hat{\wedge}^{i_c}; Y^{i_c})$:
  $(h^{i_c}; s^{i_c}) := \ ^{i_c}$;
  $(h^{i_c}; \hat{s}^{i_c}) := \hat{\wedge}^{i_c}$;
  return $f_{ext}(s^{i_c}; \hat{s}^{i_c})$
Procedure $SWGen(\ )$:
  $y^0 \quad _w$;
  $Y^0 \quad f_R(y^0)$;
  return $(Y^0; y^0)$
Procedure $NewSW((Y^{i_c^0}; y^{i_c^0}); pp)$:
  $(Y^{i_c^0+1}; y^{i_c^0+1}) \quad f_c((Y^{i_c^0}; y^{i_c^0}); pp)$;
  $P^{i_c^0+1} \quad P_c((Y^{i_c^0}; y^{i_c^0}); (Y^{i_c^0+1}; y^{i_c^0+1}))$;
  return $((Y^{i_c^0+1}; y^{i_c^0+1}); P^{i_c^0+1})$
Procedure $CVrfy((Y^{i_c}; Y^{i_c+1}); P^{i_c+1})$:
  return $1 \stackrel{?}{=} 0 \quad V_c((Y^{i_c}; Y^{i_c+1}); P^{i_c+1})$

we omit it and the corresponding proof.

4.1.3 2-Party Consecutive Linkable Ring Adaptor Signature

A linkable ring signature (LRS) [57] preserves a signer's anonymity by hiding the signer's veri cation key vk among a set of veri cation keys $\vec{vk} = \{vk_1; \ldots; vk_n\}$. LRS scheme can be extended to 2-party linkable ring signature (2P-LRS) [35] scheme, which allows two signers to jointly sign a linkable ring signature by using

their partial signing keys associated to a single public key, where the identity of the two signers is considered as a single signer for an observer and the corresponding veri cation algorithm is same as a LRS scheme.

We show that our generic consecutive adaptor signature can be applied to the 2-party linkable ring adaptor signature with aggregatable public keys denoted by 2P-CLRAS. It allows two signers with single aggregatable public key to generate a sequence of linkable ring adaptor signature together. Let A and B be two signers in a 2P-LRS scheme, and P 2 $fA; Bg$ and P^{0} = $fA; Bg$ n P. Let be a group operation in $_{r}$, and $_{R}$ be a group operation in the domain of R. The inverse functions are de ned in the corresponding group operations and $_{R}$. We present a generic construction of 2P-CLRAS in Algorithm 2.

4.2 Description of MoNet

The goal of this work is to build a payment channel network for Monero, a fully scriptless blockchain with strong privacy requirements on the payment channel (network) construction. This section proposes MoChannel, a bi-directional payment channel for Monero, and further constructs the payment channel network, MoNet, built upon MoChannel. For simplicity, we assume that Alice has frequent transactions with Bob, so they maintain a channel. However, Alice needs to pay Carol causally, through Bob.

4.2.1 Overview

For the ease of understanding, this section presents high-level description of MoNet. Alice and Bob join MoNet collaboratively by funding a MoChannel with some coins. They transact with each other by re-distributing their balances within the established channel, and exit MoNet by recording their nal balances

Algorithm 2: Generic 2-Party Consecutive Linkable Ring Adaptor Signature (2P-CLRAS) from both 2P-LRS and GCAS among n_s signers by using VCOF.

Procedure *Setup*( ):
  de ned hash $H_{lrs} : f0;1g^{*} \to {}_{c}$;
  return param;
Procedure *JGen*( ):
  receiving vk_{P^0} from P^0 and generate $vk = vk_P \quad vk_{P^0}$;
  return $(vk; sk_P)$
Procedure $PSign_{vk}(sk_P; m^{i_c}; Y^{i_c})$:
  $r_P \quad s \quad r$;
  $R_P = P_1(r_P; vk)$;
  receiving R_{P^0} from P^0 and generate $R = R_P \quad_R R_{P^0} \quad_R Y^{i_c}$;
  $c = H_{lrs}(m; R; vk)$;
  $\hat{z}_P := P_2(sk_P; vk; r_P; c)$;
  receiving $\hat{z}^{i_c}_{P^0}$ from P^0 and generate $\hat{z}^{i_c} = \hat{z}^{i_c}_P \quad \hat{z}^{i_c}_{P^0}$;
  return $\hat{\sigma}^{i_c} = (\hat{z}^{i_c}; c)$
Procedure $PVrfy_{vk}(m^{i_c}; \hat{\sigma}^{i_c}; Y^{i_c})$:
  parse $\hat{\sigma}^{i_c} = (\hat{z}; c)$;
  $R = V_0(vk; c; \hat{z}; m^{i_c}) \quad_R Y^{i_c}$;
  if $c \neq H_{lrs}(m; R; vk)$ **then**
    return 0;
  return 1;
Procedure $Vrfy_{vk}(m^{i_c}; \quad {}^{i_c})$:
  parse $\quad {}^{i_c} = (z; c)$;
  $R = V_0(vk; c; z)$;
  if $c \neq H_{lrs}(m^{i_c}; R; vk)$ **then**
    return 0;
  return 1;
Procedure $Adapt_{vk}(\hat{\sigma}^{i_c}; y^{i_c})$ *and* $Ext_{vk}(\hat{\sigma}^{i_c}; \quad {}^{i_c}, Y^{i_c})$:
  parse $\hat{\sigma}^{i} := (\hat{z}; c)$;
  $z = \hat{z} \quad y^{i_c}$;
  $\quad {}^{i_c} = (z; c)$;
  return $\quad {}^{i_c}$;
Procedure *SWGen*( ):
  pick $y^0_P \; 2 \quad {}_t$;
  $Y^0_P = g^{y^0_P}$;
  receiving $Y^0_{P^0}$ from P^0 and compute $Y^0 = Y^0_P \quad_R Y^0_{P^0}$;
  return $(Y^0; (Y^0_P; y^0_P))$;
Procedure $NewSW((Y_P^{i^0_c}; y_P^{i^0_c}); pp)$:
  $(Y_P^{i^0_c+1}; y_P^{i^0_c+1}) \quad f_c((Y_P^{i^0_c}; y_P^{i^0_c}); pp)$;
  $P_P^{i^0_c+1} \quad P_c((Y^{i^0_c}P; y_P^{i^0_c}; (Y_P^{i^0_c+1}; y_P^{i^0_c+1}))$;
  receiving $Y_{P^0}^{i_c}; P_{P^0}^{i^0_c+1}$ from P^0 and compute $Y^{i_c} = Y_P^{i_c} \quad_R Y_{P^0}^{i_c}$ and $P^{i_c} = P_P^{i_c} \quad_R P_{P^0}^{i_c}$;
  return $(Y^{i^0_c+1}; (Y_P^{i^0_c+1}; y_P^{i^0_c+1}); P^{i^0_c+1})$;
Procedure $CVrfy((Y_P^{i_c}; Y_P^{i_c+1}); P_P^{i_c+1})$:
  if $V_c((Y_P^{i_c}; Y_P^{i_c+1}); P_P^{i_c+1}) = 1$ **then**
    return 1;
  return 0;

on Monero. By using MoNet, Alice can pay Carol, who has no channel with her but does have one with Bob. By leveraging 2P-CLRAS, it is hard for each of the

channel parties to revert a history state from the latest state, but easy to infer the latest state if the counterparty tries to close the channel with a history state.

To establish a MoChannel, Alice or Bob executes 2P-CLRAS.SWGen( ) and 2P-CLRAS.JGen( ) collaboratively to generate their partial initial statement-witness pairs associated with a joint initial statement and their partial signing keys associated with an aggregatable public key. Each of them then creates two transactions. The first is funding transaction, which funds a channel by transferring Monero tokens from their private addresses respectively to the aggregatable public key address as the channel capacity. The other is commitment transaction, which spends the output of funding transaction to both parties' private addresses respectively. This transaction guarantees that each channel party can exit MoChannel even if there is no transactions within the channel. Alice and Bob perform 2P-CLRAS.PSign( ) to pre-sign commitment transaction collaboratively, and sign funding transaction and broadcast the signed funding transaction to Monero network. A channel is established between Alice and Bob, once the funding transaction is recorded on-chain.

To update the channel, Alice and Bob preforms 2P-CLRAS.PSign( ) collaboratively on a newly created transaction to redistribute their channel balances, where the difference of their balances indicates the transaction amount.

To close the channel, Alice and Bob exchange their latest witnesses, and adapt a signature from the corresponding pre-signature by executing 2P-CLRAS.Adapt( ). Finally, each of them can close the channel by uploading a transaction with the adapted signature to Monero.

When the channel between Alice and Bob is on a payment path for a multi-hop payment, they lock a payment collaboratively within their channel first and unlock the payment only if the receiver in their channel paid for the next hop.

4.2.2 Security Properties

We expect the following properties to be guaranteed:

For a single MoChannel:

Guaranteed channel closure. Normally, closing MoChannel requires both Alice and Bob's cooperation. This property requires that either Alice or Bob can close the channel unilaterally, which prevent the channel from being locked forever.

Guaranteed payout for honest channel parties. Assuming that there is at least one honest party within a channel, he can withdraw with no less than his latest balance.

On-chain unidenti ability. On-chain transactions cannot be identi ed as being for opening or closing a channel.

For a payment via multi-hop MoChannel:

Atomicity. All the payments on the path are either successful or failed, and no \half paid" scenario exists.

Unlockability. All the on-path payments can be unlocked, even if the receiver does not cooperate.

Sender/Receiver privacy. For successful payments, any intermediary cannot determine if the left (right) neighbor along the path is the actual sender (receiver) or just an honest user connected to the sender (receiver) through a path of non-compromised users.

Path privacy. In the case of a successful payment, malicious intermediaries cannot determine which users participated in the payment aside from their direct neighbors.

4.2.3 Main Construction

MoNet allows users to build payment channels, MoChannel, upon Monero and make payments to a recipient connecting directly or through a path of payment

channels with the sender. This section describes how MoNet works. Moreover, we provide intuitions on how MoNet satisfies these properties, before formally proving them in the next section.

MoChannel (Figure 4.3). To establish a channel, Alice and Bob generate their partial signing keys $\tilde{sk}_A$, $\tilde{sk}_B$ associated with their joint verification key vk_{AB} (line 1). They also generate the initial statement-witness tuples $(S^0;(S^0_A;w^0_A))$ and $(S^0;(S^0_B;w^0_B))$ (line 2) collaboratively and create two transactions: funding transaction $Tx_f := (vk_A : bal^0_A; vk_B : bal^0_B)||(vk_{AB} : bal_C)$ and commitment transaction $Tx^0_c := (vk_{AB} : bal_C)||(vk_{A'} : bal^0_A; vk_{B'} : bal^0_B)$ (line 3). The funding transaction Tx_f indicates that Alice and Bob transfer bal^0_A and bal^0_B (their initial channel balances) from vk_A and vk_B respectively to vk_{AB} as channel capacity bal_C. The commitment Tx^0_c spends the output of Tx_f to reallocate the channel capacity from their joint account vk_{AB} to $vk_{A'}$ and $vk_{B'}$ [1]. They collaboratively produce pre-signature $\hat{}^0_{\tilde{sk}_A;\tilde{sk}_B}$ over Tx^0_c by using $\tilde{sk}_A$, $\tilde{sk}_B$ and S^0 (line 4), and sign and exchange their signatures $_{sk_A}$ and $_{sk_B}$ on the funding transaction Tx_f by using their signing keys sk_A and sk_B associated with vk_A and vk_B respectively (line 5-6). Each of them can upload the signed funding transaction $(_{sk_A};_{sk_B};Tx_f)$ to Monero, and the channel Ch_{AB} established.

To update the channel, Alice and Bob collaboratively generate a statement S^i, exchange their partial statements and proofs $(S^i_A;P^i_A)$ and $(S^i_B;P^i_B)$, and keep the corresponding witnesses w^i_A and w^i_B secret (line 11). Once the received statement is verified, they collaboratively produce a pre-signature $\hat{}^i_{\tilde{sk}_A;\tilde{sk}_B}$ over $Tx^i_c := (vk_{AB} : bal_C)||(vk_{A'} : bal^i_A; vk_{B'} : bal^i_B)$ by using $(\tilde{sk}_A;S^i_A)$ and $(\tilde{sk}_B;S^i_B)$ (line 12-13).

To close a channel, Alice and Bob exchange their latest witnesses w^i_A and w^i_B (line 14-17), and adapt the corresponding signature $^0_{\tilde{sk}_A;\tilde{sk}_B}$ on Tx^i_c from $\hat{}^0_{\tilde{sk}_A;\tilde{sk}_B}$,

[1] Due to fresh key policy in Monero, $vk_{A'}$ and $vk_{B'}$ are different from vk_A and vk_B respectively.

w_A^i and w_B^i (line 18). Each of Alice and Bob can upload the signed $(\ _{\widetilde{sk}_A;\widetilde{sk}_B}^{i};Tx_c^i)$ to Monero, which closes the channel Ch_{AB} (line 19-20).

Multi-hop Payments (Figure 4.4). For the sake of clarify, we describe a scenario that Alice wants to transfer x coins to Carol via Bob, who has channels with Alice and Carol respectively.

As Monero does not support any script execution, to lock a payment within a channel in MoNet, both channel parties jointly create an incomplete pre-signature, which is an adaptor signature concealed by a secret value, by using a locking key (public key), their newly generated statements and their partial signing key associated with the joint address in the output of funding transaction on a new commitment transaction. To simplify notations, we assume that both channels Ch_{AB} and Ch_{BC} are in the $(i \quad 1)$-th state when processing a multi-hop payment. The multi-hop payment in MoNet is processed as follows: 1) **Setup**. Alice, the sender of the multi-hop payment, generates some locks and unlocking keys, $(Y_B; y_b)$ and $(Y_C; y_c)$, for each intermediate channel, and sends $((Y_B; Y_C); y_b)$ and $((Y_C; 0); y_c)$, to Bob and Carol respectively (line 1-7). 2) **Lock**. In each on-path channel, for example channel Ch_{AB}, Alice and Bob collaboratively generate their new statements, witnesses and the corresponding proofs $(S_{AB}^i; (S_A^i; w_A^i); P_A^i)$ and $(S_{AB}^i; (S_B^i; w_B^i); P_B^i)$, and an incomplete pre-signature $\hat{}_{AB}^i$ on $Tx_{c;AB}^i$ by using Y_B, which locks channel Ch_{AB} (line 8, 10). The process is identical to Bob and Carol, they collaboratively lock the channel Ch_{BC} at state i by using the lock Y_C (line 9, 11). 3) **Unlock**. Carol adapts $\hat{}_{vk_{BC}}^i$ from $\hat{}_{vk_{BC}}^{\ i}$ by adding y_c, and sends $\hat{}_{vk_{BC}}^i$ to Bob (line 12-13), who then calculates y_c from $\hat{}_{vk_{BC}}^{\ i}$ and $\hat{}_{vk_{BC}}^i$, and recovers $\hat{}_{vk_{AB}}^i$ from $\hat{}_{vk_{AB}}^{\ i}$ by adding y_b and y_c (line 16-17). Finally, all channels updated and the payment succeed.

We discuss that how MoNet satis es the properties in Section 4.2.2 below.

To ensure the **sender/receiver privacy** and **path privacy**, MoNet leverages

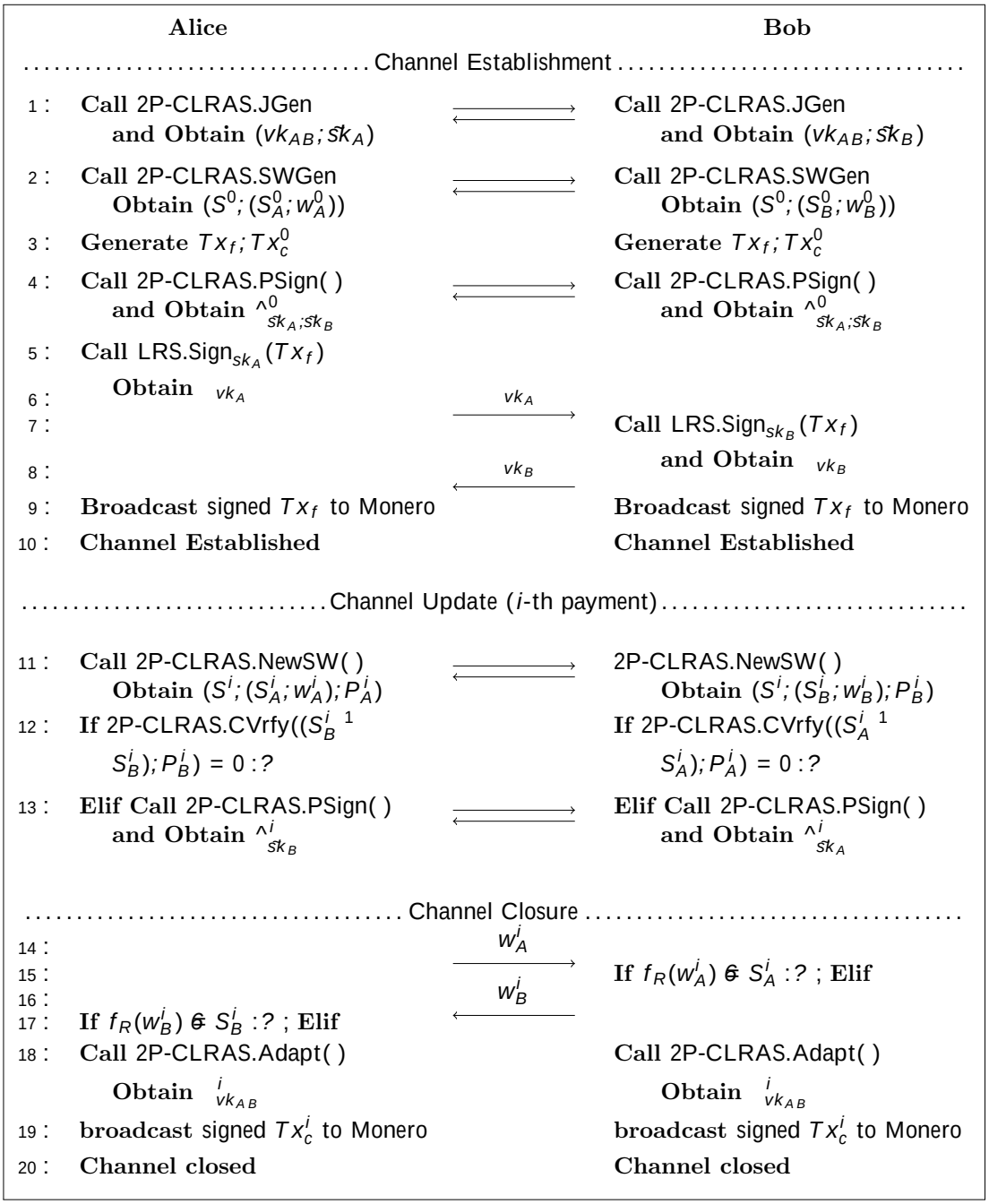

Figure 4.3: MoChannel. The double arrows (e.g. line 1, 2, 4, 11, 13) denote that both channel parties execute an interactive algorithm collaboratively, and the number of interactions are decided by the corresponding algorithm.

Figure 4.4: Multi-hop Payment in MoNet

anonymous multi-hop locks (AMHLs [18], see Section 2.3.1) scheme, which allows the sender of a multi-hop payment to generate locks and deliver messages via an anonymous communication channel [58] to each on-path user.

Resolve the Dispute. MoNet also provides solutions for resolving some potential disputes, e.g., any of channel parties does not release his witness at the channel closure phase.

We employ a distributed Key Escrow Service following AuxChannel paradigm to provide the *guaranteed channel closure*, *guaranteed payout* and *unlockability* properties. Key Escrow Service can be implemented on a script-enabled platform, for example on Ethereum. Alice and Bob escrow their initial witnesses to Key Escrow Service among n_e escrowers by using publicly veri able secret sharing scheme [59; 60; 61] before establishing a channel. Therefore, each channel party can reconstruct the other's initial witness from n_e escrowers and further compute the other's latest witness and recover a valid signature on the latest commitment transaction within the channel.

To ensure the **guaranteed channel closure**, each channel party can propose a channel dispute request to KES and reconstruct the counterparty's initial witness from escrowers if he does not cooperate to close the channel. A channel dispute request contains a timer ^{i} and both channel parties's statements $(S^{i}_{\mathrm{P}}; S^{i}_{\mathrm{P}'})$ at channel state i. Notice that, both channel parties agree on and cross-sign over a timer and two puzzles with the signature scheme employed by KES at each channel update phase.

To ensure the **unlockability** when routing a multi-hop payment, the locked payment can be cancelled within each on-path channel or processed by calling KES if there is a dispute. We follow the scenario that Alice pays Carol x Monero tokens via Bob, and a successful payment updates both Ch_{AB} and Ch_{BC} from state i 1 to i. Alice and Bob, which are the senders in channel Ch_{AB} and Ch_{BC}

respectively, can redeem their coins by updating channel Ch_{AB} or Ch_{BC} to state $i + 1$ with their balances at state i 1. This requires the collaboration between both channel parties in channel Ch_{AB} or Ch_{BC} respectively. Moreover, if Bob or Carol does not cooperate in channel Ch_{AB} or Ch_{BC} to cancel a locked payment, Alice or Bob can redeem their coins by proposing a dispute request to KES. The channel under dispute will be closed. Usually, only the channel in the last hop is close, when su ering an unexpected locking time due to the malicious recipient (Carol), who does not unlock a payment. So other on-path participants (Alice and Bob) are rational and still want to make transactions within their channels (Ch_{AB}), they would cancel the payment within their channels.

To satisfy the **atomicity**, MoNet ensures that there is no \half paid" scenario happens. Similar to LN [14], the atomicity of a multi-hop payment is guaranteed by the cascade timers, where $^{i}_{AB} > ^{i}_{BC}$, and the hard relationship of each lock. As both parties can rede ne their channel timer at each channel state, it guarantees the cascade time-lock requirement.

As both channel parties expose their initial statement-witness pairs to KES, anyone who knows both parties' witnesses can extract an adaptor signature once the corresponding signature is recorded on-chain, which will be identi ed as a closing channel transaction. This is not desired and goes against the **on-chain unidentifiability** property. To guarantee this property, both parties can re-randomize their witnesses and statements by setting $S^{\prime 0}_{\mathrm{P}} = S^{0}_{\mathrm{P}}$ g^{r} and $w^{\prime 0}_{\mathrm{P}} = w^{0}_{\mathrm{P}} + r$ for some random value r chosen uniformly over the randomness domain and generator g.

4.3 Security Model and Analysis of MoNet

This section formally de nes the ideal functionality of MoNet, provides the analysis of how the ideal functionality satis es the security properties described in Section 4.2.2, and proves that MoNet is secure under universal composable (UC) framework introduced by Canetti [62].

4.3.1 Security Model

UC-Security. Let $EXEC_{;A;E}$ be the ensemble of the outputs of the environment E when interacting with the adversary A and parties running the protocol (over the random coins of all the involved machines).

Definition 10 (UC-Security). *A protocol UC-realizes an ideal functionality* F *if for any adversary* A *there exists a simulator* S *such that for any environment* E *the ensemble* $EXEC_{;A;E}$ *and* $EXEC_{F;S;E}$ *are computationally indistinguishable.*

Attacker Model. We model participants in our protocol as interactive Turing machines that interact with a trusted functionality F via secure and authenticated channels. We model the attacker A as an interactive Turing machine that all the actions (i.e. incoming and outgoing communication) of P is taken over by A.

Communication Model. We model our system under a synchronous communication network, where a message broadcasted by a participant *P* at time T will be reached to all other participants within T + , and denotes the network latency. In the ideal world, participants do not communicate, but only receive and forward messages from and to the functionality F_{pay}.

Key Escrow Service. Let F_{kes} be a Key Escrow Service functionality that maintains a set of active contract instances, $Ke:=(Ke:id, Ke:key, Ke:timer, {}_{ke:id})$, with some attributes, an unique KES identi er, escrowed key, a timer and a verication function, where the escrowed key is de ned as a tuple $Ke:key := fk_a; k_bg$,

and the function $_{ke:id}$ de nes the validity of a dispute request message. Let P be one of two participants who deploy a KES instance together, and P^0 be the other. All the attributes should be initiated when deploying a KES instance except for the timer *Ke:timer*. The functionality *Ke* has four interfaces. *Initialization* ini-

Initialization: The functionality is initialized by a message $(ke_1;::::;ke_n)$ 2 $K^n{}_0$ that describes the initial KES instances space from the environment E. The functionality stores this tuple.
Deploy:Upon Receiving a message (Deploy, Ke.id, Ke.keys(P)) (for *Ke:id* 2 K): Within 2 , if received the message (AddOk, Ke.id, Ke.keys(P^0)) from P^0, let K := K ^ *Ke*;
If not, ignore this message and send (Ke-not-deplyed,Ke.id) to P.
Setting Timer: Upon Receiving a message (KeSetTimer, *Ke:id*, $Commit_P$, $_{Ke:id}$) for *Ke* 2 K and *Ke:timer* = *?*, if $_{ke:id}(Commit_P)$ = 1, let K := K *n Ke* and set *Ke:timer* = $_{Ke:id}$, then K := K ^ *Ke*:
Option 1): If within time $_{Ke:id}$ + after step 1, received message (Resp, *Ke:id*, $Commit_{P^0}$) from P^0, if $_{ke:id}(Commit_{P^0})$ = 1, terminates *Ke* and sends message (KeTerminated, *Ke:id*) to P and P^0 respectively;
Option 2): After $_{Ke:id}$ + , sends (KeyRelease, *Ke:keys*) to P privately, and (KeTerminated, *Ke:id*) to both P and P^0 respectively.
Otherwise, ignore the setting timer message, and sends a message (KeTimer-not-set, *Ke:id*) to P.
Terminate:Upon Receiving a message (KeTerminate, Ke.id)(for *Ke:id* 2 K) from P, F_{kes} forwards this message to P^0:
1. After the time $_{Ke:id}$ + , let K := K *n Ke*.
2. Within $_{Ke:id}$ + , F_{kes} received a message (KeTerminate-con rm, Ke.id) from P^0, let K := K *n Ke*, and forwards this message to P^0.

Figure 4.5: Functionality F_{kes}

tiates a KES space $K^n{}_0$ received from environment E. *Deploy* creates a new KES instance agreed by two participants involved in this instance within 2 . *Setting Timer* allows one of the participants P to propose a dispute by setting the timer of a KES instance *Ke*. Whether the other participant P^0 respond or not, *Ke* will be terminated after time $_{Ke:id}$. Let $Commit_P$ be a commitment for a channel state, where $_{ke:id}(Commit_P)$ = 1 indicates a valid channel state agreed by both

channel parties and vice versa. *TerminateKe* is called to remove a KES instance from K.

Money Mechanics of Monero. Monero uses unspent transaction output (UTXO) model to maintain a non-negative set of tuples R := $f(r_i; \text{xmr}_i)g$, where r_i denotes an address and xmr_i is amount of Monero tokens in this addresses. Let $_M(r_i; \text{xmr}_i)$ be the veri cation function of $(r_i; \text{xmr}_i)$, where $_M(r_i; \text{xmr}_i) = 1$ indicates that the corresponding transaction is valid and $_M(r_i; \text{xmr}_i) = 0$ stands for an invalid transaction. In the veri cation level, maintaining a Monero ledger is same as maintaining the UTXO set R. We construct the ledger functionality F_M in gure 4.6. F_M maintains R by processing the message *Remove* or *Add*,

Initialization: The functionality is initialized by a message $((r_1; \text{xmr}_1); : : : ; (r_n; \text{xmr}_n))$ 2 $R^n{}_0$ that describes the initial coin distribution and come from the environment E. F_M stores this tuple.
Remove UTXO: Upon Receiving a message (*Remove*, $(r_i; \text{xmr}_i)$) (for $(r_i; \text{xmr}_i)$ 2 R and $1 = _M((r_i; \text{xmr}_i))$), let R := R *n* r_i.
Adding UTXO: Upon Receiving a message $(Add; (r_j; \text{xmr}_j))$ (for $(r_j; \text{xmr}_j)$ 2 R and $1 = _M((r_j; \text{xmr}_j))$), let R := R ^ $(r_j; \text{xmr}_j)$.

Figure 4.6: Ledger Functionality F_M

which remove or add UTXO from or to R if the corresponding proof is veri ed. Otherwise, F_M ignores this message.

In our model, the operation to Key Escrow Service and Monero ledger will be transferred to the modi cation of the KES space and UTXO set R. Participants cannot directly access K and R. Instead the UTXO set R is maintained via the KES protocol and ledger protocol in the real world or via the functionality F_{pay} (see Figure 4.1) in the ideal world.

4.3.2 Ideal Functionality

Let Ch_{AB} := (*Ch:id*, *Ch:Alice*, *Ch:Bob*, *Ch:Bal*, *Ch:State*, *Ch:Ke*, *Ch:*) denote a channel with some attributes, channel identi er, channel parties, channel capacity (consisting of Alice and Bob's balances), channel state (a monotonic increasing number), a KES instance associated with *Ch:id*, and a timer. *Ch:Bal* is de ned as a tuple $(r; xmr)$, which has the same format as an element in R, and *Ch:Bal:r* = *xmr*. To associate a channel with the corresponding Key Escrow Service but not be identi ed by a third party observer, we require *Ch:ke* = *Ke:id* ⊕ *Ch:id*. The functionality F_{pay} maintains a *channel space* C, that consists of some registered MoChannel. The ideal functionality F_{pay} is specified in Table 4.1.

Channel Establishment:

Upon receiving a message m:=(*mc-open*, *Ke*; *Ch*) from *Ch:*P, where *Ke* is initiated with KES id *Ke:id* and P's secret key *Ke:key*(k_P), and *Ch* is initiated with channel id *Ch:id*, both channel parties *Ch:*P and *Ch:*P′, *Ch:*P's balance *Ch:Bal*($r_{Ch:P}$; xmr_P), initial channel state *Ch:State*(0) and KES identi er *Ch:Ke*(*Ke:id*) at time T, F_{pay} removes ($r_{Ch:P}$; xmr_P) from R, and forwards the message m:=(*mc-open*, *Ke*, *Ch*) to *Ch:*P′.

Within time , if F_{pay} receives a message (*mc-open*, *Ke*, *Ch*), which sets *Ke:key*($k_{P'}$) and *Ch:Bal*($r_{Ch:P'}$; $xmr_{P'}$), from *Ch:*P′, then removes $r_{P'}$ from R and adds *Ke* and *Ch* to K and C respectively. Sending a message (*mc-opened*) to parties *Ch:*P and *Ch:*P′ and to the simulator S.

Otherwise, adds $r_{Ch:\mathrm{P}}$ into R and output (*mc-not-opened*) to *Ch:*P.

Channel Update:

Upon receiving a message m:= (*mc-update*, *Ch:id; xmr*) from a channel party P, where *xmr* is the transaction amount. If *Ch* 2 C or $Ch{:}bal(r_{Ch:\mathrm{P}})$ *xmr* < 0 then ignores this message. Otherwise, set $Ch{:}bal(r_{Ch:\mathrm{P}}) = Ch{:}bal(r_{Ch:\mathrm{P}})$ *xmr* , $Ch{:}bal(r_{Ch:\mathrm{P}'}) := Ch{:}bal(r_{Ch:\mathrm{P}'}) + xmr$, and *Ch:State* = *Ch:State* + 1, replace *Ch* in C with newly updated *Ch*, and sends (*mc-updated*, *Ch:id*) to P and P' within .

Payment Routing:

Upon receiving a message *m*:= (*mc-routepay*, $\mathrm{C}'; xmr ; f_{id}g$), where the number of elements in C equals to the number of elements in $f_{id}g$ (e.g. there are *n* elements in C and $f_{id}g$ respectively), if C' 6 C or $Ch{:}bal(r_{Ch:\mathrm{P}})$ *xmr* < 0 for every *Ch* 2 C', then ignore this message. Otherwise, sends (*mc-routepay*, $Ch; Ch'; {}_{Ch:id}; {}_{Ch':id}; xmr$) to Ch':P within , where Ch':P = *Ch:*P' (*Setup*). If ${}_{id'} < {}_{id}$, Ch':P sends a lock payment message m':= (*mc-routepay-lock*, $Ch':id'; xmr ; {}_{id'}$) to F_{pay} (*Lock*). Otherwise, Ch':P ignores this message. The unlock phase is proceed as follows:

1. Within ${}_{Ch:id} + 2$, where ${}_{Ch:id}$ is the largest number in $f_{id}g$, upon receives 2*n* messages (each *Ch* has two messages) m:= (*mc-routepay-unlock*, *Ch:id*), for *Ch* 2 C', F_{pay} sets $Ch{:}bal(r_{Ch:\mathrm{P}}) := Ch{:}bal(r_{Ch:\mathrm{P}})$ *xmr* , $Ch{:}bal(r_{Ch:\mathrm{P}'}) := Ch{:}bal(r_{Ch:\mathrm{P}'}) + xmr$, and *Ch:State* = *Ch:State* + 1, and replaces the old *Ch* in C with the newly updated *Ch*.

2. For a channel *Ch* 2 C', after ${}_{id} + 2$, if received messages m:= (*mc-routepay-cancel*, *Ch:id; Ch:State* + 2) from both *Ch:*P and *Ch:*P', F_{pay} sets *Ch:State* = *Ch:State* + 2, and replaces the old *Ch* in C with newly updated *Ch*.

3. Otherwise, Ch:P sends a message m:= (*mc-close*, $Ch{:}id; Ch{:}State;\ _{id}$) following the interface Channel Closure.

Channel Closure:

Upon receiving a message m := (*mc-close*, $Ch{:}id$, $Ch{:}State$, $Commit_{\mathsf{P}}$, $_{id}$) from Ch:P.

1) If $_{ke:id}(Commit_{\mathsf{P}}) = 1$, F_{pay} calls F_{kes}.Setting Timer with a message (KeSetTimer, $Ch{:}Ke{:}id$, $Commit_{\mathsf{P}}$, $_{Ch:id}$).

Within $_{id}$, F_{pay} receives (*mc-close-Ok*, $Ch{:}id; Ch{:}State$, $Commit_{\mathsf{P}'}$) from Ch:P' and $_{ke:id}(Commit_{\mathsf{P}'}) = 1$, removes Ch from C and Ke from K, adds $r^{'}_{Ch:\mathsf{P}}$ and $r^{'}_{Ch:\mathsf{P}'}$ to R;

Otherwise, F_{pay} receives (*mc-key*, $Ch{:}id$, $Ke{:}id(Ch{:}Ke)$, $Ke{:}key(k_{\mathsf{P}'})$) from F_{kes} and forwards this message to Ch:P, adds $r^{'}_{A}$ with $Ch{:}bal(Ch{:}\mathsf{P}) + Ch{:}bal(Ch{:}\mathsf{P}')$ coin to R, and removes Ch from C and Ke from K.

2) Else if $_{ke:id}(Commit_{\mathsf{P}}) = 0$: ignore this message and send (*mc-not-closed*, $Ch{:}id$) to Ch:P.

Table 4.1: Ideal Functionality F_{pay}

The functionality F_{pay} offers three interfaces: 1) **Channel Establishment** describes an channel establishment between Alice and Bob with a deployed KES instance Ke. This interface adds a channel instance only if both participants confirm the open channel message. Otherwise, F_{pay} ignores this message. Usually, channel update can process two types of payments: a) both channel parties transfer coins for the sake of their own needs, which is proceed in one-round; b) the channel is on the path of a multi-hop payment, which requires a 2-round (lock-then-unlock) interaction among both channel parties and F_{pay}. For the sake of clarify, we divide them into two interfaces, **Channel Update** and **Pay-**

ment Routing. Moreover, *Payment Routing* can handle two scenarios, where the locked payment is successful or canceled. 3) **Channel Closure** describes the channel closure procedure, which can be called unilaterally or bilaterally as per di erent conditions de ned in this interface.

We then discuss how these security properties are captured by functionality F_{pay}.

Guaranteed channel closure. According to functionality F_{pay}, as the interface *Closing a Channel* is called collaboratively or unilaterally, an established channel can be closed even if there is a malicious party in the channel.

Guaranteed payout for honest channel parties. According to functionality F_{pay}, the honest party will be paid no less than his latest balance in the channel.

On-chain unidenti ability. The functionality F_{pay} follows the original money mechanics of Monero, thus the environment E cannot distinguish whether the operation over R is proceed by F_{pay} in the ideal world or the protocol in the real world.

Atomicity. The intermediate participants $Ch:P^{\prime}$ can unlock a *route-payment* only when he received a message from F_{pay} denoting that he has paid in the next hop. It implies that all the on-path payments are atomic.

Unlockablity. According to the interface Channel Update, it covers the scenario that a locked payment is succeed or failed, which prevent a channel from being locked forever.

Sender/Receiver privacy. According to the interface *Channel Update*, all the intermediate nodes receive messages with the same format from F_{pay}, and neither of them can inform whether the node connected with him is the sender/receiver.

Path privacy. Similar to the *Sender/Receiver privacy* property, an intermediate node will receive the messages containing only two channels he involved, but no additional information about the entire path.

F_{pay} captures the all the the security properties de ned in Section 4.2.2, if the Theorem 3 holds.

Theorem 3. *Assuming that the signature scheme on Monero and employed by Key Escrow Service are existentially unforgeable against adaptive chosen-message attacks and our CAS scheme is secure, our system running in the* $(F_{kes}; F_M)$*-hybrid world emulates an ideal functionality* F_{pay} *w.r.t Monero ledger* L_M*.*

Proof: We have already informally argued about the security of our scheme above. We then construct a simulator S in the ideal world to observe then emulate the behavior of some xed adversary A in the real world. S generates the secret-public key pairs and initial witness-statement pairs for each participants. S then sends these public keys to each corrupted participant with his secret key, and the initial statements to two corrupted participants who share a channel together with their initial witnesses respectively. We assume that honest party will response a message once he received, while the corrupted party can decide when the message are delivered within a time . S starts from the channel establishment as an honest party, and watches the instruction of A to the corrupt parties. To make it impossible to distinguish between the simulated and the real execution, S forwards the message from F_{kes}, F_M, F_{pay} and other honest parties to the corrupted parties.

a) **Channel Establishment**. The simulation of this part: S simulates the ledger functionality and KES functionality, plays the role of a honest party Ch:P sending message (*mc-open*, Ke, Ch) to the functionality F_{pay}, which removes the UTXO $r_{Ch:P}$ from R. When received another open channel message from Ch:P', F_{pay} adds Ke and Ch to K and C respectively (or adds $r_{Ch:P}$ to R if not received the message from Ch:P').

b) **Channel Update**. This part starts when S receives a message (*mc-update*, $Ch:id; xmr$) from E, and forwards the message to party Ch:P. We discuss two

cases of P below:

1. *Ch:*P is honest. *Ch:*P sends a message meaning that he wants to updates the channel. Within time , S forwards the message to F_{pay}, who sends a con rmation message to *Ch:*P indicating *Ch:*P^{0}'s agreement on the update request.

2. *Ch:*P is honest, but *Ch:*P^{0} is corrupted. *Ch:*P sends a message meaning that he wants to updates the channel. Within time , S forwards the message to F_{pay}, and *Ch:*P^{0} does not send the message to con rm this update. Whether this update is bene cial to *Ch:*P or *Ch:*P^{0}, if $Ch{:}Bal(r_{Ch:P^{0}})$ xmr > 0, S sends the con rmation message in the name of *Ch:*P^{0} as he knows the private keys and witnesses of all the parties and the channel updated. If he uses a previous state to close *Ch*, S will correct their balances and *Ch:*P loses coins. Otherwise, if $Ch{:}Bal(r^{0}_{Ch:P})$ xmr < 0 and *Ch:*P^{0} does not con rm this message, then S does not either.

c) **Payment Routing**. This part starts when S receives a message (*mc-routepay*, $Ch{:}id; xmr\ ;\ _{id}$) from E, and forwards the message to *P*, we consider the following cases:

1. *P* is the sender. *P* sends some messages to setup a multi-hop payment path. S forwards these messages to the receiver and all the intermediate participants respectively. Notice that P^{0} is on the next hop of the payment path and shares a channel *Ch* with P. P^{0} replies con rm payment message within the time . F_{pay} receives all lock payment messages within the next time , all the on-path payments are succeed.

2. *Ch:*P is an honest participant, but *Ch:*P^{0} is a corrupted receiver. *Ch:*P and *Ch:*P^{0} sends a message to con rm a payment lock within the channel between *Ch:*P and *Ch:*P^{0} However, *Ch:*P^{0} does not unlock this payment within $_{id_{P;P^{0}}}$ + . This will be corrected by S, who sends a unlock payment message to *Ch:*P in the name of *Ch:*P^{0}.

d) **Channel Closure.** This part starts when S receives a message (*mc-close*, *Ch:id*, *Ch:State*, $Commit_P$, $_{id}$) from E, we consider the following cases:

1. P proposes a channel closure request. P sends a message to F_{kes} for closing the channel, which will be forwarded to P^0 within the time . P^0 then con rm or update the request. Once the close channel request is updated by *Ch:*P^0, and S forwards P^0 response to F_{kes}.

2. P proposes a channel closure request, but P^0 is corrupted. P sends a message to F_{kes} for closing the channel, which will be forwarded to P^0 within the time . P^0 ignores this close channel request, S will respond in the name of P^0.

4.4 Performance

We develop MoNet over Monero as a proof of concept implementation to evaluate its e ciency.

4.4.1 Evaluation

Implementation. We implement 2P-CLRAS scheme over Monero in golang using the libraries, moneroutil [63], emmy [45], and kyber [64]. The implementation takes about 800 lines of code (LoC) over these libraries. All experiments are run on a macOS with processor 2.6 GHz 6-Core Intel Core i7 and memory 16GB 2400 MHz DDR4.

As per MoNet, the message complexity within a channel in each phase includes the number of *on-chain transactions*, *signatures* and *o -chain messages*.

The number of on-chain transactions. Establishing a channel requires 1 transaction on Monero and Ethereum respectively, and no on-chain transaction is required on both Monero and Ethereum for processing an o -chain payment (updating the channel). Routing a payment in the best case does not require any

on-chain transaction on neither Monero nor Ethereum, but requires 1 on-chain transaction on Monero and 2 on-chain transactions on Ethereum for the worst case. It requires 1 transaction on Monero and Ethereum respectively if both channel parties close the channel collaboratively or an additional transaction on Ethereum under a dispute scenario.

The number of signatures and o -chain messages. As per the 2P-CLRAS scheme in Algorithm 2, some algorithms requires both signers' interaction, and a single interaction has two messages exchange between them. For example, 2P-CLRAS.PSign has two interactions with 4 messages between both signers, and each of 2P-CLRAS.JGen, 2P-CLRAS.SWGen and 2P-CLRAS.NewSw requires 1 interaction with 2 messages. Signatures are required for each o -chain message, adaptor signatures, and on-chain transactions. According to Figure 4.3 and 4.4: there are 10 o -chain messages, 1 on-chain transactions with two signatures from both channel parties, and 1 adaptor signature at the channel establishment phase, thus it requires 13 signatures; there are 4 o -chain messages and an adaptor signature at the channel update phase, thus it requires 5 signatures; there are 7 o -chain messages and an adaptor signature when routing a payment, thus it requires 8 signatures (We count the number of messages delivered within a channel but do not include the messages received from the sender in the count); there are 2 o -chain messages with 2 signatures to close a channel (the signature required by the on-chain transaction is created in each channel update phase as an adaptor signature, thus we do not count it in).

Computation time. Our evaluation of 2P-CLRAS scheme shows that the computation time of $SWGen(\,)$ is about $3{:}5ms$, $NewSW(\,)$ is about $30ms$, $PSign(\,)$ is about $3{:}5ms$, $Adapt(\,)$ is about $198ns$, $PVrfy(\,)$ is about $3{:}4ms$, and $CVrfy(\,)$ is about $330ms$.

The computation time of processing an o -chain payment equals to the time

of executing the algorithms, including $NewSW()$, $PSign()$, $PVrfy()$, and $CVrfy()$, which is about 367ms. As the general network latency is about 60ms for 4G WAN and internet connections[1], it requires about 427ms to process a transaction within a channel. MoChannel improves the throughput on Monero from 1000[2] tps to $2{:}34D$, where D is the number of channels opened on Monero. For example, as of Jan 2022, there are more than $80{;}000$ channels[3] in the Lightning Network for Bitcoin. If the MoChannel is of the same scale, it has the potential to provide a throughput of over $180{;}000$ tps.

Communication Overhead. We measure the communication overhead as the size of messages that two parties need to exchange for each o -chain payment. The to be exchanged messages size, including an adaptor signature, a newly generated statement and the proof on two consecutive statements, is about 18 KB.

Optimization. We can further optimize MoChannel by pre-computing a batch of statement-witness pairs. The optimized channel update requires only the creation and veri cation of an adaptor signature. We present the comparison in processing an o -chain payment between the original and optimized MoChannel in Table 4.2. The computation time of creating and verifying an o -chain payment is about $6{:}9$ms, thus it requires $66{:}9$ms to process an o -chain payment within a channel under the 4G WAN network latency, which is $6{:}4$ faster than the non-optimized version (441ms). If at the same scale of lightning network, the optimized MoChannel can handle about $14{:}9$ transactions per channel per second, and throughput is more than $1{;}100{;}000$ tps, which could reach the same level of lightning network ($1{;}000{;}000$ tps [4]).

[1]Data source: https://www.sas.co.uk/learning/complete-guide-to-4g-wan

[2]Data source: https://alephzero.org/blog/what-is-the-fastest-blockchain-and-why-analysis-of-43-blockchains/ on 23 Aug, 2021

[3]Data source: https://txstats.com/dashboard/db/lightning-network?orgId=1.

[4]Data source: https://medium.com/coinmonks/how-does-bitcoin-get-scalable-wit

Table 4.2: Performance Comparison.

	Original MoChannel	Optimization
Creation	33:5ms	3:5ms
Veri cation	333:4ms	3:4ms
Throughput	180; 000 tps	1; 100; 000 tps

In the optimized MoChannel, it requires only about 0:03 KB exchanged data in processing an o -chain transaction. We evaluate the optimized MoChannel with 100 transactions by using the pre-computation. The result shows that if two channel parties pre-compute 100 payment sessions, it requires about 0:08ms for each party to create 100 witness-statement pairs and the proof on two consecutive witnesses, and about 3:46s to verify these witness-statement pairs consecutiveness. The size of all proof is about 1:76 MB.

We also evaluate the performance of **routing a payment via multi-hop MoChannel** described in Section 4.2.3. Our implementation simulates the optimistic scenario, where the receiver unlocks the a payment by himself. Within a channel, locking a payment means creating a locked 2P-CLRAS scheme. The performance of a multi-hop payment in a single channel (with pre-computation) is concluded in Table 4.3. Routing a multi-hop payment by using MoNet can be

Table 4.3: Performance of a Multi-hop MoChannel Payment.

	the Optimized MoChannel
Setup	0:25ms
Lock	4:78ms
Unlock	3:65ms

processed within 68:68ms n_h via n_h hops with a 4G WAN latency 60ms.

For the sake of exploring the feasibility of our system, we also provide a proof of concept implementation for Key Escrow Service contract in Ethereum using Tru e [65], a development framework of Ethereum smart contract. The evalu-

h-the-lightning-network-63591040462c

ation shows that, it costs 127869 gas to deploy a Key Escrow Service contract, about 49801 gas to retrieve both parties' funds without dispute, and about 123412 gas to process the channel dispute (including the scenario that closing a channel with dispute and routing a payment with dispute) on Ethereum.

4.5 Conclusion

This chapter introduces MoNet, a bi-directional payment channel network for Monero that operates with an unlimited lifetime. MoNet achieves full compatibility with Monero, requiring no modi cations to the existing Monero blockchain. The network ensures transaction fungibility, meaning transactions conducted over MoNet and Monero remain indistinguishable. Additionally, it guarantees user anonymity for both Monero and MoNet users, e ectively avoiding any potential privacy leakage that could arise from the implementation of a new payment channel network. Furthermore, the proposed VCOF and CAS primitives further contribute to the development of versatile and e cient building blocks for future blockchain applications beyond MoNet.

Chapter 5

FPSS

This chapter introduces the Flexible Proactive Secret Sharing (FPSS) scheme as an alternative building block to enable the third party service requried in the proposed protocols (AuxChannel, MoChannel, and MoNet) in the previous two chapters. The FPSS scheme eliminates the need for a distributed third party to resolve potential channel disputes. The FPSS scheme includes an additional phase called F-HandO , which enables e ective membership updates, allowing for seamless transitions when members join or leave the group. F-HandO only requires the interaction of the pairs of shareholders who are leaving and joining, making it a highly e cient and practical solution for managing changes in group membership.

Here is the core innovation of F-HandO . First, F-HandO adds *exibility* for shareholders to hand their shares o to another new joint one. Second, when multiple shareholders, update their shares concurrently, only the pair of the leaving and joining shareholders needs to be interactively involved. The hand-o of FPSS has two cases, *single F-HandO* and *batched F-HandO* .

- *Single F-HandO* . It allows a shareholder to hand-o his share to a new shareholder. The single F-HandO involves only two shareholders, the one

who is leaving and the other who is joining. The single F-HandO can be executed by any shareholder at any time without interactively communicating with the remaining committee.

- *Batched F-HandO* . It allows multiple shareholders to leave at the same time without interacting with each other. Leaving shareholders can be any subset of the committee. Communication is only required between the pair of leaving and joining shareholders. Thus, the single F-HandO cannot be directly used in the batch F-HandO , since each involved shareholder should have a consensus on the new committee members.

The detailed protocol of FPSS will be presented in Section 5.1. We also provide the comparison among our proposed FPSS and other existing secret share schemes in Table 5.1.

Roughly speaking, FPSS uses a blockchain to validate each F-HandO and store the information publicly. The shareholders involved in the hando process also transmit messages o -chain to each shareholder in the old and new committees. By leveraging the blockchain, any malicious behavior can be detected, enabling the system to incentivize honesty and penalize malfeasance. While discussing the details of this incentive mechanism is beyond the scope of this paper. It's important to note that the two types of F-HandO do not serve as countermeasures to each other in the event of a fault, but rather provide distinct functionalities for di erent use cases. Therefore, it's essential to carefully consider the appropriate option for the speci c situation at hand.

5.1 FPSS

FPSS, a proactive secret share protocol, is more exible by adding an F-HandO algorithm. F-HandO can be executed by a single shareholder or any subset of

Table 5.1: Comparison of PSS Schemes.

PSS scheme	Network	Adv	BC?	On-Dmd*	Hand-O			Resilience
					Step	O -Chain	On-Chain	
MPSS [66]	p-Sync	Mobile	No	No	8	$O(n^4)$	| †	1=3
COBRA [67]	p-Sync	Adaptive	No	No	4	$O(n^3)$	|	1=3
Shanrang [68]	Async	Mobile	No	No	3	$O(n^3 \log n)$	|	1=4
Yurek et al. [69]	Async	Mobile	No	No	2	$O(n^3)$	|	1=3
Opt-CHURP [21]	Sync	Mobile	Yes	No	3	$O(n^2)$	$O(n)$	1=2
Exp-CHURP-A [21]	Sync	Mobile	Yes	No	3	|	$O(n^2)$	1=2
Exp-CHURP-B [21]	Async	Mobile	Yes	No	3	|	$O(n^3)$	1=3
Benhamouda et al. [19]	Sync	Mobile	Yes	No	2	$O(nt)$ ‡	$O(n)$	1=4
Goyal et al. [20]	Sync	Adaptive	Yes	No	4 §	$O(n^2)$	$O(n)$	1=2
FPSS (This Work)	Sync	Adaptive	Yes	Yes	2	$O(nm)$ ¶	$O(m)$	1=2

*Normally, PSS divides a secret's lifetime into epochs, and hand-o will be executed at the beginning of every epoch. \On-Dmd" indicates whether the PSS scheme supports the hand-o to be executed in the middle of an epoch or not.

†The horizontal line denotes that the work did not discuss such an approach.

‡t is the threshold, where anyone with $t + 1$ valid shares from the same committee can reconstruct the corresponding secret.

§2 rounds for preparation phase and another 2 round for the hand-o phase

¶m is the number of shareholders leaving the committee concurrently.

Table 5.2: Notations

Notations	Meaning
C, $C^{\prime}$	the old (current) and new committees respectively, where $jCj = jC^{\prime}j = n$.
PKs_C	the public key set consisting all the public keys belonging to each of members in C respectively.
$(pk_{C_i}; sk_{C_i})$	the key pair of the shareholder C_i.
P / $P^{\prime}$	the leaving / joining shareholders.
$f(\ ),\ (\ ),\ ^{\prime}(\ )$	t degree polynomials, where $f(0) = s,\ (0) = \ ^{\prime}(0) = 0$.
$f\ _i g_C$	the set of shares owned by each of C respectively. Such that a number of t subset of $f\ _i g_C$ can be used to compute the secret s.
RPs_P	the set of shares of $(\ )$ or $^{\prime}(\ )$ generated by P.
$fCTR_i g_P$	the set of encrypted RPs_P.
$CTS_{pk_P;}$	the ciphertext on the share, which can be used to reconstruct a secret, under pk_P.

shareholders in the same committee concurrently at any time, where only the pair of leaving and joining shareholders need to be interactively involved. This section presents the system model, high-level description of F-HandO .

5.1.1 System Model and Security Properties

Notations. Notations that will be used to explain the protocols of FPSS are summarized in Table 5.2.

System Model. Our objective is to design a secure and exible hand-o mechanism that allows any subset of honest shareholders to transfer their shares to new members without involving the entire committee. To achieve this goal, we propose the Flexible Proactive Secret Sharing (FPSS) system, which includes a hand-o mechanism, F-HandO , that requires only the leaving and joining shareholders to interact.

The F-HandO process is straightforward, with a leaving shareholder P initiating the transfer by uploading a message to the blockchain, and the designated party $P^{\prime}$ responding with a message to accept the hand-o . To ensure the security

and integrity of the hand-o process, we employ the KZG scheme (see Section 2.6) and Veri able Encryption scheme (see Section 2.7).

Our proposed mechanism o ers exibility in situations where shareholders frequently leave or join the committee, allowing the committee to operate e ciently without requiring the participation of the entire committee for each transfer. By utilizing the KZG and Veri able Encryption schemes, our mechanism guarantees secure and veri able hand-o s, providing assurance that shares are correctly transferred to the intended recipient.

Blockchain and Network Model. FPSS employs a blockchain, which satis es the properties of *persistence* and *liveness*, as one of the communication channels. All communications during a exible hand-o are over both blockchain and the p2p network. The entire exible hand-o execution does not require the acknowledgment from all recipients.

Adversary Model. We consider FPSS under the adaptive adversary model. Assuming that a computationally bounded adversary A can adaptively choose at most t shareholders for every $_c$ time interval. That is, anyone can access at most t honest shares in the same committee. Once a shareholder has been corrupted, A can arbitrarily control his behavior as well as modify the memory. A party is honest before he joined the committee, and once he is assigned a share, A can decide to corrupt this party or not. For example, if a shareholder P_i was corrupted by A in the old committee C, he is still the corrupted shareholder in C^0, if he is not left. While if he is corrupted for the rst time in C^0, he was honest in C.

FPSS inherits the two security properties *robustness* and *secrecy* from the PSS scheme, and introduces an additional property *exibility*. The formal de nitions are given below:

Definition 11 (Robustness). *For any PPT adversary* A *with the corruption*

threshold t, once a secret s is deposited among n > 2t parties by using FPSS, anyone who has t + 1 honest shares in the same committee can reconstruct a secret s', where $s' = s$.

Definition 12 (Secrecy). *For any PPT adversary* A *with the corruption at most t out of n shares, the probability of* A *to calculate a secret s', where $s' = s$ and n > 2t, is negligible.*

Definition 13 (Flexibility). *An FPSS is flexible if a share can be successfully handed off among a pair of old and new shareholders, as long as the pair of shareholders correctly execute the hand-off.*

5.1.2 High-level Description of FPSS

We propose FPSS, which enables each execution of a hand-off interactively involves only the pair of leaving and joining shareholders. The construction of FPSS is based on the Shamir secret share protocol [70] with an additional flexible hand-off algorithm, called F-HandOff. FPSS can be deployed in two cases, single hand-off and batched hand-off:

- Single F-HandOff. The single F-HandOff requires only two parties to be interactive: the one who is leaving the committee and the other who is joining the committee. Roughly, each of the two parties contributes n random shares located on a private polynomial that passes through the point $(0; 0)$ for each of the committee members, respectively. As demonstrated in Fig 5.1, enabling a single hand-off requires the old shareholder P_4 and the new shareholder $P_{4'}$ to contribute sets of shares, respectively, and assign their shares to each member of the committee. A successful F-HandOff securely hands off a share from the leaving party to the joining party and updates all shares in the committee accordingly.

- Batched F-HandO . FPSS also allows any subset of the committee to execute the F-HandO concurrently. While each of the F-HandO is independent of other concurrent F-HandO s. To execute a F-HandO , the leaving party does not need to reach a consensus with other potential leaving party. Additionally, if one F-HandO fails, it does not a ect other concurrent F-HandO .

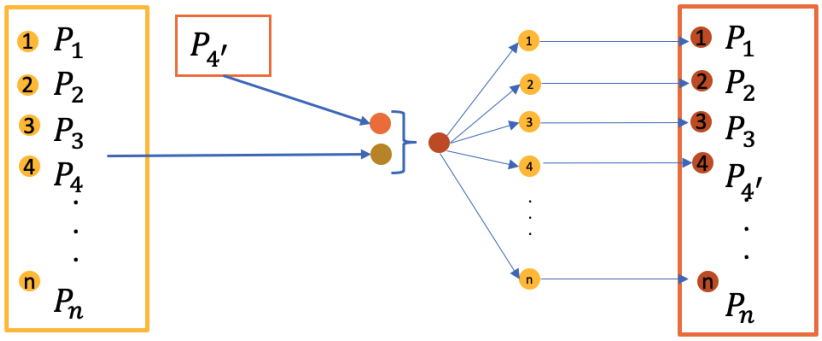

Figure 5.1: Overview of F-HandO

5.2 Formalization of FPSS

This section describes FPSS in detail. FPSS allows shareholders to leave individually (Section 5.2.3) and also supports the batched leaving scenario (Section 5.2.4).

5.2.1 Setup

Before distributing a secret, a trusted third party creates the public key KZG:PK for the KZG commitment scheme and system parameter $n = pq$, where $p; q$ are safe primes, for the veri able encryption scheme at the start of the protocol (such a trusted party can be replaced by an MPC protocol [71; 72] executed among miners and the initial committee).

5.2.2 Distribution

To distribute a secret s, the secret owner first selects an initial committee C^0 and distributes the secret to its members. Each committee member P_i is assigned a share $_i$ of the secret, which is encrypted and recorded on the blockchain for transparency and immutability. The selection process for the initial committee is assumed to be known by the secret owner and is not considered in this paper. Once the distribution phase is complete, each committee member P_i holds a unique share $_i$ of the secret s.

5.2.3 FPSS - Single F-HandOff

This section describes F-HandOff for the single leaving scenario. That is there is only one shareholder $P_j \in C$ who wants to hand-off his share to another party $P_{j'}$, and a new committee is defined as $C' = (C \setminus P_j) \cup P_{j'}$.

Once a secret s is distributed among a committee C, each shareholder of the committee C_i will be assigned a share $_i$, and each of their public keys pk is publicly accessible on-chain. Let $P = C_j \in C$ owns $_j$ be the one, who is leaving the committee, and P' be the newly joint one. Both P and P' then proceed as follows:

1. P generates a t-degree polynomial $(\ )$ and produces a set of points, $\{_i\}_{i \in [1;n]}$, where $_i = (i)$ and $(0) = 0$;
2. P randomizes his share $_j$ by $'_j = _j + _j$;
3. P encrypts the randomized share $'_{P'}$ under P''s public key as well as each of $\{_i\}_{i \in [1;n] \setminus j}$ under C_i's public key;
4. P produces $_P$ for proving the correctness of these encrypted shares $'$ and $\{_i\}$ (the definition of the correctness will be discussed later this Section);

5. P broadcasts a leaving request, which contains the leaving request indicator, the request ID, the encrypted $^{0}_{P'}$, the public key of the designated party, n 1 encrypted $f\ _i g = {}_{P'}$ as well as the encrypted randomized share ^{0}, and the corresponding proof $_P$. Notice that only these encrypted $f\ _i g = {}_{P'}$ and ^{0} will be recorded on-chain.

6. Once P$'$ receives the assigned share from P, he generates a t-degree polynomial $^{0}( )$ and produces a set of shares, $f\ ^{0}_i g_{i2[1;n]}$, where $i \neq j$, $^{0}_i = {}^{0}(i)$ and $^{0}(0) = 0$;

7. P$'$ encrypts each of $f\ ^{0}_i g$ under C$_i$'s public key;

8. P$'$ produces $_{P'}$ for proving that these encrypted shares $f\ ^{0}_i g$ are correctly generated (the de nition of the correct encrypted $f\ ^{0}_i g$ will also be discussed later this Section);

9. P$'$ broadcasts a joining request, which contains a signature proving that he is the designated party and the n 1 encrypted $f\ ^{0}_i g$ and the corresponding proof for proving the correctness of these encrypted points. Similarly, on-chain record only contains the n 1 encrypted $f\ ^{0}_i g$ if the joining request is veri ed by miners.

The leaving and joining requests always appear on-chain in pairs. That is, once a miner receives a valid leaving request, he will keep this request until he receives the corresponding valid joining request. As a result, the F-HandO between P and P$'$ completes successfully. We use both on-chain and o -chain communications, shareholders in the same committee can check the received share using the on-chain records.

Both P and P$'$ do not have to receive acknowledgment messages con rming whether all shareholders accept this request or not until there is a valid reconstruction request or the next periodic hand-o . Thus, F-HandO can tolerate an

extremely long delay for at least $t + 1$ honest shareholders to receive their shares, respectively, until there is a valid reconstruction request or the next periodic hand-o .

A successful F-HandO outputs a new committee $C^0 = P^0 [fC=Pg$ and updates shares beloning to shareholders in C^0, respectively, where any subset of $t + 1$ valid shares can reconstruct the original secret s. After a F-HandO , all honest parties erase their revoked shares.

We then discuss two scenarios when P or P^0 does not follow the protocol correctly.

Discussion 5.2.1. *What if* P *or* P^0 *does not send the leaving or joining request accordingly?*

We emphasize that the F-HandO would be executed on-demand. If the leaving shareholder P does not send a leaving request, it means that he does not wants to leaves the committee. If the newly joint shareholder P^0 did not respond, P can re-designate a party P^{00} by sending another leaving request to replace the previous one. Since the leaving and joining requests are written to blockchain in pairs, the old leaving request would never be written to blockchain. Thus, the new leaving request will not incur any additional on-chain cost.

Discussion 5.2.2. *What if some shareholders did not receive their shares after an F-HandO ?*

This is the \share availability" issue, which can be solved by the on-chain records. That is, any of shareholders can make request to blockchain to retrieve their lost shares. Since the blockchain achieves the *persistence* and *liveness*, at least $t + 1$ honest shareholders can retrieve their shares eventually.

Discussion 5.2.3. *The compromise of a shareholder's encryption key can result in the vulnerability of all shares encrypted using that key, which poses a signi cant*

security risk, especially since all shares are recorded on the blockchain. Given the assumption that the adversary can adapt their corruption strategy to target di erent shareholders, it is crucial to prevent this vulnerability from occurring.

It is crucial to prevent this vulnerability from occurring, given the assumption that the adversary can adapt their corruption strategy to target di erent shareholders.

To address the security risk posed by a compromised shareholder's encryption key, we introduce the forward-secrecy property for the shareholders' public key scheme. This property ensures that even if an adversary compromises a shareholder's private key, they cannot decrypt any ciphertext encrypted using that key before the shareholder updates their keys. To implement this property, we require that honest shareholders update their secret and public key pairs every $_c$ time interval. By doing so, any ciphertext encrypted using a previous key pair remains secure even if the corresponding private key is compromised. This added security measure mitigates the risk of a single compromised shareholder compromising the security of the entire protocol.

Formal single F-HandOff scheme (Figure 5.3). We then describe the formal single F-HandO scheme. Recall the notations in Table 5.2. Let RPs_P be the set of randomized points and $\{\mathsf{CTR}_i\}_{\mathsf{RPs}_P}$ be the set of encrypted RPs_P.

To leave the committee C, P should upload a leaving request $\mathsf{LReq}_P := (\mathsf{LR}_P; \mathsf{ID}_{\mathsf{LR}_P}; pk_{P'}; ds; \ _P)$ to blockchain, where LR_P is the leaving request indicator, $\mathsf{ID}_{\mathsf{LR}_P}$ denotes the request ID, $pk_{P'}$ is the public key of a designated party, ds is a set of encrypted data (de ned in Fig 5.3, line 13) and $_P$ denotes the proof of the correctness of this leaving request. $_P$ is outputted by a NIZK proof algorithm, F-HandO .Prf, which takes a pair of NIZK instance as inputs.

The leaving request (LReq_P) will be veri ed following the request validation check $\mathsf{RVC}(\mathsf{LReq}_P)$ (see Fig. 5.2), before it is accepted by the underlying

RVC (Req)

```
1 : Input: Req;
2 : Output: 1=0;
3 : Initiate: b = 1;
4 : Parse Req ! fRSIGN; ID; pk_P' or ; ds; g:
5 :   If RSIGN = \LReq":
6 :     If satis es R_LReq: b = 0;
7 :   Elif RSIGN = \JReq":
8 :     pk L(ID);
9 :     If pk = ? or Sign.Vrfy_pk(pk; ; ID) = ?: b = 0;
10 :    If not satis es R_JReq: b = 0;
11 :  Else b = 0;
12 : Return b;
```

Figure 5.2: Request Validation Check

blockchain. Let $i; j \; 2 \; \mathbb{N}^+$, $i; j \; 2 \; [1; n]$ and $i \neq j$, the leaving request should satisfy the following statements:

1. for each given encrypted share $(i; \mathrm{Enc}_{pk_i}(y))$, it should satisfy that $y = \ (i)$;
2. for the given point $(j; \mathrm{Enc}_{pk_j}(y))$, it should satisfy that $y = \ _j + \ (j)$ and;
3. $(0) = 0$;

y is private to veri ers. The formal relationship for the preceding proving statement is shown below.

Assuming that $\ _{R_P^{LReq}}$ is a proof generated by P for proving the relation R_P^{LReq}. For the NIZK instance $(\ _P; !_P)$, where $\ _P := (\mathrm{CTS}_{\mathrm{pk}_{P'}; \ _j}; \{\mathrm{CTR}_i\}; \ _{P; R_P^{LReq}})$ and

Single Hand-off

Input: ($_j$, t, C, fPK$_i g_{i2[1;n]}$, P$'$, $pk_{P'}$)
Output : C$'$
Initiate: P = C$_j$, C$'$ = C=P, RPs$_P$ = ;, fCTR$_i g_{RPs_P}$ = ;, fCTR$_i g_{RPs_{P'}}$ = ;, and isValids = 0
  generates a t-degree polynomial ( ), s.t. (0) = 0;
  For i 2 [1; n], calculates:
    RPs$_P$[i] = (i);
    if i 6= j: fCTR$_i g_P[i]$ = Enc$_{PKs_C[i]}$(RPs$_P$[i]);
  calculate $'_j$ = $_j$ + RPs$_P$[j];
  execute CTS$_{pk_{P'};\ '_j}$ = Enc$_{pk_{P'}}$($'_j$);
  execute $_P$:= F-HandO .Prf($_P$; $!_P$) I $_P$ and $!_P$ are de ned in Equation 5.1
  ds := (fCTR$_i g_P$; CTS$_{pk_{P'};\ '_j}$)
  send LReq$_P$:= (LR_P; ID_{LR_P}; $pk_{P'}$; ds; $_P$) to blockchain, and send each of fCTRg and CTS$_{pk_{P'};\ '_j}$ to shareholders in the new committee, respectively;
The joined party P$'$:
  If received CTS$_{pk_{P'};\ '_j}$ from P:
    Decrypt CTS$_{pk_{P'};\ '_j}$ and gain $'_j$;
    generate a t-degree polynomial $'$( ), s.t. $'$(0) = 0;
    calculate $''_j$ = $'_j$ + $'$(j);
    For i 2 [1; n]; i 6= j, execute:
      RPs$_{P'}$[i] = $'$(i);
      fCTR$_i g_{P'}[i]$ = Enc$_{PKs_{C'}[i]}$(RPs$_{P'}$[i]);
    sign $_{pk_{P'}}$ = Sign($sk_{P'}$; ID$_{LR_P}$);
    execute $_{P'}$:= F-HandO .Prf($_{P'}$; $!_{P'}$) I $_{P'}$ and $!_{P'}$ are de ned in Equation 5.2.
    ds := fCTR$_i g_{RPs_{P'}}$
    send JReq$_{P'}$:= (JR$_{P'}$; ID$_{LR_P}$; $_{pk_{P'}}$; ds; $_{P'}$) to blockchain, and send each of fCTR$g_{P'}$ to shareholders in the new committee, respectively, except for P$'$ himself;
Blockchain Validation Check:
  isValids = (RVC(LReq$_P$)&&RVC(JReq$_{P'}$));
If isValids:
  set C$'$ = C$'$ [P$'$, and the F-HandO succeed;
Else: break;

Figure 5.3: FPSS - Single Hand-o

$!_P := (\ _j;\ ^0_j; \mathsf{RPs}_P; \mathsf{f}(\ ))$, and the relation R_P^{LReq} de ned as follow:

$$
\begin{aligned}
R_P^{\mathsf{LReq}} = f(\ _j;\ ^0_j; \mathsf{RPs}; \mathsf{f}(\ )) :\ & \\
& \mathsf{KZG.cmt}_{\ ^0_j} = \mathsf{KZG.cmt}_{\ _j} + \mathsf{KZG.cmt}_{f\mathsf{CTR}_i g[j]} \wedge \\
& \mathsf{CTS}_{\mathsf{pk}_{P^0};\ ^0_j}\ ,\ \mathsf{KZG.cmt}_{\ ^0_j} \wedge \\
& \text{for } i \in [1; n] : f f \mathsf{CTR}_i g[i]\ ,\ \mathsf{KZG.cmt}_{\mathsf{RPs}[i]} g \wedge \\
& \mathsf{KZG.cmt}_{\ (0)}\ ,\ 0 g;
\end{aligned}
\tag{5.1}
$$

where , links two di erent operations on the same value. Recall that we assume that every successful update of a share will result in recording the commitments of all updated shares on-chain. The correctness of $_j$ can be easily veri ed by comparing its on-chain commitment. Therefore, we did not include the validation check in our NIZK statement.

To take over P's seat in C, P^0 should upload a joining request $\mathsf{JReq}_{P^0} := (\mathsf{JR}_{P^0}; \mathsf{ID}_{\mathsf{LR}_P};\ _{pk_{P^0}}; ds;\ _{P^0})$ to blockchain, where JR_{P^0} is the joining request indicator, $\mathsf{ID}_{\mathsf{LR}_P}$ is the index of the corresponding leaving request, $_{pk_{P^0}}$ is the signature on $\mathsf{ID}_{\mathsf{LR}_P}$ under pk_{P^0} for the identity authentication, ds is the set of encrypted shares, (de ned in Fig 5.3, line 25) and $_{P^0}$ is the proof for the validation of the joining request.

The joining request, JReq_{P^0}, should also satisfy the following statements:

1. for each given point $(i; \mathsf{Enc}_{pk_i}(y))$, it has $y = \ ^0(i)$;
2. $\ ^0(0) = 0$;

where y is private to veri ers. The formal relationship for the preceding proving statement is presented below.

Let $_{P^0;R_{P^0}^{\mathsf{JReq}}}$ be the proof generated by P^0 under the relation $R_{P^0}^{\mathsf{JReq}}$. For the NIZK instance $(\ _{P^0}; !_{P^0})$, where $_{P^0} := (f\mathsf{CTR}_i g_{\mathsf{RPs}_{P^0}};\ _{P^0;R_{P^0}^{\mathsf{JReq}}})$ and $!_{P^0} :=$

$(RPS_{P'};\ '(\ ))$, and the relation $R^{JReq}_{P'}$, where $R^{JReq}_{P'}$ is de ned as:

$$R^{JReq}_{P'} = f(RPS_{P'};\ '(\ )) : KZG.cmt\ _{'(0)}\ ,\ 0\hat{}$$
$$\text{for } i \ 2\ [i;n]=j : f\,fCTR_i g_{RPS_{P'}}[i]\ ,\ KZG.cmt\ _{'(i)}g \tag{5.2}$$

5.2.4 Batched F-HandOff

Section 5.2.3 already introduced the single F-HandO , where a single shareholder can hand-o his share to another shareholder without the rest of the shareholders' interaction. In practice, several shareholders from the same committee C might leave concurrently. This section introduces how FPSS deals with such a concurrent scenario, called batched F-HandO .

Intuitively, the batched F-HandO is consisted of multiple single F-HandO s. But the real procedure is more complicate than the intuition. Before introducing the batched F-HandO , we explain why the single F-HandO cannot be directly executed by di erent parties in the same committee. For the ease of understanding, our explanation uses a simple example. Let C be a committee of 3 members, where $C = fP_a;P_b;P_cg$. Assuming that a dealer deposits a secret s among C, such that each of P_a, P_b and P_c holds shares $_a$, $_b$ and $_c$ respectively. Any two of them can reconstruct a value s', where $s' = s$. P_a and P_b propose their leaving requests individually at the time t. P_a decides to hand-o his share $_a$ to P'_a, and P_b chooses P'_b to take his share $_b$. Thus, from P_a's view, the new committee is $C'_a = fP'_a;P_b;P_cg$, while for P_b, the new committee C'_b is consist of $fP_a;P'_b;P_cg$. Since the two requests are proposing concurrently without negotiation, the newest committee C' should be the intersection of both C'_a and C'_b, such that $C' = fP'_a;P'_b;P_cg$. The share of each party in C' are:

- $_{a'} = _a + _{a\ >a'} + {}^{'}_{a'\ >a'} + {}^{'}_{b'\ >a'}$;
- $_{b'} = _b + _{b\ >b'} + {}^{'}_{b'\ >b'} + {}^{'}_{a'\ >b'}$;

- $_c = _c + _{a>c} + _{b>c} + _{^0a^0>c} + _{^0b^0>c}$;

where the share $_{a>a^0}$ is generated by P_a and encrypted under $pk_{P^0_a}$, and so on. Namely, P^0_a losses the share $_{b>a}$, while P^0_b losses the point $_{a>b}$, since they are not able to access the decryption keys sk_{P_a} and sk_{P_b}, respectively. Due to the inappropriate hand-o , the new shares $_{P^0_a}$ or $_{P^0_b}$ cannot contribute to reconstruct a valid secret. In the demonstrated case above, no one can obtain a valid secret from C^0 in the Reconstruction phase, which breaks the *robustness* property.

We introduce two potential solutions to the above issue, but they still have drawbacks.

First Attempt. The incorrect assign issue can be revised by adding another response step. In this step, the leaving parties P_a and P_b should disclose their received shares to P^0_a and P^0_b, respectively. A successful F-HandO requires that each of P_a and P_b encrypts the two decrypted shares $_{b>a}$ and $_{a>b}$ under the public keys $pk_{P^0_a}$ and $pk_{P^0_b}$, respectively. However, it is vulnerable to single point of failure. If any of P_a and P_b fails to disclose their received points to P^0_a and P^0_b respectively, the entire hand-o failed as well, albeit the failure is auditable.

Second Attempt. Since $_b$ and $_a$ are encrypted under P_a and P_b's public keys respectively, we can require that each leaving party P discloses his secret key to the designated party P^0 within the leaving request. However, this attempt leads to a share leakage issue. That is, once P^0 knows sk_P, he can also decrypt all shares owned by P's including the share contributing to Reconstruction phase, which breaks the *secrecy* requirement.

Our approach. Comparing the two attempts above, it is hard to solve the single point of failure vulnerability in the rst attempt. Since it is too strong to assume that there is no single point of failure during the protocol execution. We resolve the share leakage issue in the second attempt by using two key-pairs to

guarantee the *secrecy* property and introduce our approach as follows:
Each shareholder P in the committee C has two key-pairs. One is called the long-term key-pair $(SK^{l}_{P}; PK^{l}_{P})$ for receiving shares assigned to him. The other is called ephemeral key-pair $\{SK^{e}_{P}; SK^{e}_{P}\}$ for receiving the random share from leaving parties and the corresponding joining parties. Thus, when P is leaving the committee, he should encrypt his SK^{e}_{P} by using $PK^{l}_{P'}$, and include the encrypted sk^{r}_{P} in the leaving request. Thus, P' can decrypt all the random shares sending to P but not the original share owned by P.

Each of the batch F-HandO s is similar to the single F-HandO with the two key-pair design. To avoid repetition, we omit the bathed F-HandO description here, and provide the formal bathed F-HandO in Figure 5.4.

Discussion 5.2.4 (Single Failure in Batched F-HandO). *Supposing that there are two concurrent leaving requests,* $BLReq_{P_a}$ *and* $BLReq_{P_b}$*, and only one joining party responses a joining request, for example* $BJReq_{P'_a}$.

To prevent being a ected by such a single failure, all shares contained in a request are encrypted under the corresponding leaving party's ephemeral public key.

If both $P'_{a'}$ and $P'_{b'}$ succeed to upload a joining request $BJReq_{a'}$ and $BJReq_{b'}$ on-chain, both F-HandO executions are succeed. Since P_a and P_b also release their ephemeral private key to $P'_{a'}$ and $P'_{b'}$ in the leaving request $BLReq_a$ and $BLReq_b$ respectively, each of $P'_{a'}$ and $P'_{b'}$ can decrypt these shares accordingly. However, if $P'_{b'}$ fails to response his joining request $BJReq_{b'}$, the F-HandO between P_b and $P'_{b'}$ failed. That is P_b is still a shareholder in the next committee $C' = \{P'_{a'}; P_b; P_c\}$, and P_b can decrypt these shares $_{a\ >b'}$ and $_{a'\ >b'}$ and update his share accordingly. So such a failed concurrent HandO between P_b and $P'_{b'}$ does not a ect the concurrent F-HandO between P_a and $P'_{a'}$.

Discussion 5.2.5 (Con dential Issue of Batched F-HandO). *A failed F-HandO leads to the disclosure of* SK^e_P. *Once* P *discloses* SK^e_P *to* P', *all messages encrypted under* SK^e_P *can be accessed by* P' *even if this F-HandO failed.*

There are two potential solutions. One is that P can update his $(SK^e_P; PK^e_P)$ on-chain if his F-HandO fails. The other is that if the failure caused by some non-malicious reasons, such as the unstable network issue of P', P can execute the hand-o phase again with the same P'.

Formal batched F-HandOff protocol (Figure 5.4). We then describe the scenario that two shareholders in the same committee are leaving concurrently. That is P_a and P_b will hand their shares o to P'_a and P'_b, respectively. While there is a case that P'_b did not response a joining request, but P'_a did. Only the hand-o between P_b and P'_b failed, P_a and P'_a can still succeed in executing the hand-o .

To leave the committee C, P uploads a batched leaving request $LReq^b_P := (LR_P; LRID_P; PK^l_{P'}; ds; {}_{P;R^{BLReq}_P})$ to blockchain. Di erent from it in the single F-HandO , $PK^l_{P'}$ indicates the long-term public key of P'. The *ds* is the set of these encrypted shares (de ned in Fig 5.4,line 15). The proof of a batched leaving request ${}_{P;R^{BLReq}_P}$ is the same as the single leaving request with an additional statement:

- for the given ephemeral public key PK^e_P and ciphertext $CTSK_{PK^l_{P'};SK^e_P}$, SK^e_P is the secret key associated with PK^e_P, where $CTSK_{PK^l_{P'};SK^e_P}$ is the ciphertext of SK^e_P.

Assuming that ${}^b_{R^{BLReq}_P}$ is a proof generated by P in the batched F-HandO case for proving the relation R^{BLReq}_P. For the NIZK instance $({}^b_P; !^b_P)$, where ${}^b_P := (CTSK_{PK^l_{P'};SK^e_P}; CTS_{PK^l_{P'};\ {}'_j}; fCTR_i g_P; PK^e_P; {}_{P;R^{BLReq}_P})$ and $!_P :=$

$(SK^{e}_{P};\ _{j};\ ^{0}_{j}; RPs_{P};\ (\ ))$, the relation R^{BLReq}_{P} is de ned as:

$$
\begin{aligned}
R^{BLReq}_{P} = f(SK^{e}_{P};\ _{j};\ ^{0}_{j}; RPs_{P};\ (\ )) :\ & \\
& CTSK_{PK^{l}_{P^{0}};SK^{e}_{P}}\ ,\ PK^{e}_{P}\ \hat{}\\
& KZG.cmt\ _{^{0}_{j}}\ ,\ KZG.cmt\ _{j} + KZG.cmt_{f CTR_{i}g[j]}\ \hat{}\\
& KZG.cmt\ _{^{0}_{j}}\ ,\ CTS_{pk_{P^{0}};\ ^{0}_{j}}\ \hat{}\ KZG.cmt\ _{(0)}\ ,\ 0\ \hat{}\\
& for\ i\ 2\ [1;n]{=}j\ :\ f f CTR_{i}g_{P}[i]\ ,\ KZG.cmt_{RPs_{P}[i]}gg:
\end{aligned}
\tag{5.3}
$$

Identical to the single F-HandO , the party that is designated in the leaving request, uploads to blockchain the joining request $BJReq_{P^{0}} := (JR_{P^{0}}; LRID_{LR_{P}};\ _{PK^{l}_{P^{0}}}; ds;\ _{P^{0};R^{BJReq}_{P^{0}}})$.

Let $_{P^{0};R^{BJReq}_{P^{0}}}$ be the proof generated by P^{0} under the relation $R^{BJReq}_{P^{0}}$. For the NIZK instance $(\ _{P^{0}}; !_{P^{0}})$, where $_{P^{0}} := (f CTR_{i}g_{P^{0}};\ _{P^{0};R^{JReq}_{P^{0}}})$ and $!_{P^{0}} := (RPs_{P^{0}};\ ^{0}(\ ))$, the relation $R^{BJReq}_{P^{0}}$ is de ned as:

$$
\begin{aligned}
R^{BJReq}_{P^{0}} = f(RPs_{P^{0}};\ ^{0}(\ )) :\ ^{0}(0) = 0\ \hat{}\\
for\ i\ 2\ [i;n]{=}j\ :\ f f CTR_{i}g\ ,\ KZG.cmt_{RPs_{P^{0}}[i]}gg
\end{aligned}
\tag{5.4}
$$

We refer the detail construction of the relation $R^{LReq^{b}}_{P}$ (Equation 5.1), $R^{JReq}_{P^{0}}$ (Equation 5.2), R^{BLReq}_{P} (Equation 5.3), and $R^{BJReq}_{P^{0}}$ (Equation 5.4) in Section 5.3.

Remark. FPSS introduces F-HandO , in both Section 5.2.3 and Section 5.2.4. As F-HandO employs the underlying blockchain to verify each of leaving and joining requests, if the pair of leaving and joining shareholders are honest, F-HandO will succeed eventually. As we assume that the underlying blockchain satis es the *liveness* property (see Section 2.1), F-HandO is *exible*.

Batched Hand-off

Initiate: $C^0 = ;; P_a = C_j; P_b = C_{j\theta}; P^0{}_a = C^0{}_j; P^0{}_b = C^0{}_{j\theta}$

Input: ($_{P_a}$; $_{P_b}$; t; C; PKs^e_C; $PK^l_{P_a}$; $PK^l_{P_b}$; $PK^l_{P^\theta{}_a}$; $PK^e_{P^\theta{}_a}$; $PK^l_{P^\theta{}_b}$; $PK^e_{P^\theta{}_b}$)

Output : C^0

The leaving party P_a :

generates a t-degree polynomial ( ), s.t. (0) = 0;

For i 2 [1; n], calculates:

$RPs_{P_a}[i]$ = (i);

if i 6= j: $fCTR_i g_{P_a}[i] = Enc_{PKs^e_C[i]}(RPs_{P_a}[i])$;

calculate $^0{}_j$ = $_j$ + $RPs_{P_a}[j]$;

execute $CTS_{PK^l_{\mathcal{P}^\theta{}_a};\ ^\theta{}_j} = Enc_{pk_{\mathcal{P}^\theta s_a}}(\ ^0{}_j)$;

execute $CTSK_{PK^l_{\mathcal{P}^\theta{}_a};PK^e_{\mathcal{P}_a}} = Enc_{PK^l_{\mathcal{P}^\theta{}_a}}(SK^e_{P_a})$;

execute $_{P_a}$:= F-HandO ^{b}:Prf($^b_{P_a}$; $!^b_{P_a}$); I ^b_P and $!^b_P$ are de ned in Equation 5.3

ds_a := ($fCTR_i g_{P_a}$; $CTS_{PK^l_{\mathcal{P}^\theta{}_a};\ ^\theta{}_j}$; $CTSK_{PK^l_{\mathcal{P}^\theta{}_a};PK^e_{\mathcal{P}_a}}$);

send $LReq^b_{P_a}$:= (LR_{P_a}; $LRID_{P_a}$; $PK^l_{P^\theta{}_a}$; ds_a; $_{P_a}$) to blockchain;

The leaving party P_b :

we omit the detail here;

send $LReq^b_{P_b}$:= (LR_{P_b}; $LRID_{P_b}$; $PK^l_{P^\theta{}_b}$; ds_b; $_{P_b}$) to blockchain;

send each of CTR and ($CTSK_{PK^l_{\mathcal{P}^\theta{}_a};PK^e_{\mathcal{P}_a}}$; $CTS_{PK^l_{\mathcal{P}^\theta{}_a};\ ^\theta{}_j}$) to shareholders in the new committee, respectively;

The joined party $P^0{}_a$:

If received ($CTSK_{PK^l_{\mathcal{P}^\theta{}_a};PK^e_{\mathcal{P}_a}}$; $CTS_{PK^l_{\mathcal{P}^\theta{}_a};\ ^\theta{}_j}$) from P_a:

Decrypt $CTSK_{PK^l_{\mathcal{P}^\theta{}_a};PK^e_{\mathcal{P}_a}}$ and $CTS_{PK^l_{\mathcal{P}^\theta{}_a};\ ^\theta{}_j}$, and gain $PK^e_{P_a}$ and $^0{}_j$;

generate a t-degree polynomial ^{0}( ), s.t. ^{0}(0) = 0;

calculate $^{00}{}_j$ = $^0{}_j$ + $^0(j)$;

For i 2 [1; n]; i 6= j, execute:

$fCTR_i g_{RPs_{\mathcal{P}^\theta}}[i] = Enc_{PKs^r_C[i]}(\ ^\theta{}_i)$;

sign $_{PK^l_{\mathcal{P}^\theta{}_a}}$ = $Sign(SK^l_{P^\theta}; LRID_{LR_\mathcal{P}})$;

execute $_{P^\theta{}_a}$:= F-HandO ^{b}:Prf($^b_{P^\theta{}_a}$; $!^b_{P^\theta{}_a}$); I $^b_{P^\theta{}_a}$ and $!^b_{P^\theta{}_a}$ are de ned in Equation 5.4

ds_{a^θ} := $fCTR_i g_{RPs_{\mathcal{P}^\theta}}$;

send $JReq^b_{P^\theta{}_a}$:= ($LR_{P^\theta{}_a}$; $JRID_{P_a}$; $_{PK^l_{\mathcal{P}^\theta{}_a}}$; ds_{a^θ}; $_{P^\theta{}_a}$) to blockchain, and send each of $fCTRg_{P^\theta{}_a}$ to shareholders in the new committee, respectively, except for $P^0{}_a$ himself;

The joined party $P^0{}_b$:

No Joining request sent by $P^0{}_b$;

Blockchain Validation Check:

$isValid s_a$ = ($RVC(LReq^b_{P_a})$&&$RVC(JReq_{P^\theta{}_a})$);

If $isValid s_a$:

set C^0 = C=P_a [$P^0{}_a$;

only the F-HandO between P_a and $P^0{}_a$ succeed;

Figure 5.4: FPSS - Batched Hand-o

5.2.5 Reconstruction

Ideally, reconstruction can be fully executed o -chain. When a requester asks for the reconstruction of a secret s, shareholders of the current committee can directly send their newest shares to the requester. Requester can execute Lagrange Interpolation to reconstruct the secret s after received $t + 1$ honest shares. This communication cost is proportional to the number of honest shareholders, which is $O(t + 1)$.

However, in order to avoid receiving malicious shares, we also provide an on-chain approach. When a requester asks for the reconstruction of a secret s, the shareholders encrypt their secret keys using the requester's secret key and record it on-chain. As we assume that the adversary can compromise at most t shareholders in the same committee, at least $t + 1$ shareholders must release their secret keys. Once the requester receives at least $t + 1$ secret keys from the shareholders of the current committee, he can decrypt at least $t + 1$ of the newest honest shares and execute Lagrange Interpolation to reconstruct the secret s. The on-chain approach has an additional cost proportional to the number of shares being stored on-chain, which is also $O(n)$.

5.3 Proof of Correct F-HandOff

This section presents the detailed construction for proving the validity of a F-HandO request, which allows for the secure F-HandO of shares between subsets of honest shareholders. We present the proof system for both single and batched F-HandO requests. To provide an overview of the proof system, we discuss its key components and how they work together to ensure security and correctness rst.

Overview. The objective of the proposed proof system is to convince any third

party that a given encrypted point $(i; ct_i)$ satisfies that $CTS_i = Enc_{pk_i}(\ _i)$ and $_i = (i)$, where is a private polynomial and pk_i is the encryption key of the i^{th} shareholder. To achieve this objective, the proposed proof system leverages the KZG commitment and Verifiable Encryption (VE) scheme (see Sections 2.6 and 2.7, respectively) to verify that an encrypted value corresponds to a valid evaluation of the private polynomial at a specific point.

By using KZG commitments to commit to the coefficients of the private polynomial and VE scheme to encrypt the values of the polynomial at specific points, the proof system can verify that a given ciphertext corresponds to a valid encryption of the polynomial evaluated at the specified point. The KZG commitment provides a commitment to the polynomial coefficients, which can be used to generate the expected evaluation of the polynomial at the given point. This evaluation is then compared to the decrypted value to ensure correctness, allowing for efficient and secure verification of encrypted values without revealing the private polynomial or the specific evaluation point.

Single F-HandOff.

We specify the proving system of the single F-HandOff relations (see Equation 5.1 and Equation 5.2).

Let G be a bilinear group of prime order p and generator g for KZG commitment scheme. Let d be a trapdoor generated by a trusted third party. This system setup is generated in the Setup phase (see Section 5.2.1).

For the leaving party.

1. **KZG Proving Phase**:

First, the leaving party generates a KZG proof of knowledge of the t polynomial (x). The KZG proof consists of a set of evaluation proofs of the polynomial at specific points in the field. The KZG commitment scheme used in the proof provides a commitment to the polynomial coefficients, which can be computed

as KZG.cmt $= g^{(d)}$. The KZG commitment KZG.cmt can be used to verify that $(i) = {}_i$ without revealing the polynomial. Therefore, proving that $(0) = 0$ simply follows from the KZG commitment scheme.

However, for the evaluation of (i) where $i \in [1; n]$, the leaving party must also provide a commitment to the evaluation point. This is to ensure that the evaluation remains hidden and secure. The commitment of (i) can then be used in the veri cation phase to verify that the re-randomized share satis es the required equation. By providing these commitments, the leaving party can ensure the correctness and security of the reshare process.

Second, to prove that the re-randomized share ${}^0_j$ assigned to the joining party satis es ${}^0_j = {}_j + (j)$, the leaving party must provide a proof. Since KZG commitments have homomorphic properties, if the leaving party's share ${}_j$ and the re-randomized share ${}^0_j$ satisfy ${}^0_j = {}_j + (j)$, the commitment can be computed as:

$$\text{KZG.cmt}_{{}^0_i} = \text{KZG.cmt}_{{}_i} \quad \text{KZG.cmt}_{(j)};$$

Recall that we require that the commitment of each updated share should be recorded on-chain. The commitment KZG.cmt ${}_i$ can be veri ed by comparing with on-chain record.

2. **Encryption Phase**: Next, the leaving party encrypts the shares using a veri able encryption (VE) scheme, which provides both con dentiality and authenticity for encrypted shares. The encryption proofs for both ${}_i = f(i)$ and ${}^0_j$ are identical. So we use ${}_i$ as an example in the following description. Speci cally, the leaving party encrypts each of the points $ct_i = \text{Enc}_{pk_i}({}_i)$, and creates the zero-knowledge proof ${}^{VE}_i$ that proves the KZG.cmt ${}_{()}$ is a commitment to the polynomial $()$, such that ${}_i = (i)$ without revealing ${}_i$ or $()$.

3. **Verification Phase**: The veri er uses the KZG and VE schemes to verify

that ct_i is the encryption of $_i = (i)$ as follows:

- KZG Veri cation: The veri er checks the KZG proof generated in the rst step to ensure that the commitments $g^{\ i}$ correspond to the evaluations $(i) = _i$ of the polynomial at the speci c points i.
- VE Veri cation: The veri er checks the validity of the encryption by using the commitments $g^{\ i}$, ciphertext ct_i and $_i^{VE}$.

For more details and implementation considerations, refer to the preliminary section (Section 2.6 and 2.7) and the original KZG paper [37] and the Camenisch-Shoup veri able encryption scheme paper [38].

By combining KZG and VE, we can verify that a ciphertext ct_i is the encryption of a point (i) without revealing the private polynomial $( )$ and the evaluation point (i) to the veri er.

For the joining party.

The proof required by the joining party is simpler than the leaving party's proof, as the joining party only needs to prove that the polynomial they generated satis es $^{0}(0) = 0$, and the evaluation of each point $^{0}_i = ^{0}(i)$, where $i \ 2 \ [1; n]$ and $i \ 6= \ j$. Since the description of the proof generation process is already covered by the leaving party's proof above, we will omit it here to avoid redundancy.

Batched F-HandOff

The proof system for batched leaving is similar to that of Single F-HandO , with an additional step for verifying the ephemeral secret key. Since the leaving party needs to release their ephemeral secret key to the joining party for accessing the points assigned to them, the leaving party encrypts their ephemeral secret key using the joining party's long term secret key and generates an encryption proof for the correctness of their ephemeral secret key. This proof can be generated using the veri able encryption scheme directly. By including this step, the proof

system ensures that the joining party has access to the correct ephemeral secret key, allowing them to securely access the point assigned to the leaving party from among concurrent leaving requests.

For the batched joining proof, it is identical to the leaving joining proof, we omit here for avoid redundancy.

5.4 FPSS Security Discussion

We assume that there are at least $t + 1$ honest shareholders in the committee, and the formal security is de ned in Theorem 5.4.1 below.

Theorem 5.4.1. *An FPSS scheme that satis es both robustness and secrecy properties is secure.*

We then provide the proof sketch of the security properties, robustness and secrecy, that FPSS required.

FPSS Robustness Proof

We adopt the proof idea of Robustness from the work by Goyal et al. [20].

For the single hand-o of FPSS, the Robustness can be proved by Lemma 5.4.1, 5.4.2, 5.4.3.

Lemma 5.4.1 (Secret Integrity)**.** *Given a secret s and its commitment* cmt_s*, with overwhelming probability, once s is successfully distributed among a committee* C *by uploading a distribution transaction on-chain, the secret s will never be modi ed for each single hand-o .*

Proof: Since we assume the underlying blockchain satis es *persistence* (Section 2.1), a transaction can be recorded on-chain only if it is valid. Thus, if a distributed transaction is recorded on-chain, each shareholder P in the committee C owes a valid share $_i = f(i)$ of the secret s, where $f(0) = s$.

Assuming that $f\ _i g_{i2[1;n]}$ is a set of shares owing by shareholders in the committee C respectively. Once a single hand-o is executed successfully, namely a pair of leaving and joining requests are recorded on-chain, the set of shares becomes $f\ ^0{}_i g_{i2[1;n]}$. where $^0{}_i = \ _i + \ (i) + \ ^0(i)$, $(0) = 0$, and $^0(0) = 0$. Thus, the value $s^0 = f(0) + \ (0) + \ ^0(0)$ equals to the original secret s committed in cmt_s.

Lemma 5.4.2 (Honest Share Integrity). *Given a secret s and its commitment* cmt_s*, with overwhelming probability, after each of the single F-HandO , each honest party in the new committee* C^0 *obtain the correct share of the given s. For any subset* U C*, the share provided by parties in* U *can only be reconstructed to either the given secret s or ?, where ? denotes a failure in the reconstruction. In particular, if* U *contains at least* $t + 1$ *honest parties, then the shares provided by parties in* U *can always be reconstructed (by anyone with a valid secret reconstruction request) to the given secret s.*

Proof: Since we assume that the underlying blockchain satis es the persistence properties (see Section 2.1), a transaction can be recorded on-chain only if it is valid (accepted by the honest party).

For the newly joined party C^0_j, his receives a share $(\ _j + \ (j))$, where $\ _j$ is a valid share of the secret s in the old committee and $(0) = 0$. He will compute the new share $^0{}_j$ by adding $^0(j)$, where $^0(0) = 0$.

For each of the remaining parties C^0_i, where $i\ 2\ [1; n]=j$ and $C_i = C^0_i$, he will receive two points (i) and $^0(i)$. Thus, their updated share is $^0{}_i = \ _i + \ (i) + \ ^0(i)$, where $\ _i$ is a valid share of the old committee.

If the leaving or joining request is recorded on-chain, the correctness of these corresponding shares is veri ed by blockchain validators.

Since there is at least $t + 1$ honest shareholders in the current committee U, the secret s committed in cmt_s can always be reconstructed by $t + 1$ valid shares

from the current committee.

Lemma 5.4.3 (Robustness of Single Hand-O). *Given a secret s and its commitment* cmt_s*, with overwhelming probability, the valid requester in Reconstruction can always reconstruct the s committed in* cmt_s.

Proof: From the above two Lemmas, Lemma 5.4.1 and Lemma 5.4.2, we conclude that the current committee contains at least $t+1$ honest parties with $t+1$ valid shares respectively. So that any honest party with valid secret request can reconstruct the secret from the current committee. Thus, the lemma is proved.

Lemma 5.4.4 (Robustness of Batched Hand-O). *Given a secret s and its commitment* cmt_s*, with overwhelming probability, the valid requester in Reconstruction can always reconstruct the s committed in* cmt_s *under the Batched Hand-O case.*

The proof of batched hand-o is similar to single hand-o , thus we omit it here.

FPSS Secrecy Proof

The proof of Secrecy is the same for both the single hand-o and batched hand-o .

Lemma 5.4.5 (Secrecy). *If the adversary can corrupt at most t shareholders in the same committee, FPSS satis es the secrecy property.*

Proof: Let S be a simulator and A be the probabilistic polynomial-time (PPT) adaptive adversary. The goal of adversary A is to output a guess b the distributed secret s. The simulator's goal is to output a guess b' that is indistinguishable from the adversary's guess b. Let L( ) be the functionality of Blockchain. Since robustness proof already proves that the secret can always been recovered after several successful F-HandO , we do not consider the veri cation

of F-HandO requests in the secrecy proof again. We simplify the Blockchain Functionality (see Figure 5.5 [73]) required in following proof.

(Simplified) Ideal Functionality L( )

This functionality stores all the published transactions in the ledger. Once a valid transaction is posted, it takes at most rounds to be added in the ledger.

Post a transaction: Upon (post; tx) ^{0}- P, publish tx on the ledger L in round $_{1}$ $_{0}$ + .

Figure 5.5: Simpli ed Blockchain Functionality

All transactions published in the ledger L, can be publicly accessed, so A can access all encrypted shares on-chain. Let ^ be the advantage of guessing the secret s in the following FPSS-Secrecy-Game. We will show that the simulator S has the same advantage in guessing the secret s correctly as A. The FPSS-Secrecy-Game is de ned as follows:

Setup Game:

- The challenger runs the setup algorithm at time to generate the public parameters and shares them with all parties participating in this protocol;
- The challenger chooses a secret value s, and divides them into n shares, such that [$_{1}$;:::; $_{n}$];
- The challenger updates all shareholders' secret and private key pairs at time , and encrypts these shares by using each shareholder's public key and distributes these encrypted shares to these shareholders, respectively. So that any t+1 shares can be used to reconstruct s.
- The challenger records the commitment of s and all encrypted shares of s, [$_{1}$;:::; $_{n}$], on L.

Single F-HandOff Game

- The adversary A chooses a set of up to t corrupted parties within $_c$ time interval and obtains their plaintext shares of s at time .
- At the time , the simulator S randomly chooses a shareholder P_i, where $i \in [1; n]$, to be the leaving party. There are two cases:
  1. if P_i is uncompromised:
     - S generates a polynomial (x) of degree t such that $(0) = 0$, and chooses n points on the polynomial, such that $(1); : : : ; (n)$;
     - S updates P_i's share by computing $^0_i = _i + (i)$, where $_i$ is P_i's original share.
     - S encrypts ^0_i by using P^0's encryption key and n 1 points, (j), where $j \in [1; n]; j \neq i$, by using other shareholders' encryption keys, respectively.
     - S sends a transaction including the encrypted share ^0_i and these encrypted points (j), where $j \in [1; n]; j \neq i$, to L.
  2. if P_i is compromised:
     - S generates a polynomial (x) of degree t such that $(0) = 0$, and chooses n points on the polynomial, such that $(1); : : : ; (n)$.
     - S updates P_i's share by computing $^0_i = _i + (i)$.
     - S encrypts ^0_i by using P^0's encryption key and n 1 points, (j), where $j \in [1; n]; j \neq i$, by using other shareholders' encryption key, respectively.
     - S sends a transaction including the encrypted share ^0_i and these encrypted points (j), where $j \in [1; n]; j \neq i$, to L.
     - S sends the plaintext ^0_i to A.

- Each shareholder P_j, where $j \neq i$, receives the corresponding point (j), and updates his share by computing $'_j = _j + f(j)$;
- The new shareholder P'_i receives the updated share $'_i$ from P_i. There are three cases:
  1. if P'_i is uncompromised:
     - S generates a polynomial $'(x)$ of degree t such that $'(0) = 0$, and chooses n points on the polynomial, such that $'(1); : : : ; '(n)$. We do not distinguish whether P_i is compromised or not under this case, since P_i does not affect P'_i's behaviour.
     - S updates P''s share by computing $''_i = '_i + '(i)$.
     - S encrypts $''_i$ by using P''s encryption key and $n - 1$ points, $'(j)$, where $j \in [1; n]; j \neq i$, by using other shareholders' encryption keys, respectively.
     - S sends a transaction including the encrypted share $''_i$ and these encrypted points $'(j)$, where $j \in [1; n]; j \neq i$, to L.
  2. if P'_i is compromised, but P_i is uncompromised:
     - S generates a polynomial $'(x)$ of degree t such that $'(0) = 0$, and chooses n points on the polynomial, such that $'(1); : : : ; '(n)$. We do not distinguish whether P_i is compromised or not under this case, since P_i does not affect P'_i's behaviour.
     - S updates P''s share by computing $''_i = '_i + '(i)$.
     - S sends $''_i$ to A.

     If A already corrupts a set of t shareholders, P_i is not included, the simulator S halts. That is once P' is corrupted, A will have $t + 1$ shares in the new committee. While this case is contradict to our assumption, such that A can corrupt at most t shareholders for every $_c$ interval.

3. if both P'_i and P_i are all compromised:
   - S generates a polynomial $'(x)$ of degree t such that $'(0) = 0$, and chooses n points on the polynomial, such that $'(1); : : : ; '(n)$.
   - S updates P''s share by computing $''_i = _i + '(i)$.
   - S encrypts $''_i$ by using P''s encryption key and n 1 points, $'(j)$, where $j \in [1; n]; j \neq i$, by using other shareholders' encryption keys, respectively.
   - S sends a transaction including the encrypted share $''_i$ and these encrypted points $'(j)$, where $j \in [1; n]; j \neq i$, to L.
   - S sends $''_i$ to A.

- S sends the corresponding points $'(j)$ to each shareholder P_j where $j \neq i$, who updates their share by computing $''_j = _j + '(j)$.
- Adversary A continues to corrupt additional shareholders (if any) and receives their shares.

Guess Game. Adversary A outputs a guess for the value of the secret s, and the simulator outputs the same guess.

Proof of Indistinguishability. We claim that the simulator S is indistinguishable from the actual game.

Case 1: Both P_i and P'_i is not corrupted. In this case, S proceeds as in the proof for the case when P_i is not compromised and P'_i is not compromised, which has already been shown to be indistinguishable from the actual game.

Case 2&3: P_i is corrupted and P'_i is corrupted, or P_i is corrupted but P'_i is not corrupted. In this cases, S proceeds as described above. Since S chooses at most t shareholders to corrupt uniformly at random and receives their shares, the shares held by the adversary A are indistinguishable from those in the actual game. Moreover, the shares distributed to the shareholders not chosen to perform the

hand-o are also indistinguishable from the actual game, since they are generated using the same polynomial as in the actual game. The simulator S then proceeds to generate a new share for the new shareholder P'_i using the same polynomial as in the actual game.

Case 4: P_i is not corrupted, but P'_i is corrupted. In this case, if S already chooses t shareholders except for P_i to corrupt uniformly at random and receives their shares, the program halts. Otherwise, since all corrupted shareholders including P'_i are chosen uniformly at random, the shares held by the adversary A are indistinguishable from those in the actual game.

Since shareholders update their secret and public key pairs every $_c$ time interval and the encryption scheme used in the protocol is semantically secure, the encrypted shares of the secret s and points of the polynomials, ( ) and $'$( ), recorded on L are indistinguishable from random strings. Therefore, the updated shares $'_i$ and $''_i$ received by the new shareholder P'_i is indistinguishable from a valid share of a random string.

Similarly, the points $'(j)$ received by the shareholders not chosen to perform the hand-o are indistinguishable from random strings, since they are generated using the same polynomial as in the actual game. Therefore, the updated share $''_j$ held by each of these shareholders is indistinguishable from a valid share of a random string.

Hence, the shares held by the adversary A are indistinguishable from those in the actual game, and the guess output by S is the same as the guess output by A. Therefore, the simulator S is indistinguishable from the actual game.

Let F_{sec} be the FPSS-Secrecy-Game. Since the advantage of the adversary A in the F_{sec} is at most ^, the simulator S is indistinguishable from the actual game, we have:

$$jPr[F_{sec}(A)] = 1j \quad Pr[F_{sec}(S)] = 1j \quad ^$$

, which implies that the FPSS protocol is secure against an adaptive adversary that can corrupt up to t shareholders.

5.5 Performance

This section introduces our proof-of-concept implementation of FPSS and the corresponding evaluation analysis.

5.5.1 Implementation

Our implementation is based on Churp [74] and the Coinbase's advanced cryptography library, kryptology [75], GNU multiple precision arithmetic Library (GMP) [76], and the Pairing-Based Cryptography Library (PBC) [77] for the cryptographic primitives, and gRPC [78] for network infrastructure.

We use the following setting same as Churp. For polynomial arithmetic, we used the polynomial ring $F_p[x]$ for 256-bit prime p. For the KZG commitment scheme, we used type A pairing on an elliptic curve $y^2 = x^3 + x$ over F_p for a 512-bit q. The order of the EC group is also p. We also use SHA-256 for hashing. For the public key encryption scheme, we use the group Z_n for 2048-bit composite group order n. For the veri able encryption scheme, we used a cyclic group G_p of the same prime order p.

FPSS can be deployed on either a permissioned or a permissionless blockchain. We use a blockchain simulator as Churp did. That is, we abstract away the speci c choice and simulate one using a trusted node. It will incur additional latency when deploying FPSS on a Blockchain. Our performance analysis is based on the batched F-HandO , since it can also be used for a single shareholder update.

5.5.2 Evaluation

Setup. Our experiments are run on a macOS with processor 2.6 GHz 6-Core Intel Core i7 and memory 16GB 2400 MHz DDR4.

On-chain communication complexity. The de nition of on-chain cost is adopted from CHURP [21]. That is the number of bits written to the blockchain. FPSS employs blockchain validators (a.k.a miners) for verifying the on-chain requests. As FPSS requires all ciphertexts to be recorded on-chain, which is about 1:5 2 n kb on-chain per F-HandO , where n is the size of the committee. FPSS can be deployed on either permissionless (such as Filecoin [79]) and permissioned (such as Corda [80]) blockchain. While choosing permissioned blockchain can reduce the on-chain cost.

Off-chain data size. F-HandO has two rounds, which incurs an $O(m)$ communication complexity on-chain and an $O(n \quad m)$ communication complexity o -chain, where n is the committee size and m is the number of leaving shareholders. The o -chain data transmitted per leaving node includes: a KZG commitment of a polynomial, n KZG proof (consisting of n commitment of polynomial points and the corresponding witnesses) and $n+1$ ciphertext (consisting of n encrypted polynomial points and the corresponding proof, and an ciphertext on ephemeral secret key and its proof). The o -chain data transmitted per joining node includes: a KZG commitment of polynomial, $n \quad 1$ KZG proof (consisting of $n \quad 1$ commitment of polynomial points and the corresponding witnesses) and $n \quad 1$ ciphertext on shares to each shareholders in the new committee, except for himself.

Fig. 5.6 shows that in a large committee, FPSS reduces o -chain communication for shareholders to update their shares. Thus, for the update of a single shareholder, the data required to be transmitted o -chain is about $6n + 6$ kB. As a result, when running a 1000-shareholder committee, it takes about 6 MB

of data for FPSS to be transmitted o -chain and 226 MB for CHURP for each single shareholder update, which is reduced to about 37 o -chain data transmission. While FPSS is not always more e cient than CHURP. For example, when the number of shareholders needing to update their shares is greater than 100 in a 200-shareholder committee, CHURP requires less o -chain data to be transmitted than FPSS. In conclusion, a committee with a majority of members who change frequently could use CHURP to update the shares of the committee. If we consider the scenario where not more than half of the committee members may leave at any time, FPSS would be the better choice to reduce o -chain data transmission.

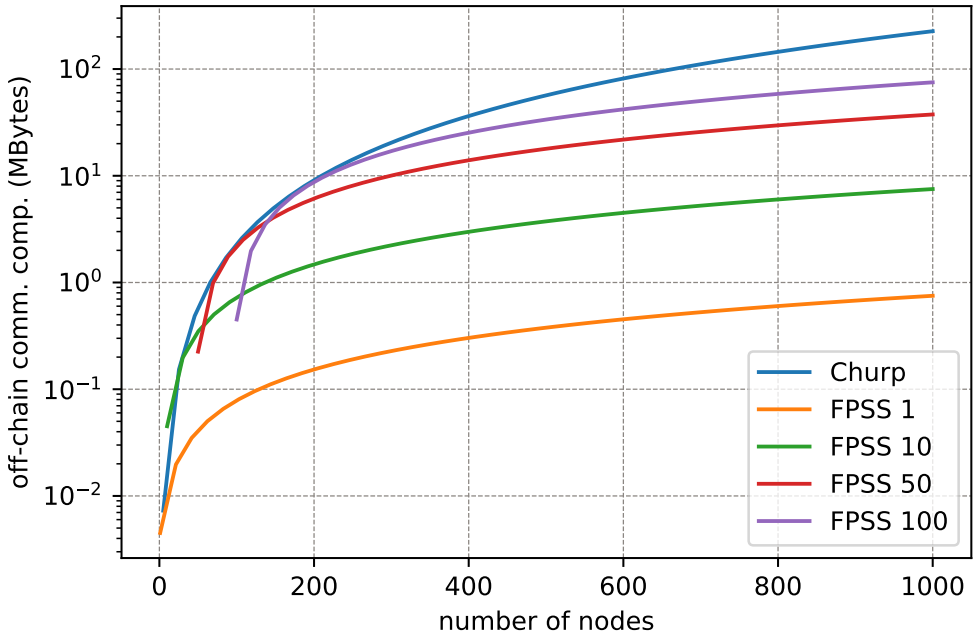

Figure 5.6: O -Chain Communication Complexity for FPSS and Churp. We depict four lines for FPSS with the number of shareholders needing to update their shares being 1, 10, 50, and 100, respectively.

Network transmission latency. For a single F-HandO , the leaving party will broadcast a leaving request to blockchain and the encrypted shares to share-

holders, respectively, in the new committee; as soon as the joining party receives a share from the leaving party, he will broadcast a joining request to the blockchain and the encrypted shares to the entire new committee except for himself. According to our Blockchain model in Section 5.1.1, the underlying blockchain satisfies the *liveness* property. Namely, if a request is valid, it will eventually be recorded on-chain. In addition, F-HandO does not require acknowledgment messages to be responded from all receivers to the sender (the leaving or joining party). Let D_L be the transmission latency, which is the time that an encrypted share (about 1.5 kb) is sent from the leaving party to the joining party, and let D_J be the latency, which is the time that the joining request (about $3n + 3$ kb) is sent from the joining party to the blockchain. In the optimal scenario, the data transmission latency of a single F-HandO is $D_L + D_J$. Since each F-HandO is independent of other concurrent ones, and multiple concurrent F-HandO s can be executed in parallel. If every participant uses the Ethernet cabling, the LAN speeds are typically 1000 mbps and WAN are 150 mbps [81]. For a committee with 100 shareholders, the data transmission latency is about 0.3 ms to transmit 0.3 MB data in a LAN setting and about 2 ms in a WAN setting.

Microbenchmarks. We also evaluate the cost of core verifications in our implementation of the KZG polynomial commitment scheme and the verifiable encryption scheme. The results are presented in Table 5.3. For the KZG scheme, the setup refers to the time required to generate a public key for a polynomial with a given degree. For the VE scheme, the setup refers to the time of generating the system parameter n=pq, where p and q are safe primes. They are all the one time cost across all committed polynomials.

Table 5.3: Cost of KZG and VE Scheme

	Degree		
	6	26	49
(KZG) Setup	0.06 s	0.2 s	0.4 s
(KZG) Poly Commit	0.06 ms	0.3 ms	0.7 ms
(KZG) Point Commit	0.02 ms	0.09 ms	0.1 ms
(KZG) Point Eval	3.7 ms	3.4 ms	3.5 ms
	Num of Nodes		
	10	50	100
(VE) Setup	0.02 s		
(VE) EncryptAndProve	0.98 s	4.7 s	9.5 s
(VE) Cipher Verify	0.72 s	3.5 s	7.0 s

5.6 Conclusion

This chapter introduces Flexible Proactive Secret Share (FPSS) scheme, a versatile solution that facilitates the provision of the distributed service required in AuxChannel and MoNet. With FPSS, updates only necessitate the interaction of pairs of shareholders who are leaving and joining, ensuring an e cient process. The scheme incorporates an additional algorithm, F-HandO , enabling e ective membership updates.

Moreover, FPSS exhibits adaptability for deployment in other applications beyond payment channels. It can serve as a fundamental building block for diverse use cases, including Distributed Key Generation (DKG), Byzantine Fault Tolerance (BFT), and more. FPSS showcases its potential for broader applications, contributing to the versatility and robustness of modern blockchain systems.

Chapter 6

Future Direction and Conclusion

This thesis proposes a novel bi-directional payment channel protocol called AuxChannel, which represents the first of its kind for scriptless blockchains. This protocol allows for duplex payments in a channel that can remain operational indefinitely, thereby overcoming the time limitations of existing payment channels. Our research also included the adaptation of the AuxChannel protocol to the Monero blockchain, which we named MoChannel. To further enhance the functionality of MoChannel, we developed MoNet, which enables multi-hop payments and further expands the potential use cases of this innovative protocol.

In addition to the development of these groundbreaking protocols, we also proposed a proactive secret share scheme called FPSS. This scheme enables flexible hand-off in a way that involves only the participation of the shareholders who are leaving or joining the network, thereby streamlining the process and minimizing disruption to the system. We believe that these design concepts and primitives represent new directions for the development of payment channels using scriptless blockchains.

Furthermore, our study demonstrated that the primitives we developed, including CVES, VCOF, GCAS, and FPSS, are versatile and can be applied to other applications beyond payment channels. For example, CVES (Cryptographic

Vector Commitment Scheme) can be used in cryptographic protocols that require commitments to vectors, while VCOF (Verifiable Committed Oblivious Function) can be used for private set intersection and private information retrieval. Additionally, GCAS (Group Communication Authentication Scheme) can be used for group communication applications that require secure and authenticated messaging.

Overall, the findings of this study contribute significantly to the development of payment channels using scriptless blockchains and provide valuable insights into the potential applications of the primitives we developed. We hope that these advancements will inspire future research and development in this exciting and rapidly evolving field.

References

[1] Monero, \Monero," 2014. 2

[2] Zcash, \Zcash," 2016. 2

[3] L. Fournier, \One-time veri ably encrypted signatures aka adaptor signatures," 2019. 5, 30

[4] P. Moreno-Sanchez, A. Blue, D. V. Le, S. Noether, B. Goodell, and A. Kate, \DLSAG: non-interactive refund transactions for interoperable payment channels in monero," in *Financial Cryptography*, vol. 12059 of *Lecture Notes in Computer Science*, pp. 325{345, Springer, 2020. 8, 17

[5] Y. Zhang, Y. Long, Z. Liu, Z. Liu, and D. Gu, \Z-channel: Scalable and e cient scheme in zerocash," *Comput. Secur.*, vol. 86, pp. 112{131, 2019. 8, 15, 17

[6] S. A. K. Thyagarajan, G. Malavolta, F. Schmidt, and D. Schröder, \Paymo: Payment channels for monero," 2020. 8

[7] S. A. K. Thyagarajan, T. Gong, A. Bhat, A. Kate, and D. Schröder, \Opensquare: Decentralized repeated modular squaring service," pp. 3447{3464, 2021. 9, 59

[8] M. Green and I. Miers, \Bolt: Anonymous payment channels for decentralized currencies," in *CCS*, pp. 473{489, ACM, 2017. 9, 15

[9] L. Aumayr, O. Ersoy, A. Erwig, S. Faust, K. Hostakova, M. Ma ei, P. Moreno-Sanchez, and S. Riahi, \Generalized bitcoin-compatible channels," *IACR Cryptol. ePrint Arch.*, p. 476, 2020. 9, 11

[10] L. Aumayr, S. A. K. Thyagarajan, G. Malavolta, P. Monero-Sanchez, and M. Ma ei, \Sleepy channels: Bitcoin-compatible bi-directional payment channels without watchtowers," *IACR Cryptol. ePrint Arch.*, p. 1445, 2021. 10

[11] M. H. Satoshi Nakamoto, \Anti DoS for tx Repleasement - Nakamoto's Idea of High-frequency Trades," 2013. 10, 16

[12] J. Spilman, \Anti DoS for tx repleasement - spilman-styled payment channel," 2013. 10, 16

[13] F. Tschorsch and B. Scheuermann, \Bitcoin and beyond: A technical survey on decentralized digital currencies," *IEEE Commun. Surv. Tutorials*, vol. 18, no. 3, pp. 2084{2123, 2016. 10, 16

[14] J. Poon and T. Dryja, \The Bitcoin lightning network: scalable o -chain instant payments," 2016. 11, 15, 16, 17, 48, 58, 61, 79

[15] C. Decker and R. Russell, \eltoo : A simple layer 2 protocol for bitcoin," 2018. 11, 17

[16] J. Gugger, \Bitcoin-Monero cross-chain atomic swap," 2020. 11, 18

[17] O. Shlomovits and O. Leiba, \Jugglingswap: Scriptless atomic cross-chain swaps," *CoRR*, vol. abs/2007.14423, 2020. 11, 18

[18] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Ma ei, \Anonymous multi-hop locks for blockchain scalability and interoperability," in *NDSS*, The Internet Society, 2019. 11, 18, 78

[19] F. Benhamouda, C. Gentry, S. Gorbunov, S. Halevi, H. Krawczyk, C. Lin, T. Rabin, and L. Reyzin, \Can a public blockchain keep a secret?," in *TCC (1)*, Lecture Notes in Computer Science, Springer, 2020. 12, 96

[20] V. Goyal, A. Kothapalli, E. Masserova, B. Parno, and Y. Song, \Storing and retrieving secrets on a blockchain," in *Public Key Cryptography (1)*, Lecture Notes in Computer Science, Springer, 2022. 12, 19, 96, 118

[21] S. K. D. Maram, F. Zhang, L. Wang, A. Low, Y. Zhang, A. Juels, and D. Song, \CHURP: dynamic-committee proactive secret sharing," in *CCS*, pp. 2369{2386, ACM, 2019. 12, 96, 127

[22] E. Kokoris-Kogias, E. C. Alp, L. Gasser, P. Jovanovic, E. Syta, and B. Ford, \CALYPSO: private data management for decentralized ledgers," *Proc. VLDB Endow.*, vol. 14, no. 4, pp. 586{599, 2020. 13

[23] M. Jourenko, K. Kurazumi, M. Larangeira, and K. Tanaka, \Sok: A taxonomy for layer-2 scalability related protocols for cryptocurrencies," *IACR Cryptol. ePrint Arch.*, p. 352, 2019. 15

[24] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, \Perun: Virtual payment hubs over cryptocurrencies," in *IEEE Symposium on Security and Privacy*, pp. 106{123, IEEE, 2019. 15, 47, 55

[25] R. Khalil and A. Gervais, \NOCUST - A non-custodial 2nd-layer nancial intermediary," *IACR Cryptol. ePrint Arch.*, p. 642, 2018. 15

[26] \Raiden Network." https://raiden.network/. 15

[27] J. Lind, I. Eyal, P. R. Pietzuch, and E. G. Sirer, \Teechan: Payment channels using trusted execution environments," *CoRR*, vol. abs/1612.07766, 2016. 15

[28] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, \Sprites and state channels: Payment networks that go faster than lightning," in *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers* (I. Goldberg and T. Moore, eds.), vol. 11598 of *Lecture Notes in Computer Science*, pp. 508{526, Springer, 2019. 16

[29] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, \Perun: Virtual payment hubs over cryptocurrencies," in *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pp. 106{123, IEEE, 2019. 16

[30] J. Coleman, L. Horne, and L. Xuanji, \Counterfactual: Generalized state channels," *Acessed: Nov*, vol. 4, p. 2019, 2018. 16

[31] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Ma ei, \Silentwhispers: Enforcing security and privacy in decentralized credit networks," in *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*, The Internet Society, 2017. 16

[32] V. Sivaraman, S. B. Venkatakrishnan, M. Alizadeh, G. C. Fanti, and P. Viswanath, \Routing cryptocurrency with the spider network," in *HotNets*, pp. 29{35, ACM, 2018. 16

[33] \Amp: Atomic multi-path payments over lightning." https://lightning.network/. 16

[34] L. Aumayr, O. Ersoy, A. Erwig, S. Faust, K. Hostakova, M. Ma ei, P. Moreno-Sanchez, and S. Riahi, \Generalized channels from limited

blockchain scripts and adaptor signatures," in *ASIACRYPT (2)*, vol. 13091 of *Lecture Notes in Computer Science*, pp. 635{664, Springer, 2021. 17

[35] S. A. K. Thyagarajan, G. Malavolta, F. Schmidt, and D. Schröder, \Paymo: Payment channels for monero," *IACR Cryptol. ePrint Arch.*, p. 1441, 2020. 18, 59, 69

[36] A. Erwig, S. Faust, K. Hostakova, M. Maitra, and S. Riahi, \Two-party adaptor signatures from identi cation schemes," in *Public Key Cryptography (1)*, vol. 12710 of *Lecture Notes in Computer Science*, pp. 451{480, Springer, 2021. 20

[37] A. Kate, G. M. Zaverucha, and I. Goldberg, \Constant-size commitments to polynomials and their applications," in *ASIACRYPT*, vol. 6477 of *Lecture Notes in Computer Science*, pp. 177{194, Springer, 2010. 20, 21, 117

[38] J. Camenisch and V. Shoup, \Practical veri able encryption and decryption of discrete logarithms," in *CRYPTO*, vol. 2729 of *Lecture Notes in Computer Science*, pp. 126{144, Springer, 2003. 23, 117

[39] S. Goldwasser, S. Micali, and C. Racko , \The knowledge complexity of interactive proof systems," *SIAM J. Comput.*, vol. 18, no. 1, pp. 186{208, 1989. 41

[40] M. Blum, P. Feldman, and S. Micali, \Non-interactive zero-knowledge and its applications (extended abstract)," in *STOC*, pp. 103{112, ACM, 1988. 41

[41] F. Boudot, \E cient proofs that a committed number lies in an interval," in *EUROCRYPT*, vol. 1807 of *Lecture Notes in Computer Science*, pp. 431{444, Springer, 2000. 46

[42] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *IEEE Symposium on Security and Privacy*, pp. 315–334, IEEE Computer Society, 2018. 46

[43] J. Camenisch, R. Chaabouni, and A. Shelat, "Efficient protocols for set membership and range proofs," in *ASIACRYPT*, vol. 5350 of *Lecture Notes in Computer Science*, pp. 234–252, Springer, 2008. 46

[44] I. Damgard and E. Fujisaki, "An integer commitment scheme based on groups with hidden order," *IACR Cryptol. ePrint Arch.*, p. 64, 2001. 46

[45] Xlab, "Library for zero-knowledge proof based applications (like anonymous credentials)." https://github.com/xlab-si/emmy, 2020. 46, 89

[46] E. Abebe, Y. Hu, A. Irvin, D. Karunamoorthy, V. Pandit, V. Ramakrishna, and J. Yu, "Verifiable observation of permissioned ledgers," in *IEEE ICBC*, pp. 1–9, IEEE, 2021. 48

[47] A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. Moreno-Sanchez, A. Kiayias, and W. J. Knottenbelt, "Sok: Communication across distributed ledgers," in *Financial Cryptography (2)*, vol. 12675 of *Lecture Notes in Computer Science*, pp. 3–36, Springer, 2021. 48

[48] R. Pass and K. Pietrzak, eds., *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part I*, vol. 12550 of *Lecture Notes in Computer Science*, Springer, 2020. 49, 52

[49] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "Sok: Layer-two blockchain protocols," in *Financial Cryptography*, vol. 12059 of *Lecture Notes in Computer Science*, pp. 201–226, Springer, 2020. 52

[50] A. Mirzaei, A. Sakzad, J. Yu, and R. Steinfeld, \FPPW: A fair and privacy preserving watchtower for bitcoin," in *Financial Cryptography (2)*, vol. 12675 of *Lecture Notes in Computer Science*, pp. 151{169, Springer, 2021. 52

[51] Z. Avarikioti, O. S. T. Litos, and R. Wattenhofer, \Cerberus channels: Incentivizing watchtowers for bitcoin," in *Financial Cryptography*, vol. 12059 of *Lecture Notes in Computer Science*, pp. 346{366, Springer, 2020. 52

[52] A. Mizrahi and A. Zohar, \Congestion attacks in payment channel networks," in *Financial Cryptography (2)*, vol. 12675 of *Lecture Notes in Computer Science*, pp. 170{188, Springer, 2021. 58

[53] S. Tikhomirov, P. Moreno-Sanchez, and M. Ma ei, \A quantitative analysis of security, anonymity and scalability for the lightning network," in *EuroS&P Workshops*, pp. 387{396, IEEE, 2020. 58

[54] S. Mazumdar, P. Banerjee, and S. Ruj, \Time is money: Countering grie ng attack in lightning network," in *TrustCom*, pp. 1036{1043, IEEE, 2020. 58

[55] G. Kappos, H. Yousaf, A. M. Piotrowska, S. Kanjalkar, S. Delgado-Segura, A. Miller, and S. Meiklejohn, \An empirical analysis of privacy in the lightning network," in *Financial Cryptography (1)*, vol. 12674 of *Lecture Notes in Computer Science*, pp. 167{186, Springer, 2021. 61

[56] A. Biryukov, G. Naumenko, and S. Tikhomirov, \Analysis and probing of parallel channels in the lightning network," *IACR Cryptol. ePrint Arch.*, p. 384, 2021. 61

[57] J. K. Liu, V. K. Wei, and D. S. Wong, \Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract)," in *ACISP*, vol. 3108 of *Lecture Notes in Computer Science*, pp. 325{335, Springer, 2004. 69

[58] J. Camenisch and A. Lysyanskaya, \A formal treatment of onion routing," in *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings* (V. Shoup, ed.), vol. 3621 of *Lecture Notes in Computer Science*, pp. 169{187, Springer, 2005. 78

[59] M. Stadler, \Publicly veri able secret sharing," in *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding* (U. M. Maurer, ed.), vol. 1070 of *Lecture Notes in Computer Science*, pp. 190{199, Springer, 1996. 78

[60] B. Schoenmakers, \A simple publicly veri able secret sharing scheme and its application to electronic," in *CRYPTO*, vol. 1666 of *Lecture Notes in Computer Science*, pp. 148{164, Springer, 1999. 78

[61] M. P. Jhanwar, \A practical (non-interactive) publicly veri able secret sharing scheme," in *ISPEC*, vol. 6672 of *Lecture Notes in Computer Science*, pp. 273{287, Springer, 2011. 78

[62] R. Canetti, \Universally composable security," vol. 67, pp. 28:1{28:94, 2020. 80

[63] Paxos, \moneroutil," 2017. 89

[64] dedis, \Advanced crypto library for the go language." https://github.com/dedis/kyber, 2020. 89

[65] C. S. Inc., \Tru e Suite." 92

[66] D. A. Schultz, B. Liskov, and M. D. Liskov, \MPSS: mobile proactive secret

sharing," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 4, pp. 34:1{34:32, 2010. 96

[67] R. Vassantlal, E. Alchieri, B. Ferreira, and A. Bessani, \COBRA: dynamic proactive secret sharing for con dential BFT services," in *IEEE Symposium on Security and Privacy*, pp. 1335{1353, IEEE, 2022. 96

[68] Y. Yan, Y. Xia, and S. Devadas, \Shanrang: Fully asynchronous proactive secret sharing with dynamic committees," *ePrint Arch.*, 2022. 96

[69] T. Yurek, Z. Xiang, Y. Xia, and A. Miller, \Long live the honey badger: Robust asynchronous DPSS and its applications," *IACR Cryptol. ePrint Arch.*, p. 971, 2022. 96

[70] A. Shamir, \How to share a secret," *Commun. ACM*, 1979. 99

[71] T. Rabin and M. Ben-Or, \Veri able secret sharing and multiparty protocols with honest majority (extended abstract)," in *STOC*, ACM, 1989. 100

[72] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, \Secure distributed key generation for discrete-log based cryptosystems," in *EUROCRYPT*, Lecture Notes in Computer Science, Springer, 1999. 100

[73] X. Qin, S. Pan, A. Mirzaei, Z. Sui, O. Ersoy, A. Sakzad, M. F. Esgin, J. K. Liu, J. Yu, and T. H. Yuen, \Blindhub: Bitcoin-compatible privacy-preserving payment channel hubs supporting variable amounts," *IEEE S&P*, 2023. 121

[74] Churp Team, \Churp." https://github.com/CHURPTeam/CHURP, 2019. 126

[75] \Kryptology: Coinbase's open source cryptography library." https://github.com/coinbase/kryptology/tree/master/pkg/verenc/camshoup. 126

[76] \the gnu multiprecision library (gmp)." https://github.com/Nik-U/pbc. 126

[77] \The pairing-based cryptography library." https://github.com/ncw/gmp. 126

[78] \The high performance remote procedure call (rpc) framework." https://pkg.go.dev/google.golang.org/grpc. 126

[79] \Filecoin." https://filecoin.io/. 127

[80] \Corda." https://www.r3.com/products/corda/. 127

[81] \WAN vs. LAN: What Is the Di erence?." https://www.truecable.com/blogs/cable-academy/wan-vs-lan. 129