Towards Large Scale Distributed Key Generation

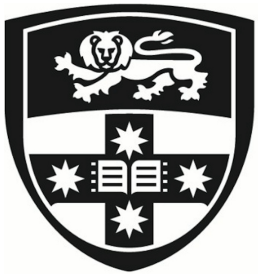
TIANCHENG MAI

Supervisor: Associate Professor Qiang Tang

This thesis is submitted in partial fulfillment of
the requirements for the degree of
Master of Philosophy

School of Computer Science
The University of Sydney
Australia

4 August 2025



THE UNIVERSITY OF SYDNEY

Abstract

Internet-based services always face security threats, and one of the fundamental problems is solving public key generation in a distributed manner. Distributed key generation (DKG) is a cryptographic primitive that produces one public key and shares of a corresponding secret key among a group of distributed parties. As the basis of threshold cryptography, the classical DKG protocols are resurging due to their widespread applications in blockchain, such as cross-chain bridge, MEV protection, censorship resistance in asynchronous consensus, checkpointing into Bitcoin, and more. Those new applications raise a new fundamental challenge of deploying distributed key generation on a very large scale. While efforts have been made to improve DKG communication, practical large-scale deployments are still yet to come due to various issues, including the heavy computation and broadcast overhead in adversarial cases. On the other hand, theoretical challenges arise when we open the blackbox of broadcasting, drop common coin assumption, and aim to optimize the latency, achieve a stronger security and allow DKG to output a more generalized type of secret. This thesis focus on solving scalable DKG on both practical and theoretical landscape.

We first investigate how to build a practical, scalable and adaptively secure DKG protocol to enable all-hands checkpointing in blockchains with weighted validators. Our Any-Trust DKG achieves (quasi-)linear computation and broadcast overhead per-node cost with the help of a common coin, against weak adaptive adversaries. The key to our improvements lies in delegating the most costly operations to an *Any-Trust* group together with a set of techniques for adaptive security. Our Any-Trust DKG leads to a fully practical instantiation of Filecoin's checkpointing mechanism, in which *all* validators of a Proof-of-Stake (PoS) blockchain periodically run DKG and threshold signing to create checkpoints on Bitcoin, to enhance the security of the PoS chain.

Then, on the theoretical side, we purpose Circular Dragon, the first construction of adaptively secure DKG protocol that (i) realizes optimal $n/2$ resilience in the synchronous network, and (ii) attains near-optimal asymptotic complexities, i.e., $\widetilde{O}(n^2)$ communication and $\widetilde{O}(1)$ rounds. In addition, the proposed DKG protocol also produces field-element secrets to support the standard discrete-logarithm based threshold cryptosystem.

Statement of Attribution

The contents of this thesis are based on one published paper and one unpublished manuscript. I was one of the main contributors on both papers. As per convention in theoretical computer science, authors are listed alphabetically.

Chapter 3. Hanwen Feng, **Tiancheng Mai**, Qiang Tang. *Scalable and Adaptively Secure Any-Trust Distributed Key Generation and All-hands Checkpointing*. In Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024.

Chapter 4. Hanwen Feng, Yuan Lu, **Tiancheng Mai**, Qiang Tang. *Circular Dragon: $\widetilde{O}$ptimal Adaptively Secure Distributed Key and Randomness Generation*. Under submission.

My particular contributions to this have been:

- Chapter 3: I designed the experiment, analyzed the experiment results, designed the sub-ID allocation and analyzed its security, and wrote the manuscript.
- Chapter 4: I designed the PCSBB protocol, analyzed the security, and wrote the manuscript.

Name: Tiancheng Mai

Signature: **Date**:

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

Name: Qiang Tang

Signature: **Date**:

Statement of Originality

This is to certify that the content of this thesis is my own work. This thesis has not been submitted for any other degree or purpose.

I certify that the intellectual content of this thesis is the product of my own work, and that all assistance received in preparing this thesis and all sources have been acknowledged.

Name: Tiancheng Mai

Signature: **Date**:

Acknowledgements

This thesis would not have been possible without the help of many people.

I thank my supervisor, Associate Professor Qiang Tang, for his expertise, patience, and guidance throughout my Master's candidature. It has been great working with you and I look forward to our continued research together.

I would like to express my sincere appreciation to Dr Hanwen Feng for his mentorship, valuable insights, and engagement in our fruitful discussions, and extend my thanks to coauthors and collaborators: Dr Zhenliang Lu and Dr Yuan Lu. Thank you for enriching our research discussions. I have learned a lot from you and I hope to work together with you soon.

I would also like to thank the extended research group (or family) for a number of things: sharing snacks and coffee, frequent random chats, organizing holiday events, hiking (for seven hours), giving me free rides, carrying me in badminton matches, finding cheap eats together, celebrating each other's birthday together, and the great working atmosphere: Quanhao Chen, Omniyyah Ibrahim, Ya-Nan Li, Sam Polgar, Tian Qiu, Yuchen Ye, Tianyi Zhang, Xinrui Zhang; and Dr Long Chen. I would like to extend my gratitude to my neighbor research students and academics for creating such a positive atmosphere and research groups for organizing wonderful seminars (with free food) and reading groups: Zhichun Lu, Zijian Luo, Qiping Yang, Yuben Yang, Yuanzhe Zhang, Donglin Zhuang; Dr Liyi Zhou; the Sydney Blockchain Group, and the Sydney Algorithms Group.

No Man is an Island. I am grateful to all my friends for their support and encouragement throughout this journey. In particular, I would like to extend my heartfelt thanks to Yuhan Chen, Jenny Lin, Jinjing Ye, and James Zhao for their invaluable companionship and support.

I would like to thank my family. Thank you, mom and dad, for being there for me and allowing me to pursue my academic goals.

Lastly, this candidature was supported in part by Protocol Labs Research Grants under RFP-012 on Checkpointing Filecoin onto Bitcoin. I express my sincere gratitude for their funding, which was essential to the successful completion of this work.

List of Figures

List of Tables

CONTENTS

CHAPTER 1

Introduction

1.1 Distributed Key Generation

Internet-based services have always been facing security threats, and one of the fundamental problems is solving *trust* in a distributing context. To avoid *single-point failures*, it is too risky to provide key services in just one server, because the system will be down when this single point halts. To circumvent this, a service provider can create multiple replicas on the Internet to improve reliability. However, replication will not prevent failures and attacks, and therefore, the system remains vulnerable when **adversaries** compromise a subset of the replicated server and force them to starts behaving arbitrarily.

Threshold cryptography studies a family of cryptographic protocols that can tolerate adversary so long as there are more **honest** parties than dishonest ones, that is, when trustworthy parties win the majority. One of most critical problems in threshold cryptography is *secret sharing*, where the goal is to secretly share a private message (**secret**) within a group of parties, and make sure: (1) the adversaries cannot ever find out what secret has been exchanged, even if they corrupt many parties within their corruption budget and collude, and (2) the honest majority can always jointly recover the secret.

Distributed key generation (DKG) is a cryptographic primitive that satisfies the secret sharing properties and generates asymmetric keys. A DKG protocol produces one public key and shares of a corresponding private key among a group of distributed parties, without a pre-determined coordinator (**dealer**). At the end of a (n, t)-DKG protocol execution among n parties, each party will hold only one share of the private key, and no party alone can reconstruct the private key, unless strictly more than t parties cooperate, where t is the threshold of the DKG protocol. This property allows the group to tolerate the presence of at most t **malicious** parties (**adversaries**) while **honest** parties still form the majority.

DKG has many applications in cryptography. In public-key cryptography (PKC), DKG is critical for dealer-less threshold public-key encryption and signature schemes. In the threshold encryption scheme,

one will use the public key generated by DKG to encrypt plaintext. Then, the ciphertext is shared among all group members, however no party alone can directly decrypt the ciphertext, but can get a decrypted share using its own share of the private key. Finally, the plaintext can be discovered when t parties combine the decrypted shares where t is the threshold.

In the threshold signature scheme, the goal is to sign a message collectively using the private key of the group. Since individual members hold distributed shares of the signing key, each party will sign the same message via its own share of signing key. Later, the group can combine more than threshold number of signature shares, and reconstruct the wanted complete signature.

It has proven crucial for distributed coin flipping (Cachin et al., 2000; DRand, 2023), Byzantine consensus (Guo et al., 2020; Lu et al., 2020), secure multi-party computation (Cramer et al., 2001). In a more boarder landscape, DKG attracts recent resurge of interests in new blockchain applications. This includes cross-chain bridge (Lee et al., 2023; Cerulli et al., 2023), MEV protection (Malkhi and Szalachowski, 2022; Ruby.Exchange, 2021; Total-blockchain, 2022), censorship resistance in asynchronous consensus (Miller et al., 2016; Guo et al., 2020), checkpointing into Bitcoin (Azouvi and Vukolic, 2022), and more. Due to its emerging real-world deployments (potentially on a large scale) (DRand, 2023; Azouvi and Vukolic, 2022), there has been a resurgence of interests (Tomescu et al., 2020; Gurkan et al., 2021; Feng et al., 2024a; Bacho et al., 2024) in designing more efficient DKG protocols.

1.2 Challenges of Efficient DKG Designs

Let us first briefly introduce the common paradigm for DKG protocols, which subsumes most DKG constructions, including (Pedersen, 1991; Gennaro et al., 2007; Kate et al., 2010; Tomescu et al., 2020; Zhang et al., 2022), to illustrate the astronomical communication and computation costs of existing DKGs in a large scale.

Verifiable Secret Sharing (VSS)-based DKG. The notation of secret sharing is an important topic in cryptographic research. Introduced independently by Shamir (Shamir, 1979) and Blakley (Blakley, 1979) back in 1979.

Within the context of $(n, t+\delta, t)$-secret sharing, a dealer aims to share a confidential message s among a set of n parties. During the sharing process, it has to guarantee that any $t+\delta$ or more parties can get the secret s eventually, but any t or fewer cannot. In the case when $\delta = 1$, a $(n, t+\delta, t)$-secret sharing is aliased with (n, t)-*threshold* secret sharing, since t becomes the strict boundary that decides whether the reconstruction of the secret can be successful.

However, (n, t)-threshold secret sharing has not taken *trust* into consideration, since it assumes a non-faulty dealer, and a *private* communication channel, and it offers no method allowing participants to verify the integrity of the dealer. To allow such protocol to use in a fault-tolerant distributed public network, verifiability is introduced by (Chor et al., 1985), and so does the concept of Verifiable Secret Sharing (VSS). In VSS, participating parties can validate whether a share sent from the dealer indeed is a *valid* piece of the secret.

Many classic DKG constructions rely on Verifiable Secret Sharing (VSS). For example, Pedersen's seminal work (Pedersen, 1991) develops an efficient DKG for DLog-based cryptosystems. In this work, n VSS instances run in parallel, and each party leads an instance of Feldman's VSS (Feldman, 1987) as a dealer and join other $(n-1)$ VSS as a participant.

At here, we briefly explain the idea behind VSS. First, we assume the dealer is non-faulty. The dealer P_i generates a random polynomial f_i of degree $t+1$ to share a secret $s_i = f_i(0)$ when f_i is evaluated at 0. The goal is to deliver the secret share $f_i(j)$, an evaluation of f_i at point j, to the party P_j independently, such that eventually any $t+1$ parties can jointly reconstruct $s_i = f_i(0)$ by interpolating the polynomial. The protocol will not break even if there exists a faulty dealer. The dealer has to broadcast a commitment of the random polynomial at the beginning, and a complaint phrase is mandatory after

the sharing phrase. When the dealer is dishonest and sends invalid shares, participants will accuse such dealer by broadcasting an accusation to the group. The dealer will dispute the complaint by broadcasting evidence against it. However, whenever a complaint is verified or a dispute is faulty, the dealer is then disqualified, and the VSS ends.

The communication cost of Feldman's VSS incurs at (1) commitment broadcast, (2) secret share sending, (3) complaint broadcast, and (4) dispute broadcast. On the other hand, the computation cost depends on the verification of (1) secret share, (2) complaint, and (3) dispute.

In the vanilla Feldman's VSS, the commitment contains t group element, and each corresponds to a coefficient of the polynomial. Therefore, since $t = O(n)$, the commitment size becomes $O(n\lambda)$, where λ is the security parameter reflecting the size of one group element. A significant improvement to minimize the size of the commitment was made by Kate, Zaverucha, and Goldberg (Kate et al., 2010) who proposed the first polynomial commitment (abbreviated as KZG) with a commitment size of only $O(\lambda)$ to commit the whole polynomial. This innovation reduces the commitment size by a factor of $O(n)$.

To construct a DKG, n VSS instances will be executed in parallel. When n instances terminate, each party P_i should hold $\{f_j(i)\}_{1\leq j\leq n}$, and therefore, can compute $f(i) = \sum_{1\leq j\leq n} f_j(i)$, where $f = \sum_{1\leq j\leq n} f_j$ is a polynomial of degree $t+1$. Up to this point, the group has collaboratively shared a joint-secret $f(0) = s = \sum_{i=1}^{n} s_i$ by n evaluation shares, and any share subset of size $t+1$ is capable to reconstruct s by a polynomial interpolation.

With the vanilla Feldman's VSS, the total communication of the DKG becomes $O(n\mathcal{B}(n\lambda) + n^2\lambda)$, where $\mathcal{B}_\eta(\cdot)$ measures the actual communication cost of a broadcast option among η parties and η is omitted when it equals to the network size. The computation cost incurs at verifications of proofs and complaints. Since each part can receive at most $O(n^2)$ complaints and each verification requires $O(n)$ group operation, the computation cost per node is $O(n^3)$. However, due to a recent work by Cascudo and David with their Scrape protocol (Cascudo and David, 2017), there exists a transferrable methodology that bounds the verification computational overhead per node by $O(n^2)$.

The DKG scheme by Kate, Zaverucha, and Goldberg (Kate et al., 2010) (and its recently improved version by Zhang et al. (Zhang et al., 2022), dubbed KZG hereafter) represents the state of the art following this paradigm. When KZG commitment is used to bring down the commitment size in Feldman's VSS, the total communication in the good case is improved to $O(n\mathcal{B}(\lambda) + n^2\lambda)$. The computation cost of a

KZG polynomial commitment generation was originally believed to be $O(n^2)$ group operations, however, a recent work by Zhang et al. (Zhang et al., 2022) shows that the computational overhead can be streamlined to $O(n \log n)$. On the other hand, both communication and computation cost will blow up in the bad case, where each node will receive and verify $O(n^2)$ share in the complaint phrase, and become $O(n\mathcal{B}(n\lambda) + n^2\lambda)$ and $O(n^2 \log^2 n)$ respectively. Indeed, improving the adversarial case performance is the major open problem left by (Tomescu et al., 2020).

Public Verifiable Secret Sharing (PVSS)-based DKG. Fouque and Stern (Fouque and Stern, 2001) offered a solution that sidestepped the necessity for a complaint phase by incorporating publicly verifiable secret sharing (PVSS). Each party could now broadcast all "encrypted" shares and enable the public verification immediately.

In the event that a PVSS transcript consists of $O(n)$ ciphertexts, the communication overhead will naturally be $O(n\mathcal{B}(n\lambda))$ when every participant chooses to broadcast its transcript. Historically, the validation of a PVSS transcript required an overhead of $O(n^2)$, suggesting that the per-node computational overhead in DKG might ascend to $O(n^3)$. However, this obstacle was surmounted by Cascudo and David with their Scrape protocol (Cascudo and David, 2017), which introduced a PVSS methodology that caps the verification duration at $O(n)$, leading the per-node computation overhead to $O(n^2)$. A few recent works focus on improving the concrete performance of PVSS schemes, including the lattice-based PVSS (Gentry et al., 2022), Groth's PVSS (Groth, 2021), and PVSS using class groups (Kate et al., 2023).

Aggregatable Public Verifiable Secret Sharing (APVSS)-based DKG. Aggregatable PVSS schemes (Gurkan et al., 2021) are PVSS schemes whose transcripts can be concisely merged into one. There are a few designs that leverage customized communication protocols rather than simply leveraging Byzantine broadcast protocols (or broadcast channels), enjoying asymptotically better complexity. Notably, Gurkan et al. (Gurkan et al., 2021) leveraged an aggregatable PVSS combined with gossip protocols to craft a publicly verifiable DKG. Their communication overhead is streamlined to $n\mathcal{B}(\lambda) + \log n \cdot \mathcal{B}(n\lambda)$ as opposed to $n\mathcal{B}(n\lambda)$, with their per-node communication overhead being $O(n \log^2 n)$. It's pertinent to note, however, that their model can only accommodate $O(\log n)$ Byzantine nodes.

Recently, Feng et al. (Feng et al., 2024a) and Bacho et al. (Bacho et al., 2023b) leverage specially designed communication protocols together with aggregatable PVSS schemes and present DKG schemes with sub-quadratic per-node computation/communication cost while enjoying optimal resilience. The

former presents two statically secure DKG while the group-element one can obtain adaptively security by using an adaptively secure PVSS scheme but the field-element one cannot. Both DKGs terminate in $O(n)$ rounds, while the group-element DKG has a total communication cost of $O(n\mathcal{B}_{\sqrt{n}}(n\lambda) + \sqrt{n}\mathcal{B}_n(n\lambda)) = O(n^{2.5}\lambda)$ and a per-node computation overhead $O(n^{1.5}\lambda)$, whereas the field-element one has a total communication cost of $O(n\mathcal{B}_{\sqrt{n}}(n\lambda) + \sqrt{n}\mathcal{B}_n(n\lambda\kappa)) = O(n^{2.5}\lambda\kappa)$ and a per-node computation overhead $O(n^{1.5}\lambda\kappa)$. The latter obtains, GRand, an adaptively secure DKG that has a total communication cost of $O(\lambda n^2 \log n)$ bits, a per-node computation overhead $O(n \log n)$ and terminates in $O(n)$ rounds.

Note that existing aggregatable PVSS schemes all produce secrets in an Elliptic curve group, thus incompatible with many threshold cryptographic protocols.

1.2.1 Major Issues

Communication and computation efficiency. Due to the linear broadcast per-node overhead and worst-case complaint phase, we can readily anticipate that both communication and computation costs would skyrocket as the scale increases for classic DKGs, as seen in scenarios like Filecoin validators. For instance, with 2^{12} participants in the DKG protocol, the entire network would need to transmit *tens of terabytes* of data, while each node would have to allocate *multiple hours* to verify shares (in response to complaints) to produce just one public key![1] On the other hand, recent DKGs that relies on APVSS may be more performant, however they face issues like not being able to produce field-element secrets or cannot establish adaptive security.

Supporting field-element secret and establishing adaptive security. We remark that publicly verifiable secret sharing (PVSS) can eliminate the complaint phase in DKG (Cascudo and David, 2017; Fouque and Stern, 2001; Gentry et al., 2021b; Bacho and Loss, 2023a), however, existing PVSS-based DKGs are either even more costly than KZG (Fouque and Stern, 2001; Groth, 2021) or only generate group-element secrets, while mainstream threshold cryptographic protocols like threshold Schnorr signatures use field-element secrets. In addition, no existing PVSS schemes with field-element secrets are provably secure against adaptive attackers (Bacho and Loss, 2023a), while adaptive security is desired in practice. We note that there are two recent PVSS-based DKG works (Bacho et al., 2023b; Feng et al.,

[1] For detailed numerical estimates and comparisons, we refer to Section 3.9.1.

2024a) have further pushed down the asymptotic complexity of DKG, but they are either only with group-element secrets or statically secure.

Round complexity. When a one-round broadcast channel is not assumed, classic DKG protocols inherit $O(n)$ round complexity from deterministic consensus subroutines where every node in the whole network participates. A global common coin may be further assumed to support randomized consensus protocols and reduce the round complexity to constant round, however this may cause a circularity issue since common coins are often instantiated by DKG.

Weighted threshold signatures. Besides those high costs, one extra challenge may make things even worse: in PoS chains, validators usually have different *weights* (proportional to the number of held stakes), while a threshold of weights is assumed to belong to honest validators. Simply viewing a validator as a participant in DKG can be leveraged by an adversary to amplify its power: the adversary can choose to corrupt many validators with small weights and eventually control the majority of DKG participants within its budget. A naive approach is to allocate different numbers of sub-IDs proportional to their weights. Each sub-ID is then treated as an independent participant. While this approach addresses the security concern, it can lead to an enormous number of sub-IDs. For example, we would need to allocate 674 trillion sub-IDs to 3700 Filecoin validators[2].

[2] `https://filfox.info/en/ranks/power`. It has a total mining power of around 25 EB, while the power unit is 32KB, so 674 trillion sub-IDs are needed.

1.3 Contributions

We summarize the main contributions in this thesis in this section. Chapters 3 contributes to implementing a practical scalable and adaptively secure DKG protocol with field-element secrets that enables all-hands participation in blockchains with weighted validators. Chapters 4 contributes to implementing an adaptively secure DKG with field-element secrets, $\widetilde{O}(n^2)$ communication cost, $\widetilde{O}(1)$ rounds, and optimal resilience.

Chapter 3: Scalable and Adaptively Secure Any-Trust Distributed Key Generation and All-hands Checkpointing

The classical distributed key generation protocols (DKG) are resurging due to their widespread applications in blockchain. While efforts have been made to improve DKG communication, practical large-scale deployments are still yet to come due to various challenges, including the heavy computation and communication (particularly broadcast) overhead in their adversarial cases. In this chapter, we propose a practical DKG for DLog-based cryptosystems, which achieves (quasi-)linear computation and communication per-node cost with the help of a common coin, even in the face of the maximal amount of Byzantine nodes. Moreover, our protocol is secure against adaptive adversaries, which can corrupt less than half of all nodes. The key to our improvements lies in delegating the most costly operations to an *Any-Trust* group together with a set of techniques for adaptive security. This group is randomly sampled and consists of a small number of individuals. The population only trusts that at least one member in the group is honest, without knowing which one. Moreover, we present a generic transformer that enables us to efficiently deploy a conventional distributed protocol like our DKG, even when the participants have different *weights*. Additionally, we introduce an extended broadcast channel based on a blockchain and data dispersal network (such as IPFS), enabling reliable broadcasting of arbitrary-size messages at the cost of constant-size blockchain storage.

Our DKG leads to a fully practical instantiation of Filecoin's checkpointing mechanism, in which *all* validators of a Proof-of-Stake (PoS) blockchain periodically run DKG and threshold signing to create checkpoints on Bitcoin, to enhance the security of the PoS chain. In comparison with the recent checkpointing approach of Babylon (Oakland, 2023), ours enjoys a significantly smaller cost of Bitcoin transaction fees. For 2^{12} validators, our cost is merely 0.4% of that incurred by Babylon's approach.

Chapter 4: Circular Dragon: $\widetilde{\text{O}}$ptimal Adaptively Secure Distributed Key and Randomness Generation

In this chapter, we give the first construction of adaptively secure distributed key generation (DKG) protocol that (i) realizes optimal $n/2$ resilience in the synchronous network, and (ii) attains near-optimal asymptotic complexities, i.e., $\widetilde{O}(n^2)$ communication and $\widetilde{O}(1)$ rounds. In addition, the proposed DKG protocol also produces field-element secrets to support the standard discrete-logarithm based threshold cryptosystem.

At the core of our construction, it is a new design methodology, which we call "Circular Dragon", that proceeds in two main steps: (1) constructing a nearly-optimal field-element DKG assuming common coins, which in turn can be derived from a group-element DKG; (2) constructing a nearly-optimal group-element DKG, via a fine-grained Dragon method (Feng et al., 2024a) that was recently developed for sub-cubic communication DKG protocols but with linear rounds.

Along the way, we formulate and efficiently construct a few new primitives (such as parallel consortium-sender Byzantine broadcast and subset Byzantine broadcast), and construct the first distributed *common coin* protocol with nearly optimal round and communication complexities, which could be of independent interest.

Chapter 5: Conclusion and Future Works

In this chapter, we summarize our contributions and point out a few interesting future directions and open problems. This includes designing a scalable and adaptively secure DKG in asynchrony and whether it is possible to remove the logarithm term in the DKG complexity to achieve a tighter quadratic result. Moreover, it is unknown how we should adapt DKG and threshold protocols to other cryptographic systems and whether we can find broader applications beyond DKG for our grouping divide-and-conquer techniques.

CHAPTER 2

Preliminaries

2.1 Notations

We let λ represent the bit length of cryptographic security parameter. The notation $[i, n]$ represents the set $\{i, i+1, \cdots, n\}$, where i and n are integers with $i < n$. We might abbreviate $[1, n]$ simply as $[n]$. For a set $\{x_1, x_2, \ldots, x_n\}$ and a sequence $(x_1, x_2, \ldots, x_n)$, we let them be denoted by $\{x_i\}_{i\in[n]}$ and $(x_i)_{i\in[n]}$ for short, respectively. A function $f(n)$ is deemed negligible in n, denoted by $f(n) \leq \mathrm{negl}(n)$, if for every positive integer c, there exists an n_0 such that for all $n > n_0$, $f(n) < n^{-c}$. Conversely, a non-negligible function is denoted as $f(n) > \mathrm{negl}(n)$. For a set $\mathbb{X}$, the notation $x \leftarrow\!\!\$\ \mathbb{X}$ signifies sampling x uniformly from $\mathbb{X}$. Given a distribution X, $x \leftarrow X$ denotes sampling x from X. For a probabilistic algorithm A, $A(x_1, x_2, \cdots; r)$ represents the result of running A with inputs $x_1, x_2, \cdots$ and random coins r. We use $y \leftarrow A(x_1, x_2, \cdots)$ to represent choosing r randomly and computing $A(x_1, x_2, \cdots; r)$ to get the output y. An execution of the protocol Π with an identifier sid, involving participants $\mathcal{P}$ where each P_i inputs v_i, is represented by $\Pi[\mathsf{sid}]\langle\{P_i(v_i)\}\rangle$.

2.2 Cryptographic Primitives

Distributed key generation. In this thesis, we mainly focus on DKG for the standard key structure of discrete-logarithm cryptosystems, where the key pair (pk, sk) is generated as follows: first uniformly sample sk from the scalar field $\mathbb{F}$, and then compute $pk = g^{sk} \in \mathbb{G}$, where $g \in \mathbb{G}$ is a generator of the group $\mathbb{G}$.

SYNTAX. An (n, t)-DKG protocol, denoted as Π_{DKG}, involves n parties $\mathcal{P} = (P_1, \ldots, P_n)$. After its execution, each P_i outputs a public key $pk \in \mathbb{G}$, a list of public key shares $(pk_i)_{i\in[n]} \in \mathbb{G}^n$, and a secret key share $sk_i \in \mathbb{F}$. The protocol also accompanies an initial phase and a reconstruction algorithm:

- $\mathsf{Init}(1^\lambda, n)$. It generates the common reference string (CRS)[1] $\mathtt{crs}$ and establishes the PKI for all participants in $\mathcal{P}$.
- $\mathsf{Rec}((i, sk_i)_{i\in\mathbb{I}})$. Given a set of $t+1$ secret key shares as input, the algorithm outputs the secret key $sk \in \mathbb{F}$ corresponding to $pk \in \mathbb{G}$, such that $pk = g^{sk}$.

Regarding security, we consider robustness, oracle-aided algebraic simulatability, full secrecy, and key-expressability, which are defined as follows.

ROBUSTNESS. It ensures all participants at the end of the protocol obtain the same public key and correct shares.

DEFINITION 1 (Robustness). Π_{DKG} *is* **robust** *if, even when up to t parties are (adaptively) compromised, each remaining honest node P_i would output the same $(pk, (pk_i)_{i\in[n]}) \in \mathbb{G}^{n+1}$, and its $sk_i \in \mathbb{F}$ satisfies $pk_i = g^{sk_i}$. Additionally, for any two sets, $\mathbb{I}_1$ and $\mathbb{I}_2$, with $t+1$ honest participants each, a unique secret key sk can be reconstructed from their secret shares, i.e.*

$$\Pr\left[\begin{array}{r}\mathsf{Rec}(pk, (pk_i)_{i\in[n]}, \{sk_i\}_{i\in\mathbb{I}_1}) = \mathsf{Rec}(pk, (pk_i)_{i\in[n]}, \{sk_i\}_{i\in\mathbb{I}_2}) \\ \wedge\ pk = g^{\mathsf{Rec}(\{(i,sk_i)\}_{i\in\mathbb{I}_1})}\end{array}\right] = 1.$$

FULL SECRECY. It captures that during the execution of Π_{DKG}, the adversary learns nothing about the secret key beyond the final public key and cannot bias the key distribution.

DEFINITION 2 (Full Secrecy). Π_{DKG} *satisfies* **full secrecy**, *if for any PPT adversary $\mathcal{A}$ which corrupts up to t nodes, there is a PPT simulator $\mathsf{Sim}^{\mathcal{A}}$, which on input a uniformly sampled public key $pk \leftarrow\!\!\$\ \mathbb{G}$,*

[1] Note that our Circular Dragon DKG protocol only requires a uniform random string (URS), though the general syntax of DKG might require CRS setup.

and outputs a simulated view $\mathsf{sview}_{\mathcal{A}}$*, which is computationally indistinguishable with the adversary* $\mathcal{A}$*'s view* $\mathsf{view}_{\mathcal{A}}$ *in a real execution of* Π_{DKG} *that ends with* pk *as its public key output. Here* $\mathsf{view}_{\mathcal{A}}$ *includes all public messages and the internal states visible to* $\mathcal{A}$*.*

ORACLE-AIDED ALGEBRAIC SIMULATABILITY. Most DKG protocols (Pedersen, 1991; Gennaro et al., 2007; Katz, 2024) cannot be proven with adaptive full secrecy. To capture the adaptive security of many classical DKG protocols, Bacho and Loss (Bacho and Loss, 2022) introduced *oracle-aided algebraic simulatability*, where the simulator is expected to simulate the execution view with the help of limited algebraic accesses to a DLog oracle. As shown in (Bacho and Loss, 2022), a DKG with this property is at least sufficient for the adaptive threshold BLS.

DEFINITION 3 (Oracle-aided algebraic simulatability). Π_{DKG} *realizes* k**-oracle-aided algebraic simulatability** *if for every PPT adversary* $\mathcal{A}$ *adaptively corrupts at most* t *parties, there exists a PPT simulator* $\mathsf{Sim}^{\mathcal{A}}$ *which on input* $\zeta = (g^{z_1}, \ldots, g^{z_k}) \in \mathbb{G}^k$*, can query the DLog oracle* $\mathsf{DL}_g(\cdot)$ *for at most* $k-1$ *times, and simulate an execution of* Π_{DKG} *for* $\mathcal{A}$*. In particular,*

- ***Algebraic oracle queries:*** *When* $\mathsf{Sim}^{\mathcal{A}}$ *queries* $\mathsf{DL}_g(\cdot)$ *with a group element* g'*, it must provide the algebraic expression of* g' *in the term of* $(g, g^{z_1}, \ldots, g^{z_k})$*, i.e., a vector* $(\hat{a}, a_1, \ldots, a_k) \in \mathbb{F}^{k+1}$ *such that* $g' = g^{\hat{a}} \prod_{j\in[k]} (g^{z_j})^{a_j}$*. The oracle will return* $a \in \mathbb{F}$ *s.t.* $g' = g^a$*.*
- ***Indistinguishable simulation:*** *Denote by* $\mathsf{view}_{\mathcal{A},y,\Pi}$ *the view of* $\mathcal{A}$ *in an execution of* Π *conditioned on all honest parties outputting* $pk = y$*. Denote by* $\mathsf{view}_{\mathcal{A},\zeta,y,\mathsf{Sim}}$ *the view of* $\mathcal{A}$ *when interacting with* Sim *on input* ζ*, conditioned on* Sim *outputting* $pk = y$*. Then, for all* y *and all* ζ*,* $\mathsf{view}_{\mathcal{A},y,\Pi}$ *and* $\mathsf{view}_{\mathcal{A},y,\Pi}$ *are computationally indistinguishable.*
- ***Invertible simulatability matrix:*** *Let* g_i *denote the* i*-th query by* Sim *to* $\mathsf{DL}_g(\cdot)$*. Let* $(\hat{a}_i, a_{i,1}, \ldots, a_{i,k})$ *be the corresponding algebraic coefficients of* g_i*, i.e.,* $g_i = g^{\hat{a}_i} \prod_{j=1}^{k} (g^{z_j})^{a_{i,j}}$ *and set* $(\hat{a}, a_{0,1}, \ldots, a_{0,k})$ *as the algebraic coefficients of* pk*. Then, the following matrix over* $\mathbb{F}$ *is invertible*

$$L := \begin{pmatrix} a_{0,1} & a_{0,2} \cdots & a_{0,k} \\ a_{1,1} & a_{1,2} \cdots & a_{1,k} \\ \vdots & \vdots & \vdots \\ a_{k-1,1} & a_{k-1,2} \cdots & a_{k-1,k} \end{pmatrix}.$$

Moreover, if Π_{DKG} *realizes* k**-oracle-aided algebraic simulatability** *for every PPT adversary* $\mathcal{A}$ *runs in time at most* $T_{\mathcal{A}}$ *and the underlying PPT simulator* $\mathsf{Sim}^{\mathcal{A}}$ *runs in time at most* T_{sim}*, we say that* Π_{DKG} *realizes* $(t, k, T_{\mathcal{A}}, T_{\mathsf{sim}})$**-oracle-aided algebraic simulatability**.

KEY-EXPRESSABILITY. Additionally, we consider "key-expressability", as introduced in (Gurkan et al., 2021), against static attackers. This property is suitable for instantiating the key generation of re-keyable primitives like BLS and ElGamal. It also works with Schnorr signature as recently shown in (Gurkan et al., 2021; Shoup, 2023). Definition is as follows:

DEFINITION 4 (Key-expressability (Gurkan et al., 2021)). *A DKG protocol* Π_{DKG} *is* **key-expressable** *if for every static PPT adversary* $\mathcal{A}$ *that corrupts up to* t *nodes, there exists a PPT simulator* Sim*, such that on input of a uniformly random element* $pk' \in \mathbb{G}$*, produces* $\alpha\mathbb{Z}_p$*,* $sk_1 \in \mathbb{Z}_p$*,* $pk_1 = g^{sk_1} \in \mathbb{G}$*, and a view which is indistinguishable from* $\mathcal{A}$*'s view from a run of the DKG protocol that ends with* $pk = pk'^{\alpha} \cdot pk_1$*.*

Aggregatable PVSS. An (n, t)-secret sharing (SS) scheme allows a dealer to distribute a secret s among n participants. Any group of $t + 1$ honest parties can reconstruct s, yet any smaller group (up to t parties) remains oblivious to s. Whereas SS assumes a trustworthy dealer, verifiable secret sharing (VSS) addresses the possibility of a malicious dealer by letting a receiver validate the consistency of its share with a public commitment. Publicly-verifiable secret sharing (PVSS) takes VSS a step further: all encrypted shares, together with a verifiable proof for the consistency of these encrypted shares, are placed on a public channel for universal validation. An *aggregatable* PVSS further can compress multiple PVSS transcripts into a single publicly verifiable transcript.

SYNTAX. We describe aggregatable PVSS with the following eight algorithms/phases. For simplicity, we assume that the "native" transcripts (produced by Deal) and the aggregated transcripts are in the same form (though they differ by the size of their CID set), and thus all algorithms and properties apply to both types of transcripts. The syntax and definitions for a (non-aggregatable) PVSS can be obtained by removing the algorithm Agg.

- $\mathsf{Init}(1^\lambda, n)$: In the initial phase, a CRS $\mathtt{crs}$ is generated, and the encryption/decryption keys $\{(ek_i, dk_i)\}_{i\in[n]}$ for all participants are set up. $\mathtt{crs}$ is an implicit input for all other algorithms.
- $\mathsf{Deal}((ek_i)_{i\in[n]}, t, \mathtt{cid}) \rightarrow (\mathsf{Trans}, sk)$: On inputs the public key list, the threshold t, and the CID of the creator, it produces a secret $sk \in \mathcal{SK}$ and a transcript Trans, consisting of a

commitment com to the secret sk, ciphertexts $(c_i)_{i\in[n]}$, a proof π of validity, and the CID set $\{\texttt{cid}\}$.

- $\mathsf{Agg}(\{(\mathsf{Trans}_i)\}_{i\in[m]}, (ek_i)_{i\in[n]}) \rightarrow \mathsf{Trans}$: It outputs an aggregated transcript Trans whose CID set is $\{\texttt{cid}_i\}_{i\in[m]}$, where $\texttt{cid}_i$ is from Trans_i.
- $\mathsf{PubVrfy}((ek_i)_{i\in[n]}, \mathsf{Trans}) \rightarrow b$: It checks if Trans is valid.
- $\mathsf{getCID}(\mathsf{Trans}) \rightarrow \{\texttt{cid}_i\}_{i\in[m]}$: It returns the CID set.
- $\mathsf{PubDriv}(\mathsf{Trans}) \rightarrow (pk, (pk_i)_{i\in[n]})$: It derives the public key (shares).
- $\mathsf{Dec}(ek_i, dk_i, \mathsf{Trans}) \rightarrow sk_i$: One can decrypt the ciphertext c_i in Trans and obtain the secret share sk_i.
- $\mathsf{Rec}(\{(i, sk_i)\}_{i\in\mathbb{I}}) \rightarrow sk$: It first determines coefficients $\{\alpha_i\}_{i\in\mathbb{I}}$, where $\alpha_i \in \mathbb{N}$ based on $\mathbb{I}$ and reconstructs the committed secret sk as $\bigoplus_{i\in\mathbb{I}} \alpha_i sk_i$ from any subset $\mathbb{I} \subset [n]$ and $|\mathbb{I}| = t + 1$.

Verifiable Random Function. A verifiable random function (VRF) is a pseudorandom function whose outputs can be publicly verified using the evaluator's public key. Throughout this thesis, we take the DDH-based VRF scheme from (Goldberg et al., 2016) as our instantiation. A VRF scheme consists of three algorithms: (1) $\mathsf{KeyGen}(1^\lambda)$ generates a verification key vk and the secret evaluation key sk; (2) $\mathsf{Eval}(vk, sk, x)$ evaluates the function with sk on the input x, and outputs y along with a proof π. (3) $\mathsf{Verify}(vk, x, y, \pi)$ verifies if y is the honest evaluation output on x with the secret key of vk.

A secure VRF satisfies (1)*Pseudorandomness*: the function values are pseudorandom, even given the public key and the proofs; (2) *Completeness*: it always holds that $\mathsf{Verify}(vk, x, \mathsf{Eval}(vk, sk, x)) = 1$; and (3) *Uniqueness*: it is infeasible to generate a public key vk, an input x, and two different (y_1, π_1) and (y_2, π_2), such that

$$\mathsf{Verify}(vk, x, y_1, \pi_1) = \mathsf{Verify}(vk, x, y_2, \pi_2) = 1.$$

(4) *Unpredictability under malicious key generation*: if the input x has enough entropy (i.e., cannot be predicted), then the correct output y is indistinguishable from a uniformly random value, no matter how the VRF keys are generated. Formal definitions can be found in (Micali et al., 1999; Badertscher et al., 2018).

VRF-based sortition. We introduce the standard VRF-based sortition scheme below and will use it as a black box in our Any-Trust DKG protocol. (1) $\mathsf{Setup}(1^\lambda)$. Each user generates their VRF key pair (vk, sk) and publishes vk. A public randomness rand is sampled independent of the key generation. (2) $\mathsf{Sortition}(vk, sk, \mathsf{rand}, \mathsf{event}, \mathsf{ratio})$. A user with (vk, sk) evaluates the VRF on the input

of $(\mathsf{rand}|\mathsf{event})$ and obtains y and a proof π. It checks if $\frac{y}{\mathsf{max}} \leq \mathsf{ratio}$. If failed, abort. Otherwise, return (y, π) as the credential of being selected. (3) $\mathsf{Vrfy}(vk, \mathsf{rand}, \mathsf{ratio}, \mathsf{event}, \mathsf{credential})$. It verifies the credential by validating the VRF output y and checking if $\frac{y}{\mathsf{max}} \leq \mathsf{ratio}$.

In above, ratio denotes the ratio of the expected committee size to the whole group size, and the expected committee size is determined by the expected ratio of honest nodes to the committee.

Security of VRF-based sortition. It is easy to argue when the underlying VRF satisfies the security properties defined above, and rand is sampled independently of the key generation, the VRF-based sortition outcome is computationally indistinguishable from the outcome of a process where each user is elected with an independent probability of ratio. Our further analysis is based on this fact.

Forward-secure digital signature. A forward-secure signature scheme $\mathsf{FS}.\Sigma$ consists of four algorithms: (1) $\mathsf{Gen}(1^\lambda) \to (\mathsf{FS}.vk, \mathsf{FS}.sk[1])$ generates a verification key and the initial signing key; (2)$\mathsf{Update}(\mathsf{FS}.sk[i]) \to \mathsf{FS}.sk[i+1]$ updates the signing key at round i to the signing key at round $i+1$; (3) $\mathsf{Sign}(\mathsf{FS}.sk[i], m) \to \sigma$ generates a signature σ for the message m; (4) $\mathsf{Vrfy}(\mathsf{FS}.vk, i, \sigma, m) \to b$ determines if σ is a valid signature for m created by the signing key at round i.

A forward-secure signature scheme guarantees the unforgeability of signatures at rounds $i < i^*$, even when the adversary has access to signing oracles at any round and corrupts the signing key at the i^*-th round. The formal security definitions and secure instantiations are available in (Itkis and Reyzin, 2001).

Multi-recipient encryption. We use the multi-recipient hybrid ElGamal encryption. Let g be a generator of $\mathbb{G}$, and let Hash be a hash function whose output space is the message space (which we assume is binary encoded and $\oplus$ is the XOR operation). The encryption scheme can be described as follows: (1) $\mathsf{Gen}(1^\lambda)$ outputs $(ek = g^x, dk = x)$, where $x \leftarrow\!\$\ \mathbb{Z}_p$. (2) $\mathsf{MREnc}(ek_1, \ldots, ek_n, m_1, \ldots, m_n)$ outputs the ciphertexts $(c_0, c_1, \ldots, c_n)$, where $c_0 = g^r$ for some uniformly sampled $r \in \mathbb{Z}_p$, $c_i = \mathsf{Hash}(ek_i^r) \oplus m_i$ for $i \in [n]$. (3) $\mathsf{Dec}(ek_i, dk_i, c_0, c_i)$ outputs $m = \mathsf{Hash}(c_0^{dk_i}) \oplus c_i$.

We use the above algorithms in our Any-Trust DKG construction, but we directly reduce Any-Trust DKG to the underlying DDH assumption without going through the security abstraction of the encryption scheme. This is because the security properties we need from the encryption are non-standard (as we discussed in the technique overview), and we would like to avoid further distractions.

Signature of Knowledge. A signature of knowledge (SoK) scheme is defined w.r.t. an NP relation R. It can be either in the common reference string model or in the random oracle model. We focus on the random-oracle-model instantiations and thus omit the algorithm for generating the common reference string. A user with the witness x of a public statement y such that $(y, x) \in R$ can sign any message m via the signer algorithm $\mathsf{SoK.Sign}(y, x, m) \to \sigma$. Later, another user can verify if m was signed by someone with the knowledge of the witness w.r.t. y via the verifier algorithm.

A secure SoK scheme satisfies the following properties: (1) *Simulatability*: there is an efficient simulator algorithm that can produce a valid signature under any statement y without using the witness x, and the produced signatures are indistinguishable from honestly generated signatures. (2) *Extractability*: There is an efficient extractor algorithm that can extract the witness x from a valid signature produced by the adversary under a statement y, even when the adversary has seen some simulated signatures. In the random oracle model, both the simulator and extractor are allowed to program the random oracle. Formal definitions are available in (Chase and Lysyanskaya, 2006).

In this thesis, we use an SoK in the random oracle model for the DLog relation, *i.e.*, for $y \in \mathbb{G}$ and $x \in \mathbb{Z}_p$, we have $(y, x) \in R$ iff $y = g^x$. Note that such an SoK is well-studied and can be instantiated with Schnorr signature scheme.

NIZK. A non-interactive zero-knowledge (NIZK) proof system Π, for an NP language L, enables the prover, who holds a witness of an instance $x \in L$, to convince the verifier that $x \in L$ via a single proof. Typically, it can be described by the following a triple of probabilistic polynomial-time (PPT) algorithms. We remark that in the random oracle model, the setup algorithm is not necessary.

- $\sigma \leftarrow \mathsf{Setup}(1^\lambda)$. The setup algorithm outputs a CRS σ.
- $\pi \leftarrow \mathsf{Prove}(\sigma, x, w)$. The algorithm takes as inputs the CRS σ, an instance $x \in L$ with its witness $w \in R_L(x)$, and outputs a string π called a proof.
- $b \leftarrow \mathsf{Verify}(\sigma, x, \pi)$. The verifier algorithm takes as inputs σ, an instance x and a proof π, and outputs either 1 accepting it or 0 rejecting it.

We consider an NIZK scheme satisfying *completeness*, *zero-knowledge*, and *simulation soundness* defined as follows.

(1) **Completeness:** For all security parameters $\lambda \in \mathbb{N}$ and for all $x \in L_\lambda$ and $w \in R_L(x)$,

$$\Pr[\sigma \leftarrow \mathsf{Setup}(1^\lambda); \pi \leftarrow \mathsf{Prove}(\sigma, x, w) : \mathsf{Verify}(\sigma, x, \pi) = 1] = 1.$$

(2) **Zero-knowledge:** There is a PPT simulator $(\mathsf{SimSetup}, \mathsf{SimProve})$, *s.t.* for every PPT adversary $\mathcal{A}$, we have

$$|\Pr[\sigma \leftarrow \mathsf{Setup}(1^\lambda) : 1 \leftarrow \mathcal{A}^{\mathcal{O}_1(\sigma,\cdot,\cdot)}(\sigma)] -$$
$$\Pr[(\sigma, \tau \leftarrow \mathsf{SimSetup}(1^\lambda) : 1 \leftarrow \mathcal{A}^{\mathcal{O}_2(\sigma,\tau,\cdot,\cdot)}(\sigma)]| \leq \mathsf{negl}(\lambda).$$

Both the oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ take as input a pair $(x, w) \in R_L(x)$. While $\mathcal{O}_1$ returns $\pi \leftarrow \mathsf{Prove}(\sigma, x, w)$, $\mathcal{O}_2$ returns $\pi \leftarrow \mathsf{SimProve}(\sigma, \tau, x)$.

(3) **Simulation soundness:** For any PPT adversary $\mathcal{A}$, it holds that

$$\Pr\left[\begin{array}{l}(\sigma, \tau) \leftarrow \mathsf{SimSetup}(1^\lambda); (x^*, \pi^*) \leftarrow \mathcal{A}^{\mathcal{O}_2(\sigma,\tau,\cdot)}(\sigma) : \\ \qquad\qquad \mathsf{Verify}(\sigma, x^*, \pi^*) = 1 \wedge x^* \notin L\end{array}\right] \leq \mathsf{negl}(\lambda).$$

In Chapter 3, the proof of decryption used in Any-Trust DKG is a NIZK proof system for the decryption correctness. We use a NIZK for proof of decryption correctness w.r.t. the encryption scheme. It consists of a prover algorithm, $\mathsf{PKE.Prove}$, which on inputs $(c_0, c_i, ek_i, dk_i, m)$ produces a proof Γ, and a verifier algorithm $\mathsf{PKE.Vrfy}(c_0, c_i, ek_i, m, \Gamma)$ which checks whether m is the correct decryption from (c_0, c_i). Particularly, $\Gamma = (m, c_0^{dk_i}, \pi)$. Here π demonstrates the discrete logarithm of $c_0^{dk_i}$ w.r.t c_0 is equal to that of ek_i w.r.t. g, which is commonly known as DLEQ proof (equality of discrete logarithms) (Chaum and Pedersen, 1992).

Scrape's polynomial commitment. We use the polynomial commitment scheme from Scrape (Cascudo and David, 2017). Particularly, let g be the generator of $\mathbb{G}$ of prime order p, let f be a t-degree polynomial over $\mathbb{Z}_p$, and let $n > t$ be an integer. Then, the commitment to the polynomial f is

$$(\mathsf{cm}_0 = g^{f(0)}, \mathsf{cm}_1 = g^{f(1)}, \ldots, \mathsf{cm}_n = g^{f(n)}).$$

One can check whether these group elements commit to a t-degree polynomial by performing the following steps: (1) Sample an $(n-t)$-degree polynomial $q(X) \in \mathbb{Z}_p[x]$, and compute the dual code: $\mathsf{cm}_\tau^\perp = \frac{q(\tau)}{\prod_{j=0, j\neq\tau}^{n}(\tau - j)}, \forall \tau \in [0, n]$. (2) Check whether $\prod_{\tau=0}^{n}(\mathsf{cm}_\tau)^{\mathsf{cm}_\tau^\perp} = \mathbf{1}_\mathbb{G}$, where $\mathbf{1}_\mathbb{G}$ is the identity element of $\mathbb{G}$.

It is worth noting that the first step can be reused to check different commitments. The effectiveness of the checking process is determined by the following lemma.

LEMMA 1 ((Cascudo and David, 2017)). *For any* $(\mathsf{cm}_0, \mathsf{cm}_1, \ldots, \mathsf{cm}_n) \in \mathbb{G}^{n+1}$*, if*

$$\prod_{\tau=0}^{n} (\mathsf{cm}_\tau)^{\mathsf{cm}_\tau^\perp} = \mathbf{1}_{\mathbb{G}},$$

then with an overwhelming probability there exists a t*-degree polynomial* f*, such that* $cm_i = g^{f(i)}$ *for* $i \in [0, n]$.

After that, a share $f(i)$ can be validated by checking whether $g^{f(i)}$ equals cm_i.

Multiverse threshold signature with transparent setup. Threshold signature scheme allows a group of nodes to jointly produce a signature for a message which is valid under the public key of the group. Normally, a threshold signature scheme needs some sort of setup for every group, and each node needs to maintain distinct secret keys for all groups it is involved. A multiverse threshold signature (MTS) scheme (Garg et al., 2024; Baird et al., 2023; Das et al., 2023a), instead, allows participants to jointly sign messages on the behalf of different groups (called *universes* in the context), while each node only needs to keep one secret key. In an MTS scheme with a transparent setup (Attema et al., 2021), there is only a global PKI setup where all participants generate their own key pairs and publish their public keys.

Formally, we describe an MTS scheme by the following algorithms.

- $pp \leftarrow \mathsf{Setup}(1^\lambda)$. It generates the global public parameter pp which is an (implicit) input to all the subsequent algorithms. The setup is considered transparent when pp can be derived from uniformly distributed strings.
- $(vk_i, sk_i) \leftarrow \mathsf{KeyGen}(pp)$. Each user P_i runs this algorithm to generate its key pair (vk_i, sk_i).
- $(\mathsf{CK}_U, \mathsf{VK}_U) \leftarrow \mathsf{MTS.UniverseSetup}(\{vk_i\}_{i \in U}, k)$: It is a *deterministic* algorithm which computes a combine key CK_U and a verification key VK_U, according to the public keys of users in U, and the threshold k.
- $\sigma_i \leftarrow \mathsf{PartSign}(sk_i, \mathtt{msg})$. P_i can sign a message $\mathtt{msg}$ with its signing secret key sk_i and produce a partial signature σ_i.
- $\mathsf{true}(1)/\mathsf{false}(0) \leftarrow \mathsf{PartVrfy}(vk_i, \sigma_i, \mathtt{msg})$: This algorithm on inputs a message msg, a partial signature σ, and a public key vk_i verifies the partial signature σ_i.

- $\sigma \leftarrow \mathsf{Combine}(\mathsf{CK}_U, \{(i, \sigma_i)\}_{i \in S \subseteq U}, \mathtt{msg})$: Given the combine key CK_U, and a set of signatures $\{(i, \sigma_i)\}_{i \in S \subseteq U}$ for a message $\mathtt{msg}$, the algorithm outputs a succinct signature σ.
- $\mathsf{true}(1)/\mathsf{false}(0) \leftarrow \mathsf{Vrfy}(\mathsf{VK}_U, \sigma, \mathtt{msg})$: This algorithm verifies the signature σ for $\mathtt{msg}$ under the universe key VK_U. It returns 1, indicating that at least k nodes within in VK_U have signed the message; Otherwise, it returns 0.

For security, we consider the robustness and unforgeability. Note that both properties are defined under a global setup for establishing the public parameter pp and the global PKI. For notational convenience, we denote an execution of the setup involving an adversary $\mathcal{A}$ as: $\mathsf{MTS.Setup}^{\mathcal{A}}(1^\lambda) \rightarrow (pp, (vk_i)_{i \in \mathcal{P}}, (sk_i)_{i \in \mathcal{H}}, \mathsf{state}_{\mathcal{A}})$, where $\mathcal{P}$ is the setup of all nodes, $\mathcal{H}$ is the set of so-far-honest nodes after the initial corruption, and $\mathsf{state}_{\mathcal{A}}$ is the state of $\mathcal{A}$ at the end of the setup.

ROBUSTNESS. An MTS scheme is robust, if the following conditions are satisfied.

- A partial signature produced by an honest node is always valid. Formally, let sk_i and vk_i be the secret key and the public key of an honest node P_i, then for any message $\mathtt{msg}$, it holds that

$$\mathsf{PartVrfy}(vk_i, \mathtt{msg}, \mathsf{PartSign}(sk_i, \mathtt{msg})) = 1.$$

- Combing any k valid partial signatures by distinct nodes within the universe U yields a valid threshold signature for that message under VK_U. Formally, for any $U \subset \mathcal{P}$, let $(\mathsf{CK}_U, \mathsf{VK}_U) \leftarrow \mathsf{UniverseSetup}(\{vk_i\}_{i \in U}, k)$. For any set $\{(\sigma_j)\}_{j \in S}$ such that $\mathsf{PartVrfy}(vk_j, \mathtt{msg}, \sigma_j) = 1$ for every $j \in S \subset U$ and $|S| = k$, it holds that

$$\mathsf{Vrfy}(\mathsf{VK}_U, \mathsf{Combine}(\mathsf{CK}_U, \{(i, \sigma_i)\}_{i \in S}, \mathtt{msg}), \mathtt{msg}) = 1.$$

UNFORGEABILITY. An MTS scheme satisfies the unforgeability, if any PPT adversary cannot forge a threshold signature for a message for which there are not enough honest participants within a specific universe having generated partial signatures. Formally, for any PPT adversary $\mathcal{A}$, it holds that

$$\Pr[\mathsf{Exp}^{\mathsf{unp}}_{\mathcal{A}}(1^\lambda, n, t) = 1] \leq \mathsf{negl}(\lambda),$$

where the experiment $\mathsf{Exp}^{\mathsf{unp}}_{\mathcal{A}}$ is defined as follows.

$\mathsf{Exp}^{\mathsf{unp}}_{\mathcal{A}}(1^\lambda, n, t)$	$\mathcal{O}_{\mathsf{PartS}}(i, \mathtt{msg})$	$\mathcal{O}_{\mathsf{Corr}}(i)$
$\mathsf{MTS.Setup}^{\mathcal{A}}(1^\lambda) \rightarrow$	**if** $V_{\mathtt{msg}} = \perp$	**if** $i \in \mathcal{H}$
$(pp, (vk_i)_{i\in\mathcal{P}}, (sk_i)_{i\in\mathcal{H}}, \mathsf{state}_{\mathcal{A}})$	$V_{\mathtt{msg}} \leftarrow \emptyset$	$\mathcal{H} \leftarrow \mathcal{H} \setminus \{i\}$
$(U^*, \mathtt{msg}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{PartS}}, \mathcal{O}_{\mathsf{Corr}}}(\mathsf{state}_{\mathcal{A}})$	**if** $i \in \mathcal{H}$	**return** sk_i
if $\mathsf{Vrfy}(\mathsf{VK}_{U^*}, \mathtt{msg}^*, \sigma^*) = 1$	$\sigma_i \leftarrow \mathsf{PartSign}(sk_i, \mathtt{msg})$	
$\wedge\, \lvert V_{\mathtt{msg}^*} \cup (U \setminus \mathcal{H})\rvert < k$	$V_{\mathtt{msg}} \leftarrow V_{\mathtt{msg}} \cup \{i\}$	
return 1	**return** σ_i	
return 0		

INSTANTIATION. Throughout the thesis, we instantiate MTS by the threshold signature scheme (TSS) in (Attema et al., 2021, Section 5) with PKI assumption and a transparent setup. This instantiation gives a size of $O(\lambda \log U)$ threshold signature where U is the size of a specific universe. The computational overhead of Sign and PartVrfy is $O(1)$, inherited from the BLS digital signature scheme, while Combine and Vrfy inherit $O(U)$ computational cost from (Attema et al., 2021).

Additionally, it is important to note that TSS in (Attema et al., 2021) does not provide nor require combine or verification keys, hence we can plug in a specific universe U as both VK_U and CK_U. The class of universes can be done in the setup phase, so each party trivially holds the VK_U and CK_U for all universes.

Cryptographic accumulators. A cryptographic accumulator provides a succinct representation of a set while ensuring efficient membership verification. Formally, such an accumulator scheme, denoted as Acc, comprises the following four algorithms: (1) $\mathsf{Gen}(1^\lambda, n)$ outputs an accumulator key ak; (2) $\mathsf{Eval}(ak, \mathcal{S})$ on inputs ak and a set $\mathcal{S}$ to be accumulated, it returns an accumulated value u for the set $\mathcal{S}$; (3) $\mathsf{Wit}(ak, u, \mathcal{S}, s_i)$ on inputs ak, u for the set $\mathcal{S}$, and an element $s_i \in \mathcal{S}$, it returns a membership witness w_i for s_i; (4) $\mathsf{Vrfy}(ak, u, s_i, w_i)$ returns a bit representing whether s_i is indeed accumulated into u ("1") or not ("0") according to the given witness w_i.

An accumulator scheme is said to be *correct*, if for $ak \leftarrow \mathsf{Gen}(1^\lambda, n)$, any set $\mathcal{S} = \{s_i\}_{i\in[n]}$, $u \leftarrow \mathsf{Eval}(ak, \mathcal{S})$, and any $w_i \leftarrow \mathsf{Wit}(ak, u, s_i)$, it holds that $\Pr[\mathsf{Vrfy}(ak, u, s_i, w_i) = 1] = 1$. An accumulator scheme is *unforgeable*, if for an honestly generated ak, and any PPT adversary,

$$\Pr\left[(\mathcal{S}, s^*, w^*) \leftarrow \mathcal{A}(ak) : s^* \notin \mathcal{S} \wedge \mathsf{Vrfy}(ak, \mathsf{Eval}(ak, \mathcal{S}), s^*, w^*) = 1\right] \leq \mathsf{negl}(\lambda).$$

For simplicity, throughout this thesis, we consider an accumulator scheme whose Eval and Wit are deterministic.

Instantiation. We primarily consider the Merkle tree based accumulator, which solely requires a hash key as its setup, and its witness size is $O(\lambda \log n)$.

Erasure code scheme. A (k, n) erasure code (EC) scheme (Blahut, 1983) consists of two deterministic algorithms Enc and Dec. The Enc algorithm maps any vector $\mathbf{m} = (m_1, \cdots, m_k)$ of k data fragments into a vector $\mathbf{c} = (\mathtt{c}_1, \ldots, \mathtt{c}_n)$ of n coded fragments, such that any k elements in the code vector $\mathbf{c}$ is enough to reconstruct $\mathbf{m}$ due to the Dec algorithm. I.e., for any $\mathbf{m} \in \mathcal{B}^k$ and any $\mathbb{I} \subset [n]$ that $|\mathbb{I}| = k$, we have

$$\Pr[\mathsf{Dec}(\{(i, \mathtt{c}_i)\}_{i \in \mathbb{I}}) = \mathbf{m} \mid \mathbf{c} := (\mathtt{c}_1, \cdots, \mathtt{c}_n) \leftarrow \mathsf{Enc}(\mathbf{m})] = 1.$$

Instantiation. Throughout the paper, we consider a $(t+1, n)$-erasure code where $2t + 1 = n$. Additionally, it's important to note that this erasure code scheme will implicitly select an appropriate $\mathcal{B}$ based on the actual length of each element in $\mathbf{v}$. This guarantees that the encoding increases the data size by no more than a constant factor.

2.3 Consensus Primitives

Multi-valued Validated Byzantine Agreement. In an (n, t, ℓ)-Multi-valued Validated Byzantine Agreement (MVBA) protocol, there are n parties $P_1, \ldots, P_n$, where each P_i has an ℓ-bit input v_i and decides an output, denoted as $\mathsf{MVBA}\langle P_i(v_i)\rangle$. MVBA is defined w.r.t. an external predicate $\mathsf{Predicate}$ on the input values. Against any adaptive PPT adversary $\mathcal{A}$ that corrupts up to t parties, a secure MVBA ensures the following properties with all but negligible probability:

- **Termination.** If each honest node P_i provides an input v_i such that $\mathsf{Predicate}(v_i) = 1$, then every honest party would output a value.
- **External-Validity.** If an honest node outputs a value v, then $\mathsf{Predicate}(v) = 1$.
- **Agreement.** The output messages of all honest parties would be the same.

INSTANTIATION. We consider a variant of the synchronous MVBA protocol provided in (Abraham et al., 2023c), which requires $O(1)$ rounds and incurs the communication cost of $O(n\ell + \lambda n^2 \log n)$ bits in expectation, for agreeing on ℓ bits among n nodes. The underlying threshold signature scheme can be instantiated with the MTS scheme (Attema et al., 2021) where the size of a threshold signature is $O(\lambda \log n)$, and the underlying accumulator scheme is implemented using the hash-based Merkle tree. In addition, this MVBA assumes coin flipping subroutines which we will realize when applying the MVBA protocol.

Assuming the security of the underlying hash function and the coin flipping subroutine, an instance $\mathsf{MVBA}[\mathsf{sid}]$ of this MVBA protocol is secure if the adversary cannot forge an MTS signatures for messages in this instance. Therefore, its security remains even when the adversary can query the signing oracle for messages which do not start with "sid", which helps prove the composition security of the MVBA protocol when every instance has been assigned with a unique identifier.

ROUNDS FOR GUARANTEED TERMINATION. While the MVBA protocol can terminate in a constant number of rounds in expectation, there is still a constant probability that an instance cannot terminate within the rounds. However, in our DKG construction, we need to ensure all MVBA instances in the system can terminate with an overwhelming probability within a fixed number of rounds. Our solution is to set a conservative number of rounds considering both the security parameter and the number of instances in the system. In particular, we have the following lemma.

LEMMA 2. *Let $\kappa \in \mathbb{N}$ be a security parameter. Then, for the MVBA protocol in (Abraham et al., 2023c), there exists an integer $\Delta_{\mathsf{MVBA}} = O(\kappa)$, such that, for any $N \leq 2^{\kappa}$, the probability that every MVBA instance among N instances terminates within Δ_{MVBA} rounds is at least $1 - 2^{-\kappa}$.*

PROOF. As with most randomized consensus protocols that are achieved in the "Las Vegas" style, an execution of the MVBA protocol from (Abraham et al., 2023c) consists of a sequence of iterations, terminating after the first successful iteration. Each iteration requires a constant number of rounds and succeeds with an independent probability of $1/2$. Hence, the probability that a single MVBA instance terminates within T' iterations, denoted as $\mathrm{psingle}$, is $1 - 2^{-T'}$. For $N \leq 2^{\kappa}$ instances, the probability that all instances terminate within T' iterations is $\mathrm{pN} = \mathrm{psingle}^{N} \geq \mathrm{psingle}^{2^{\kappa}}$, and we require $\mathrm{pN} \geq 1 - 2^{-\kappa}$. This leads to the inequality

$$(1 - 2^{-T'})^{2^{\kappa}} \geq 1 - 2^{-\kappa}.$$

Notice that when $T' = 2\kappa$ the above the inequality holds. Furthermore, since each iteration takes a constant number of rounds, the total number of rounds Δ_{MVBA} required for all N instances to terminate with overwhelming probability is also $O(\kappa)$. □

Byzantine Agreement. In an (n, t, ℓ)-Byzantine Agreement (BA) protocol, there are n parties $P_1, \ldots, P_n$, where each P_i has an ℓ-bit input v_i and decides an output, denoted as $\mathsf{BA}\langle P_i(v_i)\rangle$. Against any adaptive PPT adversary $\mathcal{A}$ that corrupts up to t parties, a secure BA ensures the following properties with all but negligible probability:

- **Validity.** If all honest parties share the same input v, they all output v.
- **Agreement.** The output messages of all honest parties would be the same.
- **Termination.** Every honest party would decide on an output message.

INSTANTIATION. We consider the BA protocol implied by (Abraham et al., 2019) and (Nayak et al., 2020). In particular, the BA protocol by Abraham et al. (Abraham et al., 2019) achieves $O(1)$ round complexity and $O(n^2|\mathsf{ThldSig}|)$ communication complexity when the input size $\ell = O(\lambda)$, where $|\mathsf{ThldSig}|$ represents the signature size of the underlying threshold signature scheme. For cases where $\ell > \lambda$, Nayak et al. (Nayak et al., 2020) proposed the state-of-the-art BA extension protocols. Throughout this thesis, we consider the BA protocol obtained by applying Nayak et al. 's extension technique to

Abraham et al.'s protocol, which achieves $O(1)$ round complexity and $O(n\ell + \lambda n^2 \log n + n^2|\mathsf{ThldSig}|)$ communication complexity.

The BA protocol from (Abraham et al., 2019) assumes coin flipping subroutines which we will realize when applying the BA protocol. The underlying threshold signature scheme can be instantiated with the MTS scheme (Attema et al., 2021) where the size of a threshold signature is $O(\lambda \log n)$. The extension protocol proposed by Nayak et al.(Nayak et al., 2020) requires an accumulator scheme and an Erasure code scheme, which can be instantiated by the hash-based Merkle tree and the Erasure code scheme from (Blahut, 1983), respectively. In summary, the overall communication complexity of the BA protocol becomes $O(n\ell + \lambda n^2 \log n)$.

ROUNDS FOR GUARANTEED TERMINATION. Similar to the MVBA from (Abraham et al., 2023c), there is a predetermined integer $\Delta_{\mathsf{BA}} = O(\kappa)$, such that for any $N \leq 2^{\kappa}$, the probability that every BA instance among the N instances terminates within Δ_{BA} rounds is at least $1 - 2^{-\kappa}$.

Byzantine Broadcast. In an (n, t, ℓ)-Byzantine Broadcast (BB) protocol, there are n parties $P_1, \ldots, P_n$, and one of them is designated as the sender P_s which has ℓ-bit input v_s. Against any adaptive PPT adversary $\mathcal{A}$ that corrupts up to t parties, a secure BB ensures the following properties with all but negligible probability:

- **Validity.** If the sender is honest, all honest nodes output v.
- **Agreement.** The output messages of all honest parties would be the same.
- **Termination.** Every honest party would decide on an output message.

INSTANTIATION. We consider the deterministic BB protocol implied by (Momose and Ren, 2021) and (Nayak et al., 2020), which, under a transparent setup, requires $O(n)$ rounds and incurs the communication cost of $O(n\ell + \lambda n^2 \log n)$ bits for broadcasting ℓ bits to n nodes. In particular, the BB protocol in (Momose and Ren, 2021) assumes a threshold signature for reducing communication complexity, which can be realized using the MTS scheme from (Attema et al., 2021) which we discussed above. And the extension technique from (Nayak et al., 2020) requires a cryptographic accumulator which can be implemented using the hash-based Merkle tree. An instance $\mathsf{BB}[\mathsf{sid}]$ of this BB protocol is secure as long as the adversary cannot forge a MTS signatures for messages in this instance. Therefore, its security remains even when the adversary can query the signing oracle for messages which do not start with

TABLE 2.1: Comparison with synchronous common coin protocols without a strong setup.

Schemes	Resi.	Comm. Cost total†	Round† worst case
TCLM (Thyagarajan et al., 2021)	$\frac{n-1}{n}$	$O(\lambda n^3)$	Time-lock Puzzl.
RandPiper (Bhat et al., 2021)	$1/2$	$O(\lambda n^3)$	$O(n)$
Hydrand (Schindler et al., 2020)	$1/2$	$O(\lambda n^3)$	$O(n)$
Scrape(Cascudo and David, 2017)	$1/2$	$O(\lambda n^3)$	$O(n)$
GRand(Bacho et al., 2024)	$1/2$	$O(\lambda n^2 \log n)$	$O(n)$
OptRand(Bhat et al., 2023)	$1/2$	$O(\lambda n^3)$	$O(n)$
Dragon (Feng et al., 2024a)	$1/2$	$O(\lambda n^{2.5})$	$O(n)$
SBKN (Shrestha et al., 2023)	$1/2$	$O(\lambda n^3)$	$O(1)$
Ours (§4.6)	$1/2$	$O(\lambda \kappa n^2 \log n)$	$O(\kappa \log n)$

† Here we consider the communication and round complexities by counting both online and offline without amortization, also in the absence of a trusted setup for threshold cryptosystems like unique threshold signature or threshold VRF.

"sid", which grants the composition security of the BB protocol when every instance has been assigned with a unique identifier, as discussed in (Lindell et al., 2002; Feng et al., 2024a; Cohen et al., 2024).

Distributed coin flipping protocols. A distributed common coin is a random value accessible to everyone in the system. It shall satisfy *unpredictability*, which means the adversary cannot predict the coin value in advance, and *unbiased*, which means the coin distribution is computationally indistinguishable from the uniform distribution. Common coins have numerous applications, like being a key building block for faster blockchain consensus, serving us as a randomness source for cryptographic operations, and more. It is well known that common coins can be efficiently generated in the presence of a strong setup assumption like established threshold VRF and unique threshold signature. However, when such strong assumptions are not granted, it becomes a challenging problem to realize efficient protocols for coin generation. In Table 2.1, we compare a few existing synchronous common coin protocols, in the setting without a trusted setup for threshold cryptosystems like threshold VRF and threshold unique signature[2].

One-round coin-flipping protocol. Cachin et al.'s pioneering work (Cachin et al., 2000) introduces a generic construction for adaptively secure one-round coin-flipping protocols, which requires a DKG setup. While a general DKG protocol, as defined above, suffices for many concrete instantiations of

[2]Notably, we omit various randomness beacon schemes, such as those based on PoW/VDF (Wang and Nixon, 2020), as they pre-suppose an initial coin, making them unsuitable for single-shot common coin generation.

(Cachin et al., 2000), a few coin-flipping protocols (Gurkan et al., 2021; Bacho et al., 2024) rely on specific key structures different from the conventional key structure of DL-based cryptosystems. Notably, these specific key structures might enable more efficient DKG constructions (Gurkan et al., 2021; Feng et al., 2024a; Bacho et al., 2024).

SYNTAX. An (n, t) one-round coin-flipping protocol can be described with the following algorithms or sub-protocols.

- $\mathsf{Setup}(1^\lambda, n, t)$. This includes an interactive DKG protocol involving all participants $\mathcal{P}$. The public outcome of the protocol is the generation of a group public key pk and a list of public key shares $(pk_i)_{i\in[n]}$, while each P_i additionally obtains a secret key share sk_i. In addition, there might be other setup protocols, and we denote the view of P_i in these setups by sv_i.
- $\mathsf{Coin}^V[\mathsf{sid}]\langle\{P_i(pk, (pk_i)_{i\in[n]}, sk_i, \mathsf{sv}_i)\}\rangle \to r_{\mathsf{sid}}$. In an instance with sid, each party P_i inputs the information from the setup phase and multicasts a single message to the network. Based on the messages sent by other parties, P_i can obtain the output r_{sid} in the range V.

We consider the *adaptive* and *concurrent* security of the coin protocol. During the setup phase, the adversary can adaptively corrupt participants, obtain their internal states, and act on their behalf in all subsequent operations. After the setup phase, polynomially many instances with different identifiers may run concurrently. The adversary can observe the messages sent by honest participants in all instances and adaptively corrupt participants. Despite that $\mathcal{A}$ can take any attacking strategies and corrupt up to t nodes, the following properties are ensured except with negligible probability:

- *Availability.* All honest participants can obtain the same public key and the corresponding secret key shares at the end of the setup phase. Under the setup, when all honest nodes are activated in an instance with identifier sid, every honest node P_i can obtain the output $r_{\mathsf{sid}} \in V$.
- *Agreement.* If two honest nodes P_i and P_j output r_{sid} and r'_{sid} in an instance with sid, respectively, then $r_{\mathsf{sid}} = r'_{\mathsf{sid}}$.
- *Bias-resistance and unpredictability.* Let $\mathsf{state}_{\mathcal{A}}^{\mathsf{before}}$ be the state of the adversary $\mathcal{A}$ at the time that less than $t + 1 - |\mathcal{C}|$ honest participants activated the instance with sid, where $\mathcal{C}$ is the set of corrupted parties, and the r_{sid} be the output of an honest node P_i in the instance. Then, for any PPT algorithm $\mathcal{D}$, and $r \leftarrow\!\$\, V$, it holds that

$$|\Pr[\mathcal{D}(\mathsf{state}_{\mathcal{A}}^{\mathsf{before}}, r_{\mathsf{sid}}) = 1] - \Pr[\mathcal{D}(\mathsf{state}_{\mathcal{A}}^{\mathsf{before}}, r) = 1]| \leq \mathsf{negl}(\lambda).$$

THE COIN-FLIPPING PROTOCOL FROM (BACHO ET AL., 2024). Bacho et al. (Bacho et al., 2024) introduced a coin-flipping protocol that requires a DKG for the following key structure: $pk = g_1^x \in \mathbb{G}_1$ and $sk = g_2^x \in \mathbb{G}_2$, where $x \leftarrow\$ \mathbb{F}$, $\mathbb{G}_1$ (with generator g_1) and $\mathbb{G}_2$ (with generator g_2) are groups with scalar field $\mathbb{F}$. A bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ exists. For completeness, we recall their scheme in Appendix B.1.2.

An advantage of the above key structure is that there are adaptively-secure aggregatable PVSS (APVSS) schemes (Cascudo and David, 2017; Bacho and Loss, 2023b) compatible with it, facilitating compatible and communication-efficient DKG designs such as (Gurkan et al., 2021; Feng et al., 2024a; Bacho et al., 2024). However, the original DKG for it requires $O(n)$ rounds. Looking ahead, in Section 4.6, we provide a new DKG protocol for the coin-flipping protocol in (Bacho et al., 2024), achieving both $\widetilde{O}(n^2)$ communication complexity and $\widetilde{O}(1)$ rounds.

CHAPTER 3

Scalable and Adaptively Secure Any-Trust Distributed Key Generation and All-hands Checkpointing

---

3.1 Introduction

Distributed key generation protocols (DKG) (Pedersen, 1991; Gennaro et al., 2007) enable a set of participants to jointly generate a public key, and each of them outputs a secret key share. It is a classical topic and the basis of threshold cryptography. They were usually considered in small-scale in-house applications. There are recent resurge of interests of those protocols, mostly because of a diverse set of new blockchain applications, for example, cross-chain bridge (Lee et al., 2023; Cerulli et al., 2023), MEV protection (Malkhi and Szalachowski, 2022; Ruby.Exchange, 2021; Total-blockchain, 2022), censorship resistance in asynchronous consensus (Miller et al., 2016; Guo et al., 2020), checkpointing into Bitcoin (Azouvi and Vukolic, 2022), and more. Those new applications raise a new fundamental challenge of deploying distributed key generation on a very large scale.

Enhancing Proof of Stake Security via All-hands Checkpointing. As one main motivational example of large-scale DKG, we elaborate on the checkpointing mechanism, which aims at addressing a prominent security challenge in the Proof-of-Stake (PoS) blockchain known as *long-range attacks* (Steinhoff et al., 2021). At a high level, in a PoS blockchain, validators (with stakes) are in charge of proposing blocks; as time evolves, validators' secret keys could be leaked or simply sold (when all coins are spent) to the adversary, who can now easily create a fork from a historic block using the corresponding secret keys. For a newly joined node, such a fork is also considered valid. This long-range attack hints at an inherent vulnerability of "revisionist history" in PoS (and other resources such as space) blockchain.

One promising defense approach is to leverage proof-of-work (PoW) blockchains which are immune to such attacks, particularly to periodically let *the whole PoS network* (e.g., all validators) produce checkpoints and put them into Bitcoin. A recent work, Babylon (Tas et al., 2023), let all validators generate checkpoints via multi-signatures and select some of them to post the checkpoints to Bitcoin. It follows that the number of Bitcoin transactions needed for each checkpoint grows linearly in the number of PoS validators (to at least put a bit vector for indicating corresponding public keys). Particularly, for a PoS chain with 2^{12} validators (e.g., Filecoin), the annual cost would be over 6 million USD (using the Bitcoin price retrieved on March 31st, 2024, and assuming checkpoints are created hourly).

Instead, Filecoin proposed a blueprint for the checkpointing mechanism called Pikachu (Azouvi and Vukolic, 2022) via threshold Schnorr signature (Komlo and Goldberg, 2020; Gennaro et al., 2007). Specifically, all validators of a PoS chain need to run a DKG protocol for *every epoch*, and the resulting public keys will serve as Bitcoin addresses. At epoch i, the validators from epoch $i-1$ will jointly create a Bitcoin transaction via a threshold Schnorr signing protocol, which contains the state of the PoS chain at epoch $i-1$ and transfers all coins from the address created in epoch $i-1$ to the newly created address. In doing so, the Bitcoin transactions uniquely decide the state of the PoS chain in each epoch, and they are verifiably endorsed by the majority of validators. Since this approach only needs exactly one Bitcoin transaction for each checkpoint, the Bitcoin transaction fees incurred are always a small constant regardless of the scale of the PoS blockchain network and much lower than Babylon's approach.

Despite being appealing, and recent progress on threshold Schnorr signatures (Komlo and Goldberg, 2020; Crites et al., 2023; Bellare et al., 2022) could potentially be deployable, Pikachu remains in theory (or toy prototype) as all validators have to jointly run a DKG protocol for every checkpoint (as validator set evolves thus previous DKG cannot be reused). Particularly, even a moderate-scale PoS chain can have thousands of validators. Moreover, some applications need to be done in a "timely" manner, which makes the task more challenging. For example, Filecoin currently has around 2^{12} validators, with anticipated growth to 2^{14} validators in the future, while checkpoints may be supposed to be created hourly (as suggested in (Tas et al., 2023)). To deploy the Pikachu checkpointing into Bitcoin mechanism (Azouvi and Vukolic, 2022) for real-world blockchains (for example, Filecoin), we need to design a scalable DKG protocol that can be efficiently run among all validators in popular public blockchains.

Existing DKGs are practically infeasible at a whole-chain scale. Let us first briefly introduce the common paradigm for DKG protocols, which subsumes most DKG constructions, including (Pedersen,

1991; Gennaro et al., 2007; Kate et al., 2010; Tomescu et al., 2020; Zhang et al., 2022), to illustrate the astronomical communication and computation costs of existing DKGs in a large scale.

In a nutshell, among n participants where up to t could be adversarial, each participant P_i selects a t-degree polynomial f_i to define $sk^{(i)} = f_i(0)$. They then deliver the share (as a dealer of a verifiable secret sharing (VSS) (Pedersen, 1991)) $sk_j^{(i)} = f_i(j)$ to other P_j and *broadcast* [1] a commitment, com_i, for the polynomial $f_i(X)$. Then, each participant P_j could verify if $sk_j^{(i)}$ is a valid share w.r.t. com_i; If there are invalid shares, the participants will collectively engage in a *complaint* phase, where they *broadcast* complaints and identify the set of qualified dealers $\mathsf{Qual} \subset [n]$, ensuring that all transmitted secret shares are valid. The final secret share for P_i is $sk_i = \sum_{j \in \mathsf{Qual}} sk_i^{(j)}$, and the aggregate secret key is $sk = \sum_{j \in \mathsf{Qual}} sk^{(j)}$.

The DKG scheme by Kate, Zaverucha, and Goldberg (Kate et al., 2010) (and its recently improved version by Zhang et al. (Zhang et al., 2022), dubbed KZG hereafter) represents the state of the art following this paradigm. In KZG, the commitment to the polynomial f_i is constant in size and enables validating a secret share with constant-sized information. In an optimistic case, when all participants are honest, KZG protocol can remain efficient even for large-scale deployment, as demonstrated in (Zhang et al., 2022). However, since there is a constant fraction of adversarial participants, its performance got dramatically worse. Particularly, when another node complains about a dealer, the dealer shall *broadcast* the corresponding secret share for public verification. Hence, in facing $O(n)$ malicious nodes, each complaining $O(n)$ nodes, there are $O(n^2)$ shares to be broadcasted, and every node shall verify these shares. Indeed, improving the adversarial case performance is the major open problem left by (Tomescu et al., 2020).

We can readily anticipate that both communication and computation costs would skyrocket as the scale increases, as seen in scenarios like Filecoin validators. For instance, with 2^{12} participants in the DKG protocol, the entire network would need to transmit *tens of terabytes* of data, while each node would have to allocate *multiple hours* to verify shares (in response to complaints) to produce just one public key![2]

We remark that publicly verifiable secret sharing (PVSS) can eliminate the complaint phase in DKG (Cascudo and David, 2017; Fouque and Stern, 2001; Gentry et al., 2021b; Bacho and Loss, 2023a), as

[1] Broadcast satisfies *agreement*, i.e., all parties receive the same message even when the sender is malicious. Thus, it is more complicated and expensive than multicast.

[2] For detailed numerical estimates and comparisons, we refer to Section 3.9.1.

each party could now broadcast all "encrypted" shares and enable the public verification immediately. However, existing PVSS-based DKGs are either even more costly than KZG (Fouque and Stern, 2001; Groth, 2021) or only generate group-element secrets, while mainstream threshold cryptographic protocols like threshold Schnorr signatures use field-element secrets. In addition, no existing PVSS schemes with field-element secrets are provably secure against adaptive attackers (Bacho and Loss, 2023a), while adaptive security is desired in practice[3].

Besides those high costs, one extra challenge may make things even worse: in PoS chains, validators usually have different *weights* (proportional to the number of held stakes), while a threshold of weights is assumed to belong to honest validators. Simply viewing a validator as a participant in DKG can be leveraged by an adversary to amplify its power: the adversary can choose to corrupt many validators with small weights and eventually control the majority of DKG participants within its budget. A naive approach is to allocate different numbers of sub-IDs proportional to their weights. Each sub-ID is then treated as an independent participant. While this approach addresses the security concern, it can lead to an enormous number of sub-IDs. For example, we would need to allocate 674 trillion sub-IDs to 3700 Filecoin validators[4].

It follows that the following question remains:

Could we have a practically feasible DKG protocol that enables all-hands participation in blockchains with weighted validators, as well as being adaptively secure?

[3]While there are two very recent PVSS-based DKG works (Bacho et al., 2023b; Feng et al., 2024a) have further pushed down the asymptotic complexity of DKG, they are either only with group-element secrets or statically secure.

[4]`https://filfox.info/en/ranks/power`. It has a total mining power of around 25 EB, while the power unit is 32KB, so 674 trillion sub-IDs are needed.

3.1.1 Our Results

In this chapter, we give an affirmative answer to this question. We proceed in two main steps:

TABLE 3.1: Comparison with the state-of-the-art DKGs for DLog-based Cryptography.

Schemes	Resilience	Adap.?*	Comm. Cost (total)***		Comp. Cost (per node)**	
			Good†	Bad†	Good†	Bad†
Pedersen (Pedersen, 1991)	$1/2$	✓	$O(n\mathcal{B}(n\lambda)) + O(n^2\lambda)$	-	$O(n^2)$	-
KZG (Kate et al., 2010) (Zhang et al., 2022)	$1/2$	✓	$O(n\mathcal{B}(\lambda))+ O(n^2\lambda)$	$+O(n\mathcal{B}(n\lambda))$	$O(n\log n)$	$+O(n^2)$
GHL (Gentry et al., 2022)‡	$1/2$	✗	$O(n\mathcal{B}(n\lambda))$		$O(n^2)$	
GJM+ (Gurkan et al., 2021)‡	$\log n/n$	✗	$O(n\mathcal{B}(\lambda) + \log n\mathcal{B}(n\lambda))$		$O(n\log^2 n)$	
BHK+ § (Benhamouda et al., 2022)	$\approx 1/4$	✓	$O(C\mathcal{B}(C\lambda))$ + $O(C\mathcal{M}(C^2\lambda))$	-	$O(C^3)$	-
Ours (Section 3.5)§	$1/2$	✓	$O(s\mathcal{B}(n\lambda))$	$+O(n\mathcal{M}(s\lambda))$	$O(sn)$	-

*Adap.? asks if the protocol is adaptively secure, and we accept the relaxed definition from (Bacho and Loss, 2022)
**Comp.Cost measures the number of group exponentiation operations performed by each node.
*** Comm.Cost measures the total communication cost, where $\mathcal{B}(\ell)$ (or $\mathcal{M}(\ell)$) denotes the cost of one node broadcasting (or multicasting) ℓ bits to the network, and $O(\ell)$ means there are $O(\ell)$ bits sent by honest nodes over pair-wise channels.
†For both communication and computation, Good considers the cost without complaints, Bad considers the **extra** cost when facing the maximal number of complaints; "-" represents no asymptotically greater cost. ‡GHL and GJM+ do not have a complaint phase.
§BHK+ and ours are committee-based approaches. For ensuring the quality of the committee with high probability (say $1 - 5 \times 10^{-9}$, as adopted by Algorand (Chen and Micali, 2019)), BHK+ needs the committee size of $C \approx 6000$ (estimated based on (Benhamouda et al., 2020)), while ours only needs $s = 38$ (further analysis available in Table.3.2).

A scalable and adaptively secure DKG. Our primary result is a practical DKG protocol (called Any-Trust DKG) for DLog-based cryptographic systems. We compare our scheme with state-of-the-art efficient DKG constructions[5] in Table 3.1 and discuss more related works in Section 3.2. Particularly, our DKG protocol features:

– *Efficiency:* It enjoys (quasi-)linear (in n) per-node computation complexity[6], even in adversarial cases. Additionally, the size of *all* data to be broadcasted also only grows linearly in n. In contrast, previous constructions suffer from quadratic per-node computation and quadratic broadcast overhead. Specifically, as our experiments will demonstrate, for $n = 2^{12}$, each node can complete all computation tasks in approximately 26 seconds, with the data to be broadcasted totaling around 7.7 MB in size.

[5]We focus on fully synchronous networks; asynchronous or partial synchronous DKGs (Gao et al., 2022; Abraham et al., 2023b; Das et al., 2022) are not included in the table, as their implementations cannot simply leverage a broadcast channel and appear to be more expensive.

[6]Although evaluating $O(n)$-degree polynomials at $O(n)$ points inherently causes $O(n\log n)$ computation, we only require $O(n)$ expensive group exponentiations.

– *Security:* Our protocol achieves optimal resilience and satisfies the security definitions achievable by classical DKGs. Specifically, in the presence of adaptive adversaries (who cannot retract messages sent by honest parties, as in many settings, such as Algorand (Gilad et al., 2017)), our scheme complies with the oracle-aided algebraic simulatability, which was recently introduced in (Bacho and Loss, 2022) for capturing the adaptive security of many practical DKGs including Pedersen (Pedersen, 1991) and KZG.

We remark that our primary goal is to enable massive-scale DKG; we may use resources that are naturally available in the application settings. Compared with the classical DKG schemes, our protocol additionally leverages one common coin that is generated after all participants' public keys are determined (as the YOSO-model DKG (Benhamouda et al., 2022)), which is available in most blockchains. Nonetheless, it enables a distributed randomness beacon (Choi et al., 2023) for continuous coin generation, by applying threshold unique signatures with secret shares from the DKG (Cachin et al., 2005) . We introduce a set of techniques for the efficient, adaptively secure DKG (detailed in the next section).

A generic transformer for weighted distributed protocols. We present a generic sub-ID allocation mechanism that enables us to efficiently apply conventional distributed protocols in the weighted setting. Our sub-ID allocation mechanism deterministically decides the number of sub-IDs for each validator according to the weight distribution, such that every sub-ID will be viewed as an individual participant in the subsequent protocol.

The trivial sub-ID allocation method that precisely preserves the portion of each validator's weight may need to issue tremendously many sub-IDs. In contrast, we have noticed and leveraged a gap between the usual assumption on the honest participant's weight ratio (assumed to be more than 2/3 due to other system components) and the honest ratio needed in threshold cryptography (usually just above 1/2). Particularly, our sub-ID allocation method is lossy-yet-qualified, guaranteeing that more than half of the sub-IDs will be issued to honest participants if they possess over 2/3 of the weights, and therefore it can be much more *compact*. The number of sub-IDs issued by our method for n validators is probably at most $2n$, regardless of the weight distribution. For the real-world weight distributions of blockchain validators, the issued sub-IDs can even be fewer. For example, our method gives only 1688 sub-IDs instead of 674 trillion to the 3700 Filecoin validators. Compared with concurrent work in (de Souza and Tonkikh, 2023), ours issues fewer sub-IDs for large validator sets like Filecoin's. More details can be found in Section 3.6.

Implementation and Evaluation. We implement our protocol in Java[7] and deploy it on AWS EC2 instances with 16, 32, 64, 128, and 256 nodes. The results demonstrate that our protocol scales effectively and completes within a few seconds (adding some ledger waiting time, which could vary depending on the blockchain if we instantiate the broadcast channel via a distributed ledger.). Additionally, we conduct computational time tests for various values of n, ranging from 2^9 to 2^{15}. In comparison with the state-of-the-art DKG protocol KZG (Kate et al., 2010), our protocol's performance in both the good-case and worst-case scenarios is comparable to or even superior to KZG's performance in the good-case scenario, while KZG's cost in the adversarial case experiences a dramatic increase. Notably, for $n = 2^{12} \sim 2^{14}$, a node in our protocol can finish all computation tasks within around 26 ~181 seconds, even facing the maximal amount of Byzantine nodes. The total amount of data to be broadcasted is around 7.7 ~30.6 MB; If the nodes broadcast the data by posting it on the Filecoin blockchain, it takes around 5 minutes ~20 minutes. The experimental results show that our protocols effectively enable massive-scale DKG deployment.

Deployment friendliness: It is worth noting our DKG is more friendly for large-scale deployment since our DKG makes exclusive use of multicast channels (besides broadcast channels), which can be efficiently implemented, e.g., with gossip protocols and does not require a node to know the IP addresses of all other peers. In contrast, both Pedersen DKG and KZG DKG require pair-wise *private* channels for their efficiency claims. While it is not infeasible for large-scale deployment, it does add extra difficulties and overheads, particularly in public blockchain settings.

Application: better all-hands checkpointing. We then apply our techniques to realize the checkpointing blueprint Pikachu of Filecoin (Azouvi and Vukolic, 2022) that requires all validators to participate. After our optimized sub-ID allocation, we execute a DKG and a threshold Schnorr signature (Shoup, 2023) among these 1688 sub-IDs to create a checkpoint. With our Any-Trust DKG, the DKG phase only incurs around 3MB of broadcast messages in total. Each node can complete all computations in just a few seconds, even when facing the maximum number of complaints. Regarding the threshold signature, we use Any-Trust DKG again to generate the nonce in the GJKR (Gennaro et al., 2007) signing protocol, resulting in a non-interactive threshold signing protocol (after nonce-generation), eliminating the potential single point of failures in coordinator-based protocols like FROST (Komlo and Goldberg, 2020).

[7] Our code is available at `https://github.com/mtc2000/AnyTrustDKG`.

Compared with the existing checkpointing scheme Babylon (Tas et al., 2023), in which the number of Bitcoin transactions per checkpoint grows linearly to the scale of the blockchain, ours/Pikachu only requires exactly *one* Bitcoin transaction for each checkpoint. This difference is reflected in the monetary cost. As an example with Filecoin, the estimated Bitcoin transaction fee incurred annually using Babylon would be over six million USD, while only 26,048.8 USD using ours/Pikachu. More details can be found in Section 3.8.

3.2 Related Works

VSS-based DKG. Distributed Key Generation (DKG) has been a prominent area of research for several decades. Pedersen's seminal work (Pedersen, 1991) established the foundation in this field by introducing an efficient protocol for DLog-based cryptosystems. This protocol builds upon Feldman's Verifiable Secret Sharing (VSS) (Feldman, 1987). Within this scheme, each participant collaboratively runs n instances of Feldman's VSS, taking on the role of the dealer in one of these instances.

In the VSS framework established by Feldman, the dealer is required to broadcast a commitment to a polynomial while distributing the shares privately among all participants. Given that the commitment's size is proportional to $O(n\lambda)$, the resultant communication overhead becomes $O(n\mathcal{B}(n\lambda))$. Additionally, Pedersen's DKG involves a complaint phase where participants broadcast any grievances against dishonest dealers. If a participant were to lodge multiple complaints concurrently, the communication overhead of this phase is likewise $O(n\mathcal{B}(n\lambda))$. It is vital to highlight that during this phase, each participant may validate up to $O(n^2)$ shares. In Feldman's VSS, the computational effort to validate a single share is equivalent to $O(n)$ group operations. This implies a per-node computational burden before the complaint phase of $O(n^2)$, which can potentially amplify to $O(n^3)$ during the complaint process.

A majority of DKG architectures conform to the joint-VSS model. In essence, any innovative VSS protocol can be adapted into a new DKG protocol. Furthermore, given that VSS can be constructed using polynomial commitments, any polynomial commitment scheme can be evolved into both a VSS and, consequently, a DKG. A significant advancement in this field was made by Kate et al. (Kate et al., 2010), who proposed the first polynomial commitment (abbreviated as KZG) with a commitment size of $O(\lambda)$. This innovation ensures that prior to the complaint phase, the communication overhead can be reduced to $O(n\mathcal{B}(\lambda))$. A notable feature of the KZG polynomial commitment is its efficiency in verifying shares; the computational cost for verifying a single share is a mere $O(1)$. This denotes that the computational overhead for each node, in terms of verification before the complaint phase, is simply $O(n)$ in group operations, but this can rise to $O(n^2)$ during the complaint process. Historically, the computational load for producing a polynomial commitment was believed to be $O(n^2)$ (Tomescu et al., 2020). However, a recent exploration by Zhang et al. (Zhang et al., 2022) revealed that the computational overhead for generating a KZG commitment can be streamlined to $O(n \log n)$. It's noteworthy that although KZG requires a CRS setup, there have been other efforts (Zhang et al., 2022; Yurek et al., 2022) that prioritize efficient polynomial commitments without relying on a trusted setup, but these don't match KZG's efficiency.

PVSS-based DKG. Fouque and Stern (Fouque and Stern, 2001) offered a solution that sidestepped the necessity for a complaint phase by incorporating publicly verifiable secret sharing (PVSS). In the event that a PVSS transcript consists of $O(n)$ ciphertexts, the communication overhead will naturally be $O(n\mathcal{B}(n\lambda))$ should every participant choose to broadcast this transcript. Historically, the validation of a PVSS transcript required an overhead of $O(n^2)$, suggesting that the per-node computational overhead in DKG might ascend to $O(n^3)$. However, this obstacle was surmounted by Cascudo and David with their Scrape protocol (Cascudo and David, 2017), which introduced a PVSS methodology that caps the verification duration at $O(n)$. It's worth highlighting that Scrape's strategy is versatile and can be applied to improve many VSS-based DKG schemes, including that of Pedersen's, ensuring that computational overhead during the complaint phase is kept at $O(n^2)$ and doesn't spike to $O(n^3)$. A few recent works focus on improving the concrete performance of PVSS schemes, including the lattice-based PVSS (Gentry et al., 2022), Groth's PVSS (Groth, 2021), and PVSS using class groups (Kate et al., 2023).

Aggregatable PVSS-based DKG. Aggregatable PVSS schemes (Gurkan et al., 2021) are PVSS schemes whose transcripts can be concisely merged into one. There are a few designs that leverage customized communication protocols rather than simply leveraging Byzantine broadcast protocols (or broadcast channels), enjoying asymptotically better complexity. Notably, Gurkan et al. (Gurkan et al., 2021) leveraged an aggregatable PVSS combined with gossip protocols to craft a publicly verifiable DKG. Their communication overhead is streamlined to $n\mathcal{B}(\lambda) + \log n \cdot \mathcal{B}(n\lambda)$ as opposed to $n\mathcal{B}(n\lambda)$, with their per-node communication overhead being $O(n \log^2 n)$. It's pertinent to note, however, that their model can only accommodate $O(\log n)$ Byzantine nodes. Very recently, Feng et al.(Feng et al., 2024a) and Bacho et al. (Bacho et al., 2023b) leverage specially designed communication protocols together with aggregatable PVSS schemes and present DKG schemes with sub-quadratic per-party computation/communication cost while enjoying optimal resilience.

Note that existing aggregatable PVSS schemes all produce secrets in an Elliptic curve group, thus incompatible with many threshold cryptographic protocols. Feng et al. (Feng et al., 2024a) also give a variant of DKG using conventional PVSS schemes but with slightly higher (still sub-quadratic) per-party complexity.

DKG in the YOSO model. A common strategy to enhance scalability is selecting a committee and executing the threshold cryptographic systems within this smaller subset. However, this approach is fraught with challenges. Once aware of the committee's composition, an adaptive adversary can compromise the entire group, thereby undermining security. Furthermore, given that each member of the

committee is required to contribute multiple times during both key generation and subsequent threshold operations, methods like silent committee sampling (e.g., using a verifiable random function (Chen and Micali, 2019)) and assuming memory erasure fail to provide protection against adaptive attackers. Recent advances in the YOSO (You-Only-Speak-Once) MPC realm (Gentry et al., 2021a; Benhamouda et al., 2022) hint at potential solutions to deter adaptive adversaries targeting the committee. Benhamouda et al. (Benhamouda et al., 2022) presents a DKG in the YOSO model. However, the YOSO techniques come with their own set of challenges. Notably, existing YOSO techniques (if without using resource-intensive tools like fully homomorphic encryption (Gentry et al., 2021b)) need to sample a huge committee, say with a few or tens of thousands of nodes, which is already as large as the network scale we are interested in, let alone the extra overhead incurred by using YOSO techniques. Furthermore, as successive committees remain anonymous, inter-committee communication is heavily dependent on a broadcast channel.

On the security of DKG. Beyond endeavors aimed at bolstering the efficiency of DKG, various research initiatives have tackled this challenge from different perspectives. Gennaro et al. (Gennaro et al., 2007) pinpointed vulnerabilities in Pedersen's DKG where the secret key distribution could be manipulated by adversaries. They addressed this flaw by achieving complete secrecy, albeit with a higher computational overhead. Gurkan et al. (Gurkan et al., 2021) conceptualized a milder form of secrecy, coined as "key-expressability", which assumes that adversaries can influence key distribution but within predetermined constraints. They postulated that a key-expressable DKG suffices for many applications, with multiple DKG architectures, including Pedersen's (Pedersen, 1991), Fouque-Stern's (Fouque and Stern, 2001), and our own, fitting this criteria. Another remarkable contribution by Canetti et al. (Canetti et al., 1999) introduced a DKG protocol with adaptive security, a departure from our model and numerous others that ensure security only against static adversaries. Bacho and Loss's recent work (Bacho and Loss, 2022) put forth an oracle-aided adaptive definition and ascertained that several protocols, including (Pedersen, 1991; Fouque and Stern, 2001), conform to this definition in the algebraic group model. Our model also complies with this adaptive security definition.

Asynchronous DKG. Lastly, some recent research efforts (Gao et al., 2022; Abraham et al., 2023b; Das et al., 2022) have pivoted towards DKG within asynchronous networks. These designs adopt the joint-VSS blueprint and depend on an asynchronous broadcast protocol, referred to as "reliable broadcast" (Bracha, 1987), to guarantee verifiability, yet they encounter the cubic computational challenge.

Notably, Das et al. (Das et al., 2022) showcased the inaugural asynchronous DKG with a communication overhead of $O(n^3\lambda)$ for field-element secrets, whereas Abraham et al. (Abraham et al., 2023b) furnished an adaptively secure asynchronous DKG with identical complexity.

3.3 Technique Overview

We give a high-level overview of how we leverage various techniques to lead to our DKG protocol. Through the analysis of how DKG usually works, we observe one major reason for the inefficiency (both high communication and high computation costs) is due to the following simple facts: everyone *broadcast* shares to everyone, and broadcast channels are expensive!

Starting observation for efficiency: Selecting an any-trust group as VSS dealers. Our starting observation is that letting all participants act as dealers of verifiable secret sharing (VSS) schemes is actually *unnecessary*. Recall that in the common DKG paradigm, the final secret is $sk = \sum_{j \in \mathsf{Qual}} s^{(j)}$, where $s^{(j)}$ is the secret dealt by a qualified participant P_j, and $\mathsf{Qual} \subset [n]$ is the set of all qualified dealers. However, the existence of **one** honest dealer would suffice for both secrecy and robustness. Particularly for secrecy, a uniformly sampled secret $s^{(j)}$ contributed by an honest P_j could conceal sk to the adversary who may corrupt all other dealers. For robustness, even when the other dealers behave arbitrarily (e.g., go offline), one honest dealer ensures the set Qual is non-empty, and thus sk is well-defined.

Therefore, we propose utilizing a small group of representatives, called an "any-trust" group (as introduced in the context of anonymous communication (Wolinsky et al., 2012)), where we trust at least one member of the group is honest but do not need to know which one to trust. Note that such an any-trust group can be obtained by randomly sampling from the whole population (with an honest majority). Notably, the size s of an any-trust group can be as small as a few tens in practice, which is in stark contrast to that of a group with an honest majority, which can be up to thousands.

If focusing on *static* adversaries who cannot corrupt parties during the protocol execution, the observation alone already leads to an efficient solution. Particularly, before the complaint phase, there are only s commitments in total to broadcast and only s secret shares for each party to verify. In the worst case, each party at most needs to verify $O(sn)$ shares, which is still feasible.

Challenges and techniques for adaptive security. Achieving efficiency while preserving adaptive security is, however, non-trivial as the adversary does have the budget to corrupt the *entire* any-trust group. Indeed, there are multiple difficulties from different layers, and we need different techniques to conquer them.

Preventing the damage of corrupting the entire any-trust group. We adopt the standard techniques from existing adaptively secure Byzantine agreement protocols (Gilad et al., 2017; David et al., 2018) to prevent the damage of entire any-trust group corruption. Particularly, we use the following techniques or assumptions.

VRF-based sortition. We use the verifiable random function (VRF) based sortition (Gilad et al., 2017) to select the any-trust group, such that only a party itself knows whether it has been selected, which prevents an adaptive adversary from targeting the group before the group members send out their first messages.

Memory erasure (assumption). Once the any-trust group of dealers sends out messages, the adversary will be aware of their identities and proceed to corrupt them. We, therefore, require all these dealers to *erase* all internal states related to dealing secrets (but not the long-term secret keys for signing and decryption) at the same time they send messages. Consequently, even when the adversary corrupts them, it cannot learn the secrets dealt by them.

Forward-secure signatures. However, the adversary can still violate the robustness and secrecy by sending different messages on behalf of newly corrupted dealers. In this case, an initially honest dealer may be disqualified by the network due to the disturbing messages sent by the adversary. To prevent such an attack, we apply forward-secure signatures (David et al., 2018), which ensures no further valid messages can be generated in this round after the dealer erases its secret states.

Efficiently deciding the qualified dealer set with silent dealers. Recall that in the conventional DKG schemes, including KZG, a dealer shall repudiate complaints (that he was silent) by broadcasting the corresponding shares for public verification. However, in our construction, all dealers may be corrupted after the dealing phase. Also, for security, they have already erased all internal states and cannot repudiate anyway. We must enable the network to decide on the qualified set of dealers while the dealers remain silent.

We tackle the problem by designing *publicly verifiable complaints*, such that a dealer can be disqualified immediately once such a complaint against it has been presented, without the need for further repudiation from the dealer. There are two types of complaints: (1) the dealer does not send anything to the receiver. (2) the dealer sent an invalid share to the receiver. To make type (1) public verifiable, we let each dealer in the deal phase *broadcast* the vector of *all encrypted* shares under the receivers' public keys, such that

everyone can check the existence of ciphertexts. [8] For type (2), we leverage *verifiable decryption*: if the decrypted share is invalid, the receiver can generate a NIZK proof showing the share is the correct decryption, which, together with the share itself, serves as a publicly verifiable complaint.

Why not using PVSS? We note that a publicly verifiable secret sharing (PVSS) scheme may look suitable for the setting with silent dealers, as the qualified set can be determined without the complaint phase. However, as we discussed before, existing PVSS schemes that produce field-element secrets are not adaptively secure. Moreover, our approach enables significantly better performance, due to the following reasons: (1) our approach incurs asymptotically lower verification cost for each node, as one verifiable complaint is sufficient to disqualify a malicious dealer, which means that a node needs to verify correctness proofs for at most $O(s + n)$ decrypted values; In contrast, in a PVSS-based approach, a node has to verify the encryption correctness proofs for all $O(sn)$ encrypted shares. (2) Proving and verifying decryption correctness can be pratically more efficient. Our scheme can be instantiated with standard ElGamal encryption and Schnorr proof, while proving the validity of encrypted shares usually requires a special encryption scheme (e.g., Paillier (Lindell and Nof, 2018), Lattice-based (Gentry et al., 2022)) and a range proof (Gentry et al., 2022), which are considerably more expensive. (3) In our approach a node is not required to prove or verify decryption corretness when there is no complaint, while proving and verifying encryption correteness are always mandatory in PVSS-based approaches.

Simulating encrypted shares in the face of adaptive corruption. Now an honest (selected) dealer needs to broadcast the sequence of encrypted shares $(\mathsf{Enc}(ek_1, f(1)), \mathsf{Enc}(ek_2, f(2)), \ldots, \mathsf{Enc}(ek_n, f(n)))$, where ek_i is the public encryption key of the party P_i, and f is the secret polynomial such that $f(0)$ defines his secret. When an adversary corrupts P_i, it knows the decryption key dk_i and thus the decrypted share $f(i)$. However, in the security proof, a simulator should not know $f(0)$ and all $f(i)$'s, while it needs to generate all ciphertexts to simulate an honest dealer. Under adaptive corruptions, we essentially need a non-committing public key encryption scheme (Brunetta et al., 2024), which enables the simulator to generate valid ciphertexts without knowing the plaintexts and later open the ciphertext to an arbitrary value. However, general non-committing encryption is impossible in the standard model, unless the secret key is unresonably long (Nielsen, 2002; Brunetta et al., 2024). We may employ a public key encryption scheme in the random oracle model to circumvent this difficulty [9]. Particularly, let us

[8] We remark that one can broadcast the vector of shares at a marginal cost increase compared to broadcasting one share by leveraging the effective broadcast extension trick, such as (Nayak et al., 2020).

[9] Now we can see the choice that we do not prove the validity of encrypted shares is critical, as otherwise, we may not use random-oracle model PKE.

think about the hybrid ElGamal encryption: we have $ek = g^x \in \mathbb{G}$, $dk = x \in \mathbb{Z}_p$, and the ciphertext c in the form of $(g^r, \mathsf{Hash}(ek^r) \oplus m)$, where $g \in \mathbb{G}$ is the generator of the group $\mathbb{G}$ of prime order p, $r \in \mathbb{Z}_p$ is the fresh encryption randomness, m is the plaintext, Hash is a hash function modeled as a random oracle, and $\oplus$ is the XOR operation on the message space (assuming binary encoded for simplicity). Then, the simulator could first generate a ciphertext as (g^r, u), where u is uniformly sampled from the plaintext space. Later, when the plaintext m is known, the simulator programs the random oracle such that $\mathsf{Hash}(ek^r) = u \oplus m$, which opens the ciphertext to m.

Preventing leakages due to publicly verifiable complaints. We observe that our publicly verifiable complaints expose the decrypted results to the public, which, in some sense, provides a *decryption oracle* and can potentially be leveraged by malicious nodes to break the confidentiality of the encryption scheme. We can patch this issue by employing a chose-ciphertext-attack (CCA) secure encryption scheme. However, as we already have many other requirements for the encryption scheme, we must be careful to ensure all requirements are compatible. For example, the encryption scheme must be non-committing and require programmable random oracles, which cannot coexist with standard CCA approaches like Naor-Yung (Naor and Yung, 1990). Meanwhile, our complaint phase needs efficient proof of decryption, which means the ciphertext must preserve some structures to enable efficient proof systems.

We use a signature of knowledge (Chase and Lysyanskaya, 2006) to handle this issue. Specifically, for the hybrid ElGamal encryption whose ciphertexts are in the form of $(c_0 = g^r, c_1 = \mathsf{Hash}(ek^r) \oplus m)$, we require the dealer P_i who produces (c_0, c_1) to sign its ID i using the knowledge of r against $c_0 = g^r$. Then, in the security proof, the simulator could *extract* r from the signature of knowledge, which enables the simulator to know the encrypted share without the help of a decryption oracle. We will see other benefits of this approach when we detail the concrete encryption scheme.

Further optimizations to DKG: After overcoming the difficulties of adaptive security, we turn back to optimizing the performance.

Reducing communication of complaints by any-trust group again. A straightforward complaint phase is to let all nodes directly broadcast their (verifiable) complaints to the network. In practice, it means there could be $O(n)$ broadcast again, which can incur an unpleasant overhead. In addition, the cost cannot be reduced by our extended broadcast channel techniques either. Our broadcast technique enables one node to broadcast large-size messages, but now there are many senders.

We optimize the complaint phase via the following observation: one valid complaint is enough to disqualify a dealer, and thus, there is no need to include all complaints in the broadcast channel. We, therefore, design a complaint phase with the following three steps. First, each node disseminates the complaints using a multicast channel so that all nodes receive all complaints made by all other honest nodes. Second, we sample an any-trust group again, and let the group members deduplicate the complaints. Each group member will maintain a concise complaint list that contains at most one complaint for each dealer and all dealers complained by honest nodes. Finally, we let the group members broadcast their complaint lists, which guarantees that all malicious dealers will be disqualified. With the optimized complaint phase, there are $O(s)$ any-trust group members, each posting at most $O(s)$ complaints, where s is the size of an any-trust group.

On the choice of VSS/polynomial commitments. While the VSS scheme in KZG DKG (Kate et al., 2010) is usually believed to be the most efficient instantiation, we do not use it in our DKG scheme due to the following considerations: (1) The polynomial commitment scheme in KZG VSS necessitates a structural common reference string, and securely establishing it in decentralized applications requires additional efforts. (2) The communication benefits of the VSS scheme do not exist in our setting. Although its commitment size is constant, we need to broadcast all encrypted shares (and their encrypted proofs) anyway. (3) The generated public key is in a pairing-friendly group. We need to make an extra effort to adapt it for Schnorr signatures.

Instead, we employ a VSS scheme based on a more classical polynomial commitment. Specifically, the commitment to a t-degree polynomial f is the form of $g^{f(0)}, g^{f(1)}, \ldots, g^{f(n)}$, where $n > t$ is the number shares needed to distribute. By the checking technique from Scrape (Cascudo and David, 2017), a receiver could verify the $n+1$ group elements committing to a t-dgree polynomial at the cost of $O(n)$ group operations. Then, to verify each share $f(i)$, one just needs to perform one group exponentiation operation, such that verifying $O(n)$ shares from the dealer just costs $O(n)$ group operations, which guarantees a computationally efficient complaint phase.

Using multi-recipient encryption. For the PKE scheme, we have proposed the hybrid version of ElGamal, which is *non-committing* and supports verifiable decryption. In our DKG, we use the multi-recipient variant of it (Bellare et al., 2007), which reuses the g^r component across ciphertexts under different public keys. It greatly reduces the broadcast cost, making the ratio of ciphertext size and share size close to 1. Moreover, recall that we use proof of knowledge of r to prevent leakages from decryption oracles, and using multi-recipient encryption will only incur one proof of knowledge by each dealer.

Optimization to broadcast channels: A practical extension trick. Two primary approaches for broadcast channels include using Byzantine broadcast (BB) protocols (Gilad et al., 2017; Chen and Micali, 2019; Dolev and Reischuk, 1982) or utilizing existing infrastructure like blockchains. Implementing a large-scale BB protocol can be intricate and susceptible to errors; thus, using established blockchains is an attractive, simpler, and modular alternative. However, on-chain storage is generally an expensive and scarce resource. While the broadcast cost in our DKG for thousands of participants has been reduced to a few Megabytes, it can still be a considerable burden for blockchains.

Therefore, we present a practical extension to a blockchain-based broadcast channel by leveraging a multicast channel and a data dispersal network (DNN) like IPFS (Trautwein et al., 2022). Our design is simple and modular, retaining the major benefits of using blockchain, and it enables a sender to broadcast an *arbitrarily long* message while incurring *constant* on-chain storage cost. Though it may be folklore to write digests alone into a blockchain to save bandwidth, we are unaware of any design with a formal agreement guarantee. We believe this component may be of independent interest. More details are in Section 3.7

3.4 Modeling

Communication model. We assume the network is synchronous, and protocols proceed by rounds. Every participant has access to multicast and broadcast channels with different guaranteed delivery time. They both achieve *validity*, while broadcast channel additionally guarantees *agreement*.

Validity. When an honest node sends a message via this channel, all honest nodes can receive this message by the end of the round.

Agreement. At the end of a broadcast round, honest receivers always receive the same message from this channel, even when the sender is Byzantine.

Threat model. Prior to protocol execution, every node honestly generates their public key/secret key pairs and sends public keys to all other nodes. After the setup, the adversary can adaptively corrupt any node during the protocol execution and control their subsequent behaviors. Particularly, the adversary controls what messages a corrupted node will send in the same round it gets corrupted. However, messages already multicasted or broadcasted by node i before i become corrupted *cannot be retracted*.

3.5 Any-Trust DKG Protocol

Following the technique overview in Section 3.3, we present our Any-Trust DKG in Section 3.5.1 based on the building blocks in Section 3.4, and analyze it in Section 3.5.2.

Round 1 (broadcast): each P_i do:

1 : // determine whether it is elected as a dealer.

2 : $\mathsf{VRF.Sortition}(rvk_i, rsk_i, \mathsf{rand}, \text{“deal"}, \mathsf{ratio}) \to \mathsf{CR}_i^{\mathsf{deal}}$

3 : **if** $\mathsf{CR}_i^{\mathsf{deal}} = \perp$, // if not, update FS secret key and exit the round

4 : **then** $\mathsf{FS.Update}(\mathsf{FS}.sk_i[1]) \to \mathsf{FS}.sk_i[2]$, **erase** $\mathsf{FS}.sk_i[1]$, **exit Round 1**

5 : // only elected users continue the followings.

6 : sample $(a_0, a_1, \ldots, a_t) \leftarrow\$ \mathbb{Z}_p^{t+1}$, define $f(X) = \sum_{\tau=0}^{t} a_\tau X^\tau$

7 : commit to the random polynomial $f(X)$: $(\mathsf{cm}_j = g^{f(j)})_{j\in[0,n]}$

8 : encrypt shares: $\mathsf{PKE.MREnc}((ek_i)_{i\in[n]}, (f(i))_{i\in[n]}) \to (c_0, \ldots, c_n)$

9 : sign the ID i using the knowledge of r: $\mathsf{SoK.Sign}(c_0, r, i) \to \sigma_{\mathsf{DL}}$

10 : denote $\mathsf{trans}_i[1] \leftarrow (\mathsf{CR}_i^{\mathsf{deal}}, (\mathsf{cm}_j)_{j\in[n]}, (c_j)_{j\in[n]}, \sigma_{\mathsf{DL}})$

11 : sign the transcript using FS: $\mathsf{FS.Sign}(\mathsf{FS}.sk_i[1], \mathsf{trans}_i[1]) \to \sigma_i$

12 : Updat the secret key of FS: $\mathsf{FS.Update}(\mathsf{FS}.sk_i[1]) \to \mathsf{FS}.sk_i[2]$

13 : **erase** $\mathsf{FS}.sk_i[1]$, $f(X)$, $(f(i))_{i\in[0,n]}$, and encryption randomness

14 : **broadcast** $(i, \mathsf{trans}_i[1], \sigma_i[1])$

Round 2 (multicast): each P_i do:

1 : **receive:** $\{(j, \mathsf{trans}_j[1], \sigma_j[1])\}_{j\in\mathbb{D}}$, for $\mathbb{D} \subset [n]$

2 : set $\mathbb{D}_1, \mathbb{D}_2, \mathbb{D}_3, \mathbb{C} = \emptyset$

3 : // prepare dual code for verifying polynomial commitments

4 : sample an $(n-t)$-degree polynomial $q(X) \in \mathbb{Z}_p[x]$, compute

5 : $\mathsf{cm}_\tau^\perp = \dfrac{q(\tau)}{\prod_{j=0, j\neq\tau}^{n}(\tau - j)}, \forall \tau \in [0, n]$

6 : **for** $j \in \mathbb{D}$// verify each broadcast transcript as below

7 : // ignore the transcript if the FS signature is invalid

8 : **if** $\mathsf{FS.Vrfy}(\mathsf{FS}.vk_j, 1, \sigma_j[1], \mathsf{trans}_j[1]) = 0$, **then continue**

9 : // verify if it is in a good format and the sortition credential

10 : **if** parse $\mathsf{trans}_j[1] = (\mathsf{CR}_j^{\mathsf{deal}}, (\mathsf{cm}_\tau^{(j)})_{\tau\in[n]}, (c_\tau^{(j)})_{\tau\in[n]}, \sigma_{\mathsf{DL}}^{(j)})$ failed

11 : $\vee\ \mathsf{VRF.Vrfy}(rvk_j, \mathsf{rand}, \mathsf{ratio}, \text{“deal”}, \mathsf{CR}_j^{\mathsf{deal}}) = 0$

12 : //check if $(\mathsf{cm}_\tau^{(j)})_{\tau\in[n]}$ commits to a t-degree polynomial

13 : $\vee \prod_{\tau=0}^{n} (\mathsf{cm}_\tau^{(j)})^{\mathsf{cm}_\tau^\perp} \neq \mathbf{1}_{\mathbb{G}} \vee \mathsf{SoK.Vrfy}(c_0^{(j)}, \sigma_{\mathsf{DL}}^{(j)}, j) = 0$

14 : **then** $\mathbb{D}_1 = \mathbb{D}_1 \cup \{j\}$// if any fails, disqualify j immediately

15 : **elseif** $\mathsf{PKE.Dec}(dk_i, c_i^{(j)}) = sk_i^{(j)} \wedge g^{sk_i^{(j)}} \neq \mathsf{cm}_i^{(j)}$

16 : //generate a complaint, and update the complaint list

17 : **then** $\mathsf{PKE.Prove}(c_0^{(j)}, c_i^{(j)}, dk_i, sk_i^{(j)}) \to \Gamma_j, \mathbb{D}_2 = \mathbb{D}_2 \cup \{(j, \Gamma_j)\}$

18 : //otherwise, update the candidate output list

19 : **else** $\mathbb{D}_3 = \mathbb{D}_3 \cup \{j\}, \mathbb{C} = \mathbb{C} \cup \{(j, ((\mathsf{cm}_\tau^{(j)})_{\tau\in[0,n]}, sk_i^{(j)}))\}$

20 : $\mathbb{D}_2 \to \mathsf{trans}_i[2], \mathsf{FS.Sign}(\mathsf{FS}.sk_i[1], \mathsf{trans}_i[2]) \to \sigma_i[2]$

21 : **if** $\mathbb{D}_2 \neq \emptyset$, **then multicast** $(i, \mathsf{trans}_i[2], \sigma_i[2])$

FIGURE 3.1: The Any-Trust DKG construction (Part 1).

Round 3 (broadcast): each P_i do :

1 : **receive** $\{(j, \mathsf{trans}_j[2], \sigma_j[2])\}_{j\in\mathbb{R}_1}$, for $\mathbb{R}_1 \subset [n]$
2 : // determine whether it is elected for broadcasting complaint list
3 : $\mathsf{VRF.Sortition}(rvk_i, rsk_i, \mathsf{rand}, \text{“agree"}, \mathsf{ratio}) \to \mathsf{CR}_i^{\mathsf{agree}}$
4 : **if** $\mathsf{CR}_i^{\mathsf{agree}} = \perp$, // if not, update FS secret key and exit the round
5 : **then** $\mathsf{FS.Update}(\mathsf{FS}.sk_i[1]) \to \mathsf{FS}.sk_i[2]$, **exit Round 2**
6 : set DisQual, CompList $= \emptyset$// start to deduplicate complaints
7 : **for** $j \in \mathbb{R}_1$, **if** $\mathsf{FS.Vrfy}(\mathsf{FS}.vk_j, 1, \sigma_j[2], \mathsf{trans}_j[2]) = 1$
8 : **then for** $(k, \Gamma_k) \in \mathsf{trans}_j[2]$// check every complaint by P_j
9 : // if D_k has not been disqualified, verify the complaint
10 : **if** $k \notin$ DisQual, parse $\Gamma_k = (sk_j^{(k)}, \cdot)$
11 : $//c_0^{(k)}, c_j^{(k)}$ are what P_i received at line 1 of round 2
12 : **if** $\mathsf{PKE.Vrfy}(c_0^{(k)}, c_j^{(k)}, \Gamma_k) = 1 \wedge g^{sk_j^{(k)}} \neq \mathsf{cm}_j^{(k)}$
13 : // disqualify the dealer given a valid complaint
14 : **then** DisQual $=$ DisQual $\cup \{k\}$
15 : CompList $=$ CompList $\cup \{(k, \Gamma_k)\}$
16 : // stop verifying complaints from j if the complaint is invalid.
17 : **else break**
18 : $(\mathsf{CR}_i^{\mathsf{agree}}, \mathsf{CompList}) \to \mathsf{trans}_i[3]$
19 : sign $\mathsf{FS.Sign}(\mathsf{FS}.sk_i[2], \mathsf{trans}_i[3]) \to \sigma_i[3]$
20 : $\mathsf{FS.Update}(\mathsf{FS}.sk_i[2]) \to \mathsf{FS}.sk_i[3]$; **erase** $\mathsf{FS}.sk_i[2]$
21 : **if** ComList $\neq \emptyset$, **then broadcast** $(i, \mathsf{trans}_i[3], \sigma_i[3])$

At the end of Round 3: each P_i do :

1 : **receive** $\{(j, \mathsf{trans}_j[3], \sigma_j[3])\}_{j\in\mathbb{R}_2}$, for $\mathbb{R}_2 \subset [n]$
2 : set DisQual $= \emptyset$
3 : // decide the disqualified set based on broadcast message
4 : **for** $j \in \mathbb{R}_2$
5 : parse $\mathsf{trans}_j[3] = (\mathsf{CR}_j^{\mathsf{agree}}, \mathsf{CompList})$
6 : **if** $\mathsf{FS.Vrfy}(\mathsf{FS}.vk_j, 2, \sigma_j[3], \mathsf{trans}_j[3]) = 1$
7 : $\wedge\, \mathsf{VRF.Vrfy}(rvk_j, \mathsf{rand}, \mathsf{ratio}, \text{“agree”}, \mathsf{CR}_j^{\mathsf{agree}}) = 1$
8 : **then for** $(k, \Gamma_k) \in$ CompList
9 : // put a newly complained dealer in the list
10 : **if** $k \notin$ DisQual $\wedge\, \mathsf{PKE.Vrfy}(c_0^{(k)}, c_j^{(k)}, \Gamma_k) = 1$
11 : $\wedge\, g^{\Gamma_k.m} \neq \mathsf{cm}_j^{(k)}$
12 : **then** DisQual $=$ DisQual $\cup \{k\}$,
13 : set Qual $= \mathbb{D}_3 \setminus$ DisQual
14 : **output:**
15 : $pk = \prod_{j\in\mathsf{Qual}} \mathsf{cm}_0^{(j)}, sk_i = \sum_{j\in\mathsf{Qual}} sk_i^{(j)}$
16 : $pk_\tau = \prod_{j\in\mathsf{Qual}} \mathsf{cm}_\tau^{(j)}$, for every $\tau \in [n]$

FIGURE 3.2: The Any-Trust DKG construction (Part 2).

3.5.1 The Construction

The construction is based on building blocks such as PKE (along with the proof decryption system), the forward-secure signature FS, the VRF-based sortition VRF, and SoK.

Setup. Given the security parameter λ, the number of participants n, and the corruption bound t (where the adversary can corrupt up to t parties), configure the system as follows:

GROUP DESCRIPTION: Based on the security parameter λ, determine the group $\mathbb{G}$ of prime-order p and its generator g.

PKI SETUP: Every participant P_i produces three key pairs: (ek_i, dk_i) for PKE, (rvk_i, rsk_i) for VRF, and $(\mathsf{FS}.vk_i, \mathsf{FS}.sk_i[1])$ for the forward-secure digital signature scheme.

RANDOM COIN: Uniformly select a string $\mathsf{rand} \leftarrow\!\!\$\ \{0,1\}^\lambda$ that is independent of all users' public keys.

The setup also determines the value of ratio for VRF-based sortition. Given n participants executing the sortition algorithm with ratio, the chosen committee will form an any-trust group. We assume the required configurations for the underlying channels are established during this setup phase.

Protocol Details. Post-setup, all participants collaboratively run our DKG protocol, detailed in Fig.3.1. The protocol initiates with a broadcast round, transitions to a multicast round, and concludes with a final broadcast. A brief overview of each round is as follows:

–Round 1. Nodes initially determine if they're selected as dealers. If not, they refresh the secret key and exit the round (lines 1-4). Elected dealers sample a t-degree polynomial f to decide secret shares $sk_i = f(i)$, commit to $sk_0, \ldots, sk_n$, encrypt shares $sk_1, \ldots, sk_n$ using others' encryption keys (lines 6-8), and sign their ID using the knowledge of r w.r.t. $c_0 = g^r$ in the ciphertext (line 9). Dealers then sign the commitments and ciphertexts, update their signing keys, erase secret information, and broadcast the signed commitments and ciphertexts (lines 10-14).

–Round 2. Nodes receive the broadcasted messages (line 1). For every message, they authenticate the signature (line 8); if failed, they move to the next message. Otherwise, they validate its format, the VRF sortition certificate (lines 10-11), and the signature of knowledge (line 13). Moreover, they ascertain if the committed values match valid coefficients of a t-degree polynomial (lines 3-5 and 12-13). If

a transcript fails verification, the dealer is instantly disqualified (line 14). Otherwise, they check the decrypted share's validity against the commitments and, if inconsistent, generate a verifiable complaint against the dealer (lines 15-17). All complaints are multicast.

–Round 3. Nodes first verify if they are selected as senders (lines 1-4). If so, they collect and verify all complaints (using ciphertexts received from line 1 of round 2), de-duplicate them, and curate a complaint list documenting all complained dealers (lines 6-17). They then sign and broadcast this complaint list (lines 18-21).

–End of Round 3. Nodes finalize the set of disqualified dealers based on received complaint lists (lines 1-13). Following that, they create the public key (shares) and secret key share by aggregating contributions from qualified dealers (lines 14-16).

3.5.2 The Analysis

Computation complexity analysis. We primarily focus on the computationally intensive operations, particularly the group exponentiation operation EXP, and will omit inexpensive operations like multiplication operation and hash evaluation. In Round 1, a node who is elected as a dealer needs to generate a commitment (line 6), which takes $O(n)\mathsf{EXP}$, encrypt shares under n different public keys (line 7), which takes $O(n)\mathsf{EXP}$ group expo. In Round 2, to verify a transcript, each node needs to validate its commitment (line 10), which takes $O(n)\mathsf{EXP}$, and there are expected to be s transcripts to verify. If there are the maximal amount of Byzantine nodes, an honest node may need to verify $O(n)$ VRF proofs and $O(n)$ signatures (lines 6, 8), which takes $O(n)\mathsf{EXP}$, and generate $O(s)$ complaints, which takes $O(s)\mathsf{EXP}$. In Round 3, each node verifies the complaints from other nodes. Note that a node P_i stops verifying complaints from P_j upon finding an invalid complaint made by P_j, and it also stops verifying complaints against a dealer once the dealer has verifiably complained. Therefore, a node verifies at most $O(n)$ complaints, which takes $O(n)\mathsf{EXP}$. There are no group exponentiation operations in Round 4. Thus, even facing the maximal amount of Byzantine nodes, each node only needs to perform $O(sn)$ group exponentiation.

Communication complexity analysis. In Round 1, an elected node will broadcast $O(n\lambda)$-size transcript, and there are expected to be s elected node, where λ is the computational security parameter, and the sizes of a group element, a digital signature, and a VRF credential, are counted as $O(\lambda)$. So the communication cost in Round 1 is $s\mathcal{B}(n\lambda)$, where $\mathcal{B}(\ell)$ denotes the communication cost of broadcasting

ℓ bits by one sender. There will be no further communication if all nodes behave honestly. Otherwise, in Round 2, each node may need to multicast $O(s)$ complaints, which in total incurs the communication complexity of $n\mathcal{M}(s\lambda)$, where $\mathcal{M}(\ell)$ denotes the communication cost of multicasting ℓ bits by one sender. In Round 3, there are $O(s)$ elected nodes each broadcasting $O(s\lambda)$-sized complaints, which incurs the communication complexity of $s\mathcal{B}(s\lambda)$. In summary, the good case communication complexity is $s\mathcal{B}(n\lambda)$, and the adversarial-case communication complexity can be $s\mathcal{B}(n\lambda) + n\mathcal{M}(s\lambda)$.

Security analysis. The robustness ensures all participants at the end of the protocol obtain the same public key and correct shares. It follows the facts: (1) there is at least one qualified dealer, and (2) all malicious dealers will be disqualified. From the security of VRF-based sortition and the forward-secure signature, at least one honest node will be selected as a dealer, and it can successfully broadcast its valid transcript even if it later becomes corrupted, which ensures (1). From the security of the polynomial commitment, if a dealer is not complained by any node, then all honest users receive consistent shares from the dealer. Our complaint phase guarantees that every dealer who is complained by an honest node will disqualified. Therefore, we have (2).

Proving the oracle-aided algebraic simulatability is more involved. At a high level, we need to construct an efficient simulator that, on input from a sequence of group elements, produces an indistinguishable view for an adaptive adversary with the help of a DLog oracle. We follow the techniques from (Bacho and Loss, 2022) to simulate the polynomial commitments and opening shares for corrupted nodes. However, broadcasting all encrypted shares in our protocol poses additional challenges for security proof. Particularly, the simulator needs to simulate all ciphertexts without knowing the shares. It also needs to simulate the proof of decryption without using the decryption oracle of the underlying encryption scheme. As sketched in Section 3.3, we leverage the non-committing encryption and signature of knowledge to handle these challenges. Formally, we establish the following theorem.

THEOREM 1. *The Any-Trust DKG satisfies robustness and $(t, k, T_{\mathcal{A}}, T_{\mathsf{sim}})$-oracle-aided algebraic simulatability against adaptive adversaries (cf Def.3), with $n \geq 2t+1$, $k \leq n(t+1)$ and $T_{\mathsf{sim}} \leq T_{\mathcal{A}}+\mathcal{O}(snt)$, under the DDH assumption in the ROM, and assuming the security of the underlying forward-secure signature scheme. For static adversaries, it further achieves the key-expressability(cf. Def.4).*

PROOF. Under DDH assumption in ROM, our building blocks, including the VRF, the NIZK proof of knowledge, the proof of decryption, and the multi-recipient encryption are secure.

First, we argue that when up to t parties are (adaptively) compromised, each remaining honest node P_i would output the same $(pk, (pk_i)_{i\in[n]}) \in \mathbb{G}^{n+1}$. Note that the public key pk and the vector of public key shares are deterministically computed based on the set of qualified dealers, which are further determined by the information in the broadcast channel. As all honest users have the same view of the broadcast channel, our argument follows easily.

Then, we show that when up to t parties are (adaptively) compromised, the secret share $sk_i \in \mathbb{F}$ of each remaining honest node P_i satisfies $pk_i = g^{sk_i}$. Recall that $pk = \prod_{j\in\mathsf{Qual}} \mathsf{cm}_0^{(j)}$, and $pk_i = \prod_{j\in\mathsf{Qual}} \mathsf{cm}_i^{(j)}$ for $i \in [n]$. Based on line 10 of round 2, for each $j \in \mathsf{Qual}$, with an overwhelming probability, there is a polynomial $f_j(x) \in \mathbb{Z}_p[X]$ whose degree is up to t, such that $\mathsf{cm}_i^{(j)} = g^{f_j(i)}$. Therefore, define $f(X) = \sum_{j\in\mathsf{Qual}} f_j(X)$, and then it follows that $pk = g^{f(0)}$ and $pk_i = g^{f(i)}$. Meanwhile, every honest P_i should have $f(i)$. If an honest P_i does not have $f(i)$, there must exist an index $j \in \mathsf{Qual}$ such that P_i does not have $f_j(i)$. In this case, P should follow the protocol description and multicast a verifiable complaint against the dealer j to all other parties. As the verifiable complaints are posted to the broadcast channel by an any-trust group, a verifiable complaint against j must be included. Then, j should be disqualified, which contradicts our assumption that j is in Qual.

It remains to show that the set Qual is non-empty. By parameter and the security of VRF, the sampled committee contains at least one honest node with high probability. We argue this honest node will be included in Qual. Particularly, this node shall broadcast an honestly generated transcript that contains valid shares. It is easy to see that the complaints in our system are unforgeable due to the soundness of proof of decryption. Therefore, this node cannot be disqualified because of this transcript. Moreover, although this node may be corrupted after it sends out the transcript, by the forward security of the underlying signature scheme, the adversary cannot send another message with a valid signature in this round, which means the honest node cannot be disqualified because of post-corruption.

Finally, we show that, for any two sets, $\mathbb{I}_1$ and $\mathbb{I}_2$, with $t + 1$ honest participants each, a unique secret key sk can be reconstructed from their secret shares. Recall that every honest P_i should have $f(i)$ and for all $j \in \mathsf{Qual}$, polynomial $f_j(x) \in \mathbb{Z}_p[X]$ whose degree is up to t. By the uniqueness of polynomial interpolation, for any two sets, $\mathbb{I}_1$ and $\mathbb{I}_2$, with $t + 1$ honest participants each, the same $sk = f(0)$ will be reconstructed. Moreover, $pk = g^{sk}$.

Given its length, the analysis for the oracle-aided algebraic security is presented in Lemma.3, and the analysis for the key expressability is in Lemma.4. □

LEMMA 3. *The Any-Trust DKG satisfies $(t, k, T_{\mathcal{A}}, T_{\mathsf{sim}})$-oracle-aided algebraic simulatability.*

PROOF. By definition, if Π satisfies the oracled-aided algebraic simulatability, then, for every adversary $\mathcal{A}$, there will be an algebraic simulator Sim which can indistinguishably simulate the environment for $\mathcal{A}$. We proceed with the proof by presenting the code of a universal simulator Sim, which has access to the adversary $\mathcal{A}$.

On inputs a vector of group elements $\zeta = (g^{z_1}, g^{z_2}, \ldots, g^{z_k})$ for $k = n(t+1)$, Sim can simulate each phase of Π for $\mathcal{A}$ as follows.

SETUP. Sim initializes the set of corrupted parties $\mathcal{C} = \emptyset$, the set of honest parties $\mathcal{H} = \{P_i\}_{i\in[n]}$, and a table $\mathsf{RO}_{\mathsf{hist}} = \emptyset$ to record the query history of the random oracle. Then, it follows the protocol specifications to generate the public parameters and key pairs for all honest users. It answers the adversary's queries as follows.

- **Corruption queries.** When $\mathcal{A}$ asks to corrupt the party P_i, Sim first checks if $|\mathcal{C}| \leq t$. If the check fails, it ignores this query; otherwise, return the secret keys of P_i, and update the sets $\mathcal{H} = \mathcal{H}\backslash\{P_i\}$ and $\mathcal{C} = \mathcal{C} \cup \{P_i\}$.
- **Random oracle queries.** When $\mathcal{A}$ queries the random oracle with an input x, Sim checks if x has been asked before. If there is a record of (x, output_x) in $\mathsf{RO}_{\mathsf{hist}}$, return output_x; otherwise, uniformly sample output_x, add (x, output_x) to $\mathsf{RO}_{\mathsf{hist}}$, and return output_x.

Round 1. For every honest party $P_i \in \mathcal{H}$, Sim runs the *Self-Election* procedure using P_i's VRF secret key. We assume w.l.o.g. there are $s' \leq n$ honest parties being selected and denote the set by $\mathcal{H}_{\mathsf{ele}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_{s'}\}$, where each party has its credential $\mathsf{CR}_{j,\mathsf{deal}}$. Then, Sim simulates the *Commit to secret* procedure on behalf of each $\mathcal{D}_j \in \mathcal{H}_{\mathsf{ele}}$ as follows.

- Denote $\zeta_j = (\zeta_{j,0}, \zeta_{j,1}, \ldots, \zeta_{j,t})$
  $= (g^{z_{(j-1)(t+1)+1}}, g^{z_{(j-1)(t+1)+2}}, \ldots, g^{z_{j(t+1)}})$.
- Generate the commitments $\mathsf{cm}_\tau^{(j)} = \prod_{\mu\in[0,t]} \zeta_{j,\mu}^{\tau^\mu}$, for every $\tau \in [0, n]$.
- Generate the ciphertext $(c_0^{(j)}, c_1^{(j)}, \ldots, c_n^{(j)})$, where $c_0^{(j)} = g^{r_j}$ for some $r_j \leftarrow\$ \mathbb{Z}_p$, and $c_\tau^{(j)} \leftarrow\$ \{0,1\}^{\lceil \log p \rceil}$ for $\tau \in [n]$.
- Use the simulated signer algorithm of SoK to sign j w.r.t. $c_0^{(j)}$ and obtain a simulated signature of knowledge $\sigma_{\mathsf{DL}}^{(j)}$.

- Broadcast $(\mathsf{CR}_j^{\mathsf{deal}}, \mathsf{cm}_0^{(j)}, \ldots, \mathsf{cm}_n^{(j)}, c_0^{(j)}, \ldots, c_n^{(j)}, \sigma_{\mathsf{DL}}^{(j)})$ along with its forward-secure signature on it.

Sim needs to answer the queries from the adversary. For the random oracle queries and corruption queries made before broadcast, Sim can respond as it does in the SETUP phase. We discuss its strategy for answering these queries that are made after the broadcast below.

- **Corruption queries.** When $\mathcal{A}$ asks to corrupt the party P_i, Sim first checks if $|\mathcal{C}| \leq t$. If the check fails, it ignores this query; otherwise, Sim queries the oracle $\mathsf{DL}_g(\cdot)$ with $\mathsf{cm}_i^{(j)}$ for all $j \in \mathcal{H}_{\mathsf{ele}}$ with its representation $(1, i^1, \ldots, i^t)$ over ζ_j. Sim will receive $\xi_i^{(j)}$ from the oracle. Then, Sim records all $\{x_i^{(j)}\}$, which are secret shares dealt by honest dealers. Sim can obtain the shares dealt by corrupted dealers for P_i by decrypting the encrypted shares using dk_i. Finally, Sim returns the secret shares for P_i and its decryption key dk_i to $\mathcal{A}$, and updates the sets $\mathcal{H} = \mathcal{H} \backslash \{P_i\}$ and $\mathcal{C} = \mathcal{C} \cup \{P_i\}$.
- **Random oracle queries.** Before answering any random oracle queries at this stage, Sim first calculates a matrix of group elements

$$\gamma = \begin{pmatrix} \gamma_{1,1} & \gamma_{1,2} \cdots & \gamma_{1,n} \\ \gamma_{2,1} & \gamma_{2,2} \cdots & \gamma_{2,n} \\ \vdots & \vdots & \vdots \\ \gamma_{s',1} & \gamma_{s',2} \cdots & \gamma_{s',n} \end{pmatrix},$$

where each $\gamma_{j,\tau} = pk_\tau^{r_j}$ for $j \in [s']$ and $\tau \in [n]$, pk_τ is the encryption public key of P_τ, and r_j is the randomness used in encryption by Sim when simulating $\mathcal{D}_j$. Sim checks if any $\gamma_{j,\tau}$ has been asked before and **aborts** in one is in the query history. Otherwise, continue.

When $\mathcal{A}$ queries a message x, Sim performs as follows.

– If $x \neq \gamma_{j,\tau}$ for any j and τ, proceed as what it did in the Setup phase.

– If $x = \gamma_{j,\tau}$ for some j and τ, checks if P_τ has been corrupted. If it has not been corrupted, then Sim first queries the oracle $\mathsf{DL}_g(\cdot)$ with $\mathsf{cm}_\tau^{(j)}$ and its representation $(\tau^0, \tau^1, \ldots, \tau^t)$ over ζ_j. Sim will receive $\xi_{j,\tau}$ from the oracle. If it is corrupted, then $\xi_{j,\tau}$ has been recorded by Sim. Finally, it sets $\mathsf{output}_{\gamma_{j,\tau}} := c_\tau^{(j)} \oplus \xi_{j,\tau}$, records $(\gamma_{j,\tau}, \mathsf{output}_{\gamma_{j,\tau}})$ into $\mathsf{RO}_{\mathsf{hist}}$, and returns $\mathsf{output}_{\gamma_{j,\tau}}$ to $\mathcal{A}$.

Other rounds. Sim simulates the behavior of honest parties by following the specifications of the protocol, except that whenever an honest party P_i needs to decrypt an encrypted share $(c_0, c_1, \ldots, c_n)$, Sim instead performs the following procedures for decryption.

- Use the extractor of the SoK to obtain r, such that $c_0 = g^r$. Then, use r to "decrypt" the encrypted share as $sk_i = \mathsf{Hash}(ek_i^r) \oplus c_i$.

The queries from $\mathcal{A}$ are answered in the same way as Sim did in Round 1.

Let Qual_C be the set of qualified nodes that are corrupted before the **Round 1**, and $\mathsf{Qual} = \mathsf{Qual}_C \cup \mathcal{H}_{\mathsf{ele}}$. For every $j \in \mathsf{Qual}_C$, the dealer must have distributed its secret shares to honest nodes; otherwise, it will be disqualified. As Sim has always controlled more than $t+1$ honest participants, it can recover the secret key sk_j w.r.t. pk_j for every $j \in \mathsf{Qual}_C$. Therefore, Sim can output the algebraic representation for the public key as:

$$pk = \prod_{j \in \mathsf{Qual}} pk_j = g^{\sum_{j \in \mathsf{Qual}_C} sk_j} \prod_{j \in [s']} g^{z_{(j-1)(t+1)+1}}.$$

Now, we argue that the simulator specified above satisfies the requirements of oracle-aided simulatability. First, it is easy to verify that the running time of Sim is $T_{\mathcal{A}} + \mathcal{O}(snt)$.

Then, we show that $\mathsf{view}_{\mathcal{A},y,\Pi}$ and $\mathsf{view}_{\mathcal{A},y,\Pi}$ are identical, under the condition that Sim never aborts during the simulation. Specifically, from the point of $\mathcal{A}$'s view, the commitment sequence outputted by an honest party $\mathcal{D}_j$ is a commitment to the polynomial $f_j(x) = \sum_{\tau=0}^{n} z_{(j-1)(t+1)+\tau+1} x^{\tau}$. Note that the input group elements of Sim are uniformly sampled, and thus, the distribution of $f_j(x)$ is also uniform, which is identical to that in the real experiment. Moreover, in the random oracle model, the distribution of ciphertexts simulated by Sim is also identical to the real distribution. Notably, for every $pk_\tau^{r_j}$ that has been issued to the random oracle, which means that $\mathcal{A}$ can decrypt the ciphertext $c_{j,\tau}$, it follows that

$$c_\tau^{(j)} = \mathsf{Hash}(pk_\tau^{r_j}) \oplus f_j(\tau).$$

Next, we argue that Sim only aborts with a negligible probability. When Sim aborts, $\mathcal{A}$ must have queried the random oracle with some $x = \gamma_{j,\tau}$ before seeing the broadcast messages. However, $\gamma_{j,\tau} = pk_\tau^{r_j}$ is a uniformly random group element, as r_j is uniformly chosen from $\mathbb{Z}_p$ and completely independent of $\mathcal{A}$'s view before g^{r_j} is broadcasted. Therefore, $\mathcal{A}$ has negligible probability if outputting $pk_\tau^{r_j}$.

Then, we show that Sim has made at most $k-1$ queries to the $\mathsf{DL}_g(\cdot)$ oracle. Recall that Sim makes a query to $\mathsf{DL}_g(\cdot)$ whenever $\mathcal{A}$ corrupts a party or queries the random oracle with a message x which is equal to some $\gamma_{j,\tau}$. We note that under the DDH assumption, $\mathcal{A}$ can output $\gamma_{j,\tau} = pk_\tau^{r_j}$ only when $\mathcal{A}$ has corrupted the party P_τ (and thus can compute $\gamma_{j,\tau} = (g^{r_j})^{sk_\tau}$), except a negligible probability. As $\mathcal{A}$ can corrupt at most t parties, Sim will query $\mathsf{DL}_g(\cdot)$ at most ts' times, which is smaller than $k-1$.

Finally, we show the simulatability matrix L of Sim is invertible. Without loss of generality, we assume that the adversary has corrupted the parties $P_1, \ldots, P_t$, and Sim has made $s't$ queries to $\mathsf{DL}_g(\cdot)$ for simulating the queries from the adversary. For ease of analysis, we let Sim make some dummy queries such that the representations of all the queries are gonna form a square matrix of order $n(t+1)$. Specifically, Sim makes the following extra queries:

$$g^{z_{s'(t+1)+1}}, g^{z_{s'(t+1)+2}}, \ldots, g^{z_{n(t+1)}},$$

and

$$\prod_{\mu\in[0,t]} \zeta_{j,\mu}^{(t+1)^\mu}, \text{ for } j \in [1, s'-1].$$

The number of all queries by Sim is $s't + (n-s')(t+1) + s' - 1 = n(t+1) - 1$, which is still smaller than k. It is easy to verify the matrix L is invertible. □

LEMMA 4. *The Any-Trust DKG satisfies the key-expressability.*

PROOF. This proof is similar to the proof for Lemma.3, except we don't need to handle adaptive corruption queries. For any PPT adversary $\mathcal{A}$, we can construct a PPT simulator Sim that takes as input a public key $pk' \in \mathbb{G}$ and simulates the view of $\mathcal{A}$. Assume the sef of corrupted paries is $\{P_i\}_{i\in\mathsf{Corr}}$ for some $\mathsf{Corr} \subset [n]$ and $|\mathsf{Corr}| \leq t$. After sampling the any-trust group, Sim, on behalf of the honest node in the group, creates the following transcript: $\mathsf{cm}_0 = pk'$, $c_0 = g^r$ for some $r \leftarrow\!\$\ \mathbb{Z}_p$; For $i \in \mathsf{Corr}$, $sk_i \leftarrow\!\$\ \mathbb{Z}_p$, $\mathsf{cm}_i = g^{sk_i}$, and $c_i = \mathsf{Hash}(ek_i^r) \oplus sk_i$. For $i \notin \mathsf{Corr}$, cm_i are created by Lagrange interpolation in the exponent, while c_i are randomly sampled. This transcript is indistinguishable from an honestly generated one in the view of $\mathcal{A}$ and cannot be disqualified. For every other transcript with $\mathsf{cm}^{(j)} = pk^{(j)}$ which is eventually included in the qualified set, Sim can know the secret key $sk^{(j)}$ by reconstructing it from shares held by honest nodes. Note that the final public key is in the form of $pk' \cdot \prod pk^{(j)}$, and the simulator can express it by setting $\alpha = 1$, $sk'' = \sum sk^{(j)}$. □

Committee Size. Recall that our construction employs a VRF-based sortition to decide the committee, in which each node can be independently elected a committee member with a probability `ratio`. Assume a network of n nodes while at least h of them remains honest. Such a sortition process will produce a committee of the expected size of $s = \texttt{ratio} \cdot n$. Then the probability $\texttt{p}$ that at least one honest node being elected can be expressed as follows:

$$\texttt{p} = 1 - (1 - \frac{s}{n})^h. \tag{3.1}$$

We compute the expected committee sizes necessary to ensure different values of $\texttt{p}$ in networks with varying ratios of honest parties, as depicted in Table 3.2. For example, assuming over 51% participants of the whole network are honest, we can set the expected committee size as 38, which ensures the resulting committee contains at least one honest node with the probability of at least $1 - 5 \times 10^{-9}$. These findings are applicable to networks of any size, although for networks with $n \leq 10^4$, a slightly smaller committee size may be achievable.

PR \ HR	51%	67%	80%
$1 - 5 \times 10^{-9}$	38	29	24
$1 - 2^{-30}$	41	32	26
$1 - 2^{-40}$	55	42	35

TABLE 3.2: Expected committee sizes for different probability guarantees (PR) under different honest-party ratio (HR).

3.6 Sub-ID Allocation for the Weighted Setting

In this section, we present a simple yet effective sub-ID allocation mechanism that enables us to apply a conventional distributed protocol like our Any-Trust DKG in the weighted setting. Compared with the straightforward sub-ID allocation mechanism, ours dramatically reduces the number of required sub-IDs.

Qualified allocation. The traditional sub-ID allocation method ensures that the proportion of sub-IDs held by honest participants is equal to the proportion of an honest participant's weights, which we call a *perfect* allocation. However, we notice a gap between the usual assumption on the honest participant's weight ratio, which is typically assumed to be more than 2/3 due to other components of the system, and the honest ratio needed in threshold cryptography, which is usually just above 1/2. Therefore, we consider a lossy-yet-qualified allocation, which guarantees that more than half of the sub-IDs will be issued to honest participants if they have more than 2/3 of the weights[10]. Formally, we have the following definition.

DEFINITION 5 (Qualifed Allocation). *Let $W = (w_1, \ldots, w_n)$ be a sequence of positive integers. Let A and B be any partition of the index set $[n]$ (i.e., $A \cup B = [n]$ and $A \cap B = \emptyset$). We say a function* $\mathsf{AllocateSubID}(w_1, \ldots, w_n) \to (d_1, \ldots, d_n)$*, where d_i's are non-negative integers, is a* ***qualified allocation*** *for W, if for every (A, B) s.t.*

$$\sum_{i \in A} w_i > 2 \cdot \sum_{i \in B} w_i, \text{ it holds that } \sum_{i \in A} d_i > \sum_{i \in B} d_i.$$

While such a qualified allocation suffices for security, we need to find an allocation method that minimizes the number of all sub-IDs, *i.e.*, $\sum_j d_j$ is as small as possible.

Our method. We start by observing that dividing each w_i by the greatest common division (GCD) leaves the fraction for any index subset A unchanged. This realization provides a straightforward allocation approach: $d_i = \frac{w_i}{\mathsf{gcd}}$. However, if the GCD is small, the total sub-IDs can be vast.

A viable approach is to modify each w_i to w'_i so the new sequence $W' = (w'_1, \ldots, w'_n)$ has a substantial GCD. This adjustment might increase some subsets' proportions while reducing others, potentially

[10] While our discussion primarily centers on the gap between 2/3 and 1/2, the underlying concept can be effortlessly extended to address other thresholds or scenarios.

strengthening the adversary. Still, we determine that any increased power for the adversary remains capped if we limit the total adjustments.

Specifically, we call an adjustment t-bounded for $(w_1, \ldots, w_n)$, if the adjusted values $(w'_1, \ldots, w'_n)$ satisfies $\sum_{i\in[n]} |w_i - w'_i| \leq t$. Then, if $\sum_{i\in[n]} w_i \geq 3t + 1$, it ensures that for any partition (A, B) over $[n]$ satisfying $\sum_{i\in A} w_i > 2 \cdot \sum_{i\in B} w_i$, it holds that $\sum_{i\in A} w'_i > \sum_{i\in B} w'_i$, given the inequality:

$$\sum_{i\in A} w'_i - \sum_{i\in B} w'_i \geq \sum_{i\in A}(w_i - \Delta_i) - \sum_{i\in B}(w_i + \Delta_i) \geq 1 \tag{3.2}$$

Here, $\Delta_i = |w_i - w'_i|$. Following the adjustment, Sub-IDs, d_i, are derived by dividing w'_i by this higher GCD.

Given our objective to minimize $\sum_j d_j$, the goal is to enhance the GCD. To achieve this, we consider a target gcd, defining an adjustment function $f_{\mathsf{gcd}}(w_i) \to w'_i$ as:

$$w'_i = \begin{cases} w_i - (w_i \bmod \mathsf{gcd}), \text{ if } w_i \bmod \mathsf{gcd} < \mathsf{gcd}/2, \\ w_i + \mathsf{gcd} - (w_i \bmod \mathsf{gcd}), \text{ otherwise.} \end{cases} \tag{3.3}$$

Starting with $\mathsf{gcd} = 1$, we increase it until f_{gcd} is no longer a t-bounded adjustment for W. Utilizing binary search can quickly find a very large gcd. While variations in $(w_1, \ldots, w_n)$ may suggest larger gcd', our found gcd is practically near-optimal. The allocation algorithm is detailed below.

$\mathsf{AllocateSubID}(w_1, \ldots, w_n)$

binary search the largest gcd from 0 to $\max_i w_i$

s.t. f_{gcd} is t-bounded for $(w_1, \ldots, w_n)$

output $(d_i = \dfrac{f_{\mathsf{gcd}}(w_i)}{\mathsf{gcd}})_{i\in[n]}$

Efficiency and effectiveness. Note that our $\mathsf{AllocateSubID}$ is only supposed to find a t-bounded f_{gcd} for its input $(w_1, \ldots, w_n)$. So we can efficiently check whether $\sum_{i\in[n]} |f_{\mathsf{gcd}}(w_i) - w_i| \leq t$. Thus, the time-cost of $\mathsf{AllocateSubID}$ is $O(n \log n)$. Meanwhile, using binary search is effective since there is a general trend that the larger the gcd is, the larger adjustment is needed. It can give us a t-bounded f_{gcd} for the input with a large gcd (not necessarily optimal).

TABLE 3.3: Comparison with Swiper/Dora.

Systems	# Parties	#Total Weights	Swiper/Dora (2023)	Ours
Aptos (Blockchain)	104	8.4708×10^8	27	34
Tezos (Tezos)	382	6.7579×10^8	75	77
Filecoin (Filecoin)	3700	2.5242×10^{19}	1895	1688
Algorand(Chen and Micali, 2019)	42920	9.7223×10^9	373	301

Our sub-ID allocation is a qualified allocation as per Def.5, since f_{gcd} is t-bounded for $(w_1, \ldots, w_n)$. Moreover, for a set of n validators with an arbitrary power distribution, our method only issues at most $2n$ sub-IDs.

LEMMA 5. *Given any sequence* $W = (w_i)_{i\in[n]}$ *with* $\sum_{i\in[n]} w_i = 3t + 1$ *for some integer* t*, let* $(d_1, \ldots, d_n)$ *be the output of our* AllocateSubID*. It follows that* $\sum_{i\in[n]} d_i \leq \frac{4t+1}{\lfloor 2t/n \rfloor}$*, which is around* $2n$ *when* $n \ll t$*.*

PROOF. Let $\mathsf{gcd} = \lfloor 2t/n \rfloor$. It is easy to see that $(w'_1, \ldots, w'_n)$ outputted by $f_{\mathsf{gcd}}(w_1, \ldots, w_n)$ and $(w_1, \ldots, w_n)$ are bounded by $n \cdot \lfloor 2t/n \rfloor/2 = t$. Let $d_i = \frac{w'_i}{\mathsf{gcd}}$. It holds that $\sum_{i\in[n]} d_i = \frac{\sum_{i\in n} w'_i}{\lfloor 2t/n \rfloor} \leq \frac{4t+1}{\lfloor 2t/n \rfloor} \approx 2n$. □

Comparison with Swiper/Dora (de Souza and Tonkikh, 2023). A concurrent work, Swiper/Dora (de Souza and Tonkikh, 2023), also addresses the imparity between conventional threshold cryptography and the weighted setting. In Table 3.3, we compare our method and theirs for validator sets across various PoS systems. The comparison is under the same condition, *i.e.*, ensuring more than 1/2 sub-IDs are allocated to honest parties with more than 2/3 weights. The result shows our method issues fewer sub-IDs to large sets of validators, such as Algorand and Fielcoin[11].

[11] We note a recent version of Swiper/Dora(Tonkikh and de Souza, 2024) has further reduced the number of sub-IDs.

3.7 Practical Extended Broadcast Channels

In this section, we introduce a practical extension to the blockchain-based broadcast channel. Although it is folklore knowledge that, theoretically, one may throw all messages into the ledger to facilitate a broadcast, this may incur prohibitive costs in practice, as on-chain resources are generally very expensive. Instead, our extension empowers users to broadcast a message of arbitrary length while inscribing only a *constant-size* storage on the blockchain. Crucially, our enhanced broadcast channel retains its original simplicity and modularity. Users can conveniently interact with it using the APIs of well-established infrastructures, including both blockchains and a data dispersal network (DDN) like IPFS (Trautwein et al., 2022).

3.7.1 Building Blocks

We formalize our building blocks. For simplicity, we model a blockchain as a public bulletin board (PBB) that allows users to post and retrieve data.

Public Bulletin Board. We follow the model of PBB presented in (Kidron and Lindell, 2011) and extend it to support *keyword*-based retrieval. A user can interact with PBB by using the following queries:

- $\mathsf{getCounter}() \to t$. It returns the current counter value t.
- $\mathsf{post}(\mathsf{kw}, v) \to t$. On receiving value v along with a keyword kw, it increments the counter value by 1 to t, stores (t, kw, v), and responses t.
- $\mathsf{retrieve}(t_{\mathsf{start}}, t_{\mathsf{end}}, \mathsf{kw}) \to \{(v_i, t_i)\}$. It returns all pairs of (v_i, t_i), such that $t_{\mathsf{start}} \leq t_i \leq t_{\mathsf{end}}$ and kw is their keyword.

We care about the storage cost of PBB. For a user posting ℓ bits to the PBB, we denote the cost as $\mathcal{PB}(\ell)$. We assume that a PBB satisfies the *validity* and *agreement*.

VALIDITY. Assume an honest user posted (v, kw) to the PBB and received t. Then, every honest user who retrieves with $(t_{\mathsf{start}}, t_{\mathsf{end}}, \mathsf{kw}')$ such that $t_{\mathsf{start}} \leq t \leq t_{\mathsf{end}}$ and $\mathsf{kw}' = \mathsf{kw}$ will receive a sequence of value/counter pairs containing (v, t).

AGREEMENT. If an honest user retrieving with $(t_{\mathsf{start}}, t_{\mathsf{end}}, \mathsf{kw})$ when $\mathsf{getCounter}() \geq t_{\mathsf{end}}$ receives a sequence of value/counter pairs S, then every honest user retrieving with $(t_{\mathsf{start}}, t_{\mathsf{end}}, \mathsf{kw})$ will receive the same S.

It is rather straightforward to use PBB as a broadcast channel by simply posting a broadcast message into the PBB. The authenticity can be established with standard digital signatures in the PKI model.

Data Dispersal Network. A data dispersal network (DDN) provides a platform where one can provision a data block for others who may need it. Compared with standard multicast, which is also for data dissemination, DDN saves communication costs when there are multiple nodes providing the same data block. Assuming there are m receivers out of n potential receivers, and there are k data providers for a data block of ℓ bits. Through multicast, every sender needs to send their data to every potential receiver, incurring the communication cost of $k \cdot \mathcal{M}(\ell) = O(kn\ell)$. In contrast, through DDN, each receiver receives exactly one copy of data, incurring a total communication cost of $O(m\ell)$, which is smaller than $\mathcal{M}(\ell)$.

In principle, we can either use an erasure-code-based information dispersal protocol (Reed and Solomon, 1960) or practical infrastructure like IPFS (Trautwein et al., 2022) to instantiate a DDN. In this work, we focus on the IPFS-based instantiation as it becomes easier to implement (given IPFS already exists) and model it with the following two queries, which might be specific to the instantiation.

- register: on receiving a node ID nid and a block ID bid (which is the hash value of the data), it checks whether bid has been registered. If not, add a new entry $(\mathsf{bid}, \mathsf{nid})$; otherwise, it appends nid to the existing entry with bid.
- retrieve: on receiving a block ID bid, it returns the associated datablock v, by orchestrating the data flow from candidate providers.

We assume as long as there is an honest data provider who has registered bid and remains active, everyone can retrieve the data block with bid. We denote the cost of registering for s data blocks as $\mathcal{R}(s)$.

3.7.2 Our Extended Broadcast Channel

A strawman and our intuition. A naive approach to broadcasting a sizeable data block involves posting its ID, denoted as bid, on the PBB while simultaneously registering both bid and the sender's ID (nid) on the DDN. However, this methodology cannot guarantee agreement. Specifically, a malicious sender has the capability to selectively deny some retrieval requests on the DDN. Moreover, an adaptive adversary, upon observing the bid on the PBB, can corrupt the sender, subsequently rendering the data inaccessible on the DDN.

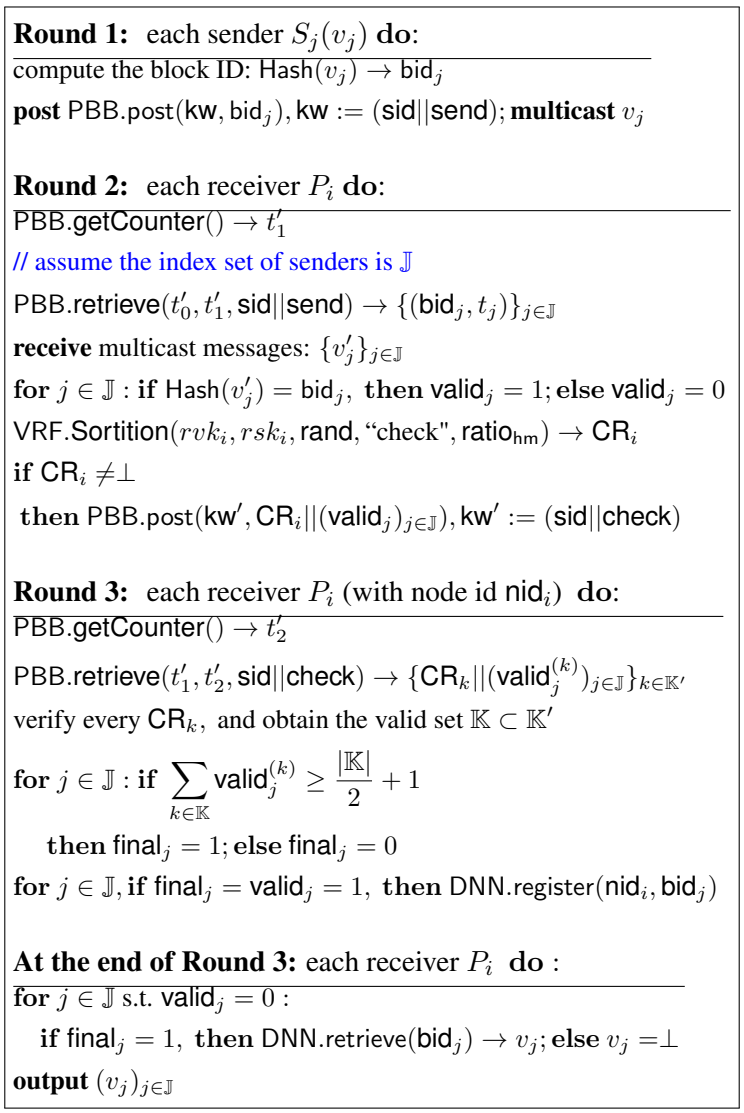


FIGURE 3.3: Our extended broadcast channel.

To address these security vulnerabilities, we suggest using DDN and PBB together in a smarter way. Recognizing the potential threat of adaptive corruption, the sender directly multicasts the data block to all receivers while posting the block ID bid into the PBB. Importantly, this process does not induce additional overhead compared with the DDN-based dissemination since there is only one provider, and all receivers will require the data block. For agreement, an honest majority committee is sampled, which subsequently votes to validate the accessibility of the data block against the advertised bid. In scenarios where the majority of the committee members vouch for the data block's availability, all receivers who successfully received the data block are then prompted to register on the DDN. This ensures that any receivers who fail to receive the data through multicast will be able to retrieve it from DDN.

Protocol details. We assume the PKI setup, as well as the setup for the VRF-based sortition, such that everyone in the group gets to know others' verification keys w.r.t. a digital signature scheme and VRF. A ratio $\mathsf{ratio}_{\mathsf{hm}}$ is also determined in the setup, which ensures a high probability that the sampled committee will contain an honest majority. Moreover, we assume every message has been signed by the sender. Besides that, a session id sid and an initial counter t'_0 are supposed to be known to everyone in the group and can be used to retrieve related messages from the PBB. We w.l.o.g. describe our protocols in a batch manner, *i.e.*, there can be multiple senders, as this is the situation of our DKG protocol. We elucidate our design in Fig.3.3.

Complexity analysis. Assume there are s senders, and each of them broadcasts a message of ℓ bits to the group with n nodes. The communication cost of our extended broadcast channel is

$$s \cdot \mathcal{B}(\ell) = s\mathcal{PB}(\lambda) + \mathcal{O}(sn\ell) + c\mathcal{PB}(\lambda + s) + n\mathcal{R}(s),$$

where λ denotes the security parameter (*i.e.*, the size of a digest, the output length of a VRF, *e.t.c.*), $s\mathcal{PB}(\lambda)$ is caused by that s senders post their digests into the PBB, $\mathcal{O}(sn\ell)$ is caused by that the senders multicast their message and the receivers retrieve from a DDN, $c\mathcal{PB}(\lambda + s)$ is caused by the selected committee members vote for the broadcast status, and $n\mathcal{R}(s)$ is caused by that the honest parties register to the DDN. Now, the on-chain storage cost is *independent of* ℓ.

Security analysis. We establish the security of our extended broadcast channel in the following lemma.

LEMMA 6. *Assume the underlying PBB satisfies validity and agreement, and the DDN guarantees the data block can be retrieved when there is an honest and active provider. The protocol in Fig.3.3 satisfies the validity and agreement.*

PROOF. Our construction satisfies both the validity and agreement. Regarding validity, in our protocol, when the sender is honest, every honest receiver can receive the message v from the multicast channel and retrieve the digest from the PBB. Then, in round 2, selected honest committee members would vote for this broadcast (by setting and posting $\mathsf{valid} = 1$), such that the final status of this broadcast will be 1, and honest nodes can decide on v.

Regarding agreement, note that whether $v = \perp$ is determined by the votes on PBB. Therefore, if an honest receiver decides on $v = \perp$, everyone will do the same thing. The potential chance causing disagreement is that when an honest receiver decides on $v \neq \perp$, some receiver cannot successfully retrieve v from the DDN. Below, we show that this case is unlikely to happen.

Assume that the adversary is allowed to corrupt at most t participants among all the n participants, and the VRF-based sortition at round 2 will yield a committee $\mathcal{C}$ of $c = 2t' + 1$ participants. As the parameter is configured to guarantee the honest majority of the elected committee, it implies that, for any subgroup A whose size is not greater than t, the following probability is very small:

$$\Pr[|Z| \geq t' + 1 : Z = A \cap \mathcal{C}].$$

Now, we consider the group B of nodes that are, before the election, either corrupted nodes or honest nodes that have received v. In the case that there are $t' + 1$ votes endorsing the availability of v, it holds that $|B \cap \mathcal{C}| \geq t' + 1$, which implies the probability $\Pr[|B| \leq t]$ is small. Therefore, the adversary cannot corrupt all nodes in B even after knowing the committee $\mathcal{C}$. It follows that there is always at least one honest node that has received v and provisioned it to the DDN, such that everyone can retrieve the data from the DDN and can agree on the value v. □

3.8 Application to All-hands Checkpointing into Bitcoin

In this section, we delineate how our DKG yields the first realization of the checkpointing blueprint Pikachu of Filecoin (Azouvi and Vukolic, 2022) that involves all validators in the whole blockchain network, e.g., Filecoin, that has 3700 of them, with various mining power.

3.8.1 Realizing the Bitcoin Checkpointing Pikachu with Any-Trust DKG

We review the checkpointing blueprint Pikachu in Appendix A.1. At a high level, all validators need to run a DKG for Schnorr signature every epoch, and the resulting public keys will be used as Bitcoin addresses [12]. A Bitcoin transaction that embeds the digest of the PoS chain at epoch $i-1$ and transfers assets from the address at epoch $i-1$ to epoch i will serve as a checkpoint for epoch $i-1$. All validators jointly run the threshold Schnorr signing protocol to create such checkpointing transactions.

Pikachu only gave a proof-of-concept prototype with 21 participants due to the inefficiency of their underlying DKG scheme. Meanwhile, as they instantiated the threshold Schnorr signature with FROST (Komlo and Goldberg, 2020), which relies on a coordinator, there may be a single point of failure. In the following, we demonstrate how our Any-Trust DKG can realize the blueprint efficiently and securely.

Sub-ID allocation. At each epoch i, the current validators of the blockchain locally run the deterministic sub-ID allocation algorithm on a publicly agreed power distribution, and then they obtain the same sub-ID allocation outcome. A validator with m sub-IDs will participate in further protocols as m individuals.

Our optimized sub-ID allocation algorithm in Section 3.6 issues fewer sub-IDs to validators than the straightforward approach. We consider a snapshot of Filecoin's validator distribution[13], which has 3700 validators with a total mining power of around 25 EB, while the power unit is 32KB. The standard method may issue around 674 trillion sub-IDs. In contrast, our method identifies that 13 PB can be a good GCD, and only 1688 sub-IDs need to be issued, significantly reducing the scale of the problem.

Apply Any-Trust DKG. The validators with 1688 sub-IDs will act like 1688 participants to execute the DKG protocol to generate a public key for Schnorr signature and share the secret keys. We set $\mathsf{ratio}_{\mathsf{at}} = 38/1688$, guaranteeing the committee has at least one good node with a probability of $1 - 5 \times 10^{-9}$. Then, the validators can run our Any-Trust DKG which incurs only around 3 MB of data that needs

[12] Bitcoin has supported Schnorr signature since its TAPROOT update.

[13] `https://filfox.info/en/ranks/power`

TABLE 3.4: Checkpointing cost per annum. in USD.

#Parties	2^7(Cosmos)	2^{10}(Polkadot)	2^{12}(Filecoin)
Babylon	1510826.9	2266245.6	6043306.5
Ours	26048.8		

*Based on the Bitcoin price on Mar. 31, 2024: 0.000708 USD per Satoshi.

to be broadcasted. It takes each node a few seconds to finish computation, even facing the maximum number of complaints.

Checkpointing with non-interactive threshold Schnorr signature. At epoch i, the validators of epoch $i-1$ use their shared keys to sign the checkpointing Bitcoin transaction. Note that no matter how many nodes try to post the signed transaction to the Bitcoin, there will be only one transaction appearing on the chain. To sign this transaction, we adopt the GJKR protocol(Gennaro et al., 2007), which does not require a coordinator and is thus free of single-point failures. The GJKR protocol involves a DKG as its subroutine for generating the nonce and follows a non-interactive phase where every signer can locally compute its signature share (or called a partial signature). GJKR was believed to be inadequate for large-scale deployment due to its DKG subroutine, which, however, is no longer a bottleneck with our any-trust DKG. Since our DKG is key-expressable (cf. Def.4 and (Gurkan et al., 2021)), the static security of the resulting scheme directly follows the recent result in (Shoup, 2023). Note that despite recent advancements (Crites et al., 2023), achieving adaptively secure and robust threshold Schnorr without using a coordinator remains a significant open problem. We leave it as future work to analyze the adaptive security of this scheme, namely GJKR with an oracle-aided algebraic simulatable DKG.

3.8.2 Comparison with Babylon Checkpointing

Overview of Babylon. Babylon (Tas et al., 2023) is a recently proposed checkpointing scheme that does not use DKG and threshold signature. Instead, it employs the following approach: (1) All validators sign the digest of the PoS block to be checkpointed. (2) One honest validator collects and aggregates enough signatures (using the BLS aggregatable signature scheme (Boneh et al., 2001)) and publishes a Bitcoin transaction with the OP_RETURN code. This transaction contains the epoch number, the digest, the aggregated signature, and a bit vector that indicates the public keys involved.

Comparison of Bitcoin Transaction Fees. It's important to note that for n validators, at least n bits are needed to encode the public key list. A Bitcoin transaction allows 80 bytes with OP_RETURN, which means the number of Bitcoin transactions per checkpoint grows linearly with the number of validators. Particularly, the epoch number, the block digest, and the aggregated signature together take 88 bytes;

the bit-vector requires n bits. Therefore, the number of Bitcoin transactions for a Babylon checkpoint can be calculated as $\#\mathsf{Bitcoin\ Tx}_{\mathsf{Babylon}} = 1 + \lceil \frac{n+64}{640} \rceil$.

Moreover, since it assumes an honest validator to create the checkpointing transaction, it might have a single point of failure. This issue can be resolved by sampling a committee that includes at least one honest validator for creating Bitcoin transactions. For the more secure version of Babylon, the number of Bitcoin transactions per checkpoint would increase by a factor of the any-trust committee size κ, i.e., $\#\mathsf{Bitcoin\ Tx}_{\mathsf{secure-Babylon}} = \kappa + \kappa \cdot \lceil \frac{n+64}{640} \rceil$. For $\kappa = 29$ (see Table 3.2) and $n = 2^{12}$, we have $\#\mathsf{Bitcoin\ Tx}_{\mathsf{Babylon}} = 8$, while $\#\mathsf{Bitcoin\ Tx}_{\mathsf{secure-Babylon}} = 232$.

In comparison, our approach (Pikachu) only requires 1 Bitcoin transaction for each checkpoint, since the transaction is uniquely created via threshold signing, and Bitcoin will only accept one transaction no matter how many validators try to publish it. It is naturally free of single points of failure.

We compare the Bitcoin transaction fees for checkpointing per annum in Table 3.4, where the cost of Babylon is for its secure version. Following (Tas et al., 2023), we consider the checkpoint transactions to be created hourly. We assume, without loss of generality, each Bitcoin transaction has 300 bytes, the transaction fee is 14 Satoshi per byte (such that the transaction can be confirmed within six blocks as per [14]), and the price of a Satoshi is 0.000708 USD [15]. We evaluate the cost for PoS chains with different numbers of validators: 2^7 validators for small-scale PoS chains (like the ones in Cosmos (Cosmos, 2016)), 2^{10}validators for moderate-scale chains (like Polkadot(Polkadot, 2016)), and 2^{12} validators for relatively large-scale chains (like Filecoin (Filecoin, 2017)).

[14] `https://btc.network/estimate`
[15] updated on Mar. 31, 2024, from `https://coincodex.com/crypto/satoshi-sats/`

3.9 Implementation and Evaluation

We implemented our proposed DKG and present the experimental results in this section.

Implementation. We implemented our protocol in Java 8, comprising approximately 1500 lines of code. To facilitate Elliptic Curve operations and communication, we utilized the open-source Java library `mpc4j`[16] and the `Bouncy Castle` library[17]. Given our protocol's primary application in creating checkpoints on Bitcoin, we opted for the secp256k1 curve and SHA-256 for cryptographic operations. Our implementation includes components such as VRF and multi-recipient encryption but does not employ the broadcast extension trick in Section 3.7. It is essential to note that this implementation serves as a proof-of-concept, demonstrating the practicality of our protocol for large-scale deployment, even under the presence of the maximal number of Byzantine nodes. We do not implement forward-secure signatures; however, their cost is marginal and independent of the scale. Whenever possible, we set the expected size s of an any-trust group to 38, which ensures that the committee qualifies with a probability of $1-5\times10^{-9}$, as in Table 3.2. For small-scale tests like $n = 16$ and 32, we set $s = n/2+1$.

Evaluation Setup. We evaluate our Any-Trust DKG implementation with a varying number of nodes: 16, 32, 64, 128, and 256. Each node is encapsulated within an individual Amazon Web Services (AWS) **t3a.medium** EC2 virtual machine (VM). Each VM has 2 vCPUs and 4 GiB RAM and runs in Amazon Linux 2023 AMI 2023.4.20240416.0 x86_64 HVM kernel-6.1. All nodes are placed in the same AWS region and are connected pair-wise; for example, every two nodes are directly connected. Since the network delay within the same AWS region is almost negligible, we simulate a more realistic delay by employing the Linux command `tc` (traffic control) to introduce an artificial delay of 100 ms for all traffic.

Implementation Remarks. We set up an additional node to simulate a blockchain, which serves as the broadcast channel in our implementation. The blockchain node is directly connected to all other nodes. In Round 1 and Round 3 of our DKG, whenever a node needs to broadcast a message, it sends the message directly to the blockchain node. The blockchain node then relays all received messages in the round to every node in the network. As our protocol assumes network synchrony and proceeds round by round, we need to specify the time window for each round.

[16] `https://github.com/alibaba-edu/mpc4j`
[17] `https://www.bouncycastle.org/`

In practice, the time window setting for Round 1 and 3 can vary depending on the blockchain. For simplicity, we artificially configure the time window to be 30 seconds: the blockchain node receives messages in the first 20 seconds and then relays the messages. All nodes stop receiving current-round messages at 30 seconds and move to the next round. Given such a configuration, a 60-second broadcast running time is inherent to our experiments, and our experiments are more concerned about the running time incurred by Round 2 and the computation cost in Round 1, Round 3 and at the end of Round 3.

We evaluate the performance of our DKG in both the good-case and bad-case scenarios. In the good cases, all nodes are honest. In the bad cases, we set all nodes whose node ID is smaller than $n/2$ to be corrupted. A corrupted node, if elected as a dealer in Round 1, will broadcast malformed ciphertexts to all nodes, causing n complaints against it in Round 2. Given a fixed number of nodes and good or bad cases, each experiment configuration is repeated eight times.

In reality, the timeout parameter for the **receive{}** at the start of Round 3 should be calibrated based on the communication cost in bad cases. In our implementation, the calibration is implicit: the timeout parameter is set to a sufficiently large value, while the actual communication cost in bad cases is measured independently.

Adjusted Running Time. The total running time of the entire Any-Trust DKG protocol can be defined by the time difference between the moment when the communication network is established and when a node finishes computing the shared public key and its secret share. However, this measurement will always incorporate the 60-second broadcast time, which may vary in different settings. Hence, an adjusted running time is measured by subtracting the 60-second broadcast cost from the running time of the whole Any-Trust DKG protocol. Observe that the adjusted running time consists only two components: the communication cost incurred by the **multicast** in Round 2, and all computation costs throughout this protocol.

We take the **maximum** adjusted running time across all nodes, and all repeats, to represent the end-to-end running time of our protocol.

The adjusted running time of bad cases are shown in Fig.3.4. In additional, a breakdown by the **multicast** communication cost and the computation cost is shown within the stacked bar chart. Note that the good cases should always perform better than the bad cases, hence, we only represent the bad cases to demonstrate the worst-cast scenario.

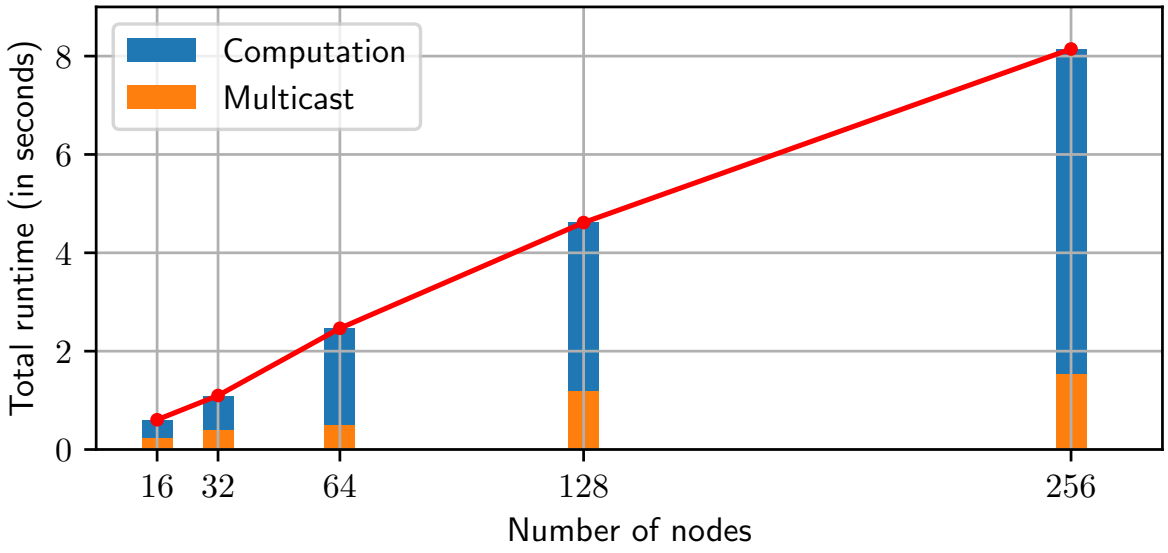


FIGURE 3.4: Worst-case adjusted running time of bad instances.

Our DKG protocol only requires a few seconds to finish the multicast round and all computation tasks, in additional to the omitted 60-second broadcast cost.

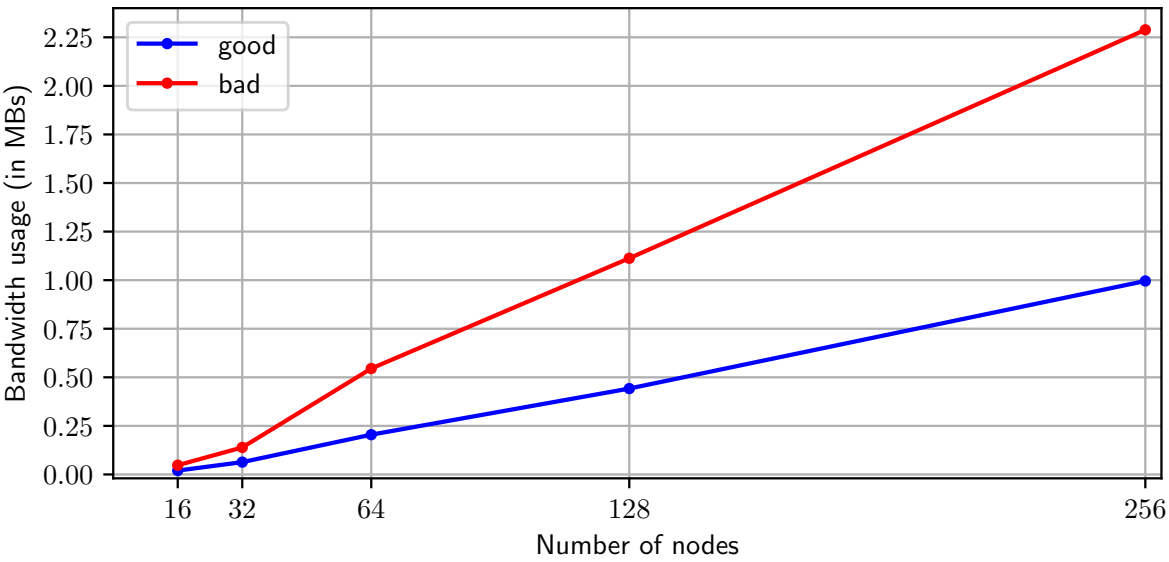


FIGURE 3.5: Worst-case bandwidth usage, the amount of data transfers inbound to and outbound from a node during the protocol execution.

Bandwidth Usage. We record the inbound and outbound bandwidth of each node in Megabytes (10^6 bytes) and demonstrate the **maximum** bandwidth usage of all nodes, and all repeats in Fig.3.5. The key observation is that the bandwidth grows linearly depending on the size of the group.

At first glance at the results, some non-linearity may be noticed. However, this is mainly caused by (a) a lower sortition ratio for $n = 16$ and $n = 32$ and (b) the randomness in the sortition results in the Any-Trust DKG protocol. Specifically, the protocol has no deterministic control over the actual number of parties being elected as dealers, meaning fluctuations will be observed in bandwidth usage, as the actual number of dealer may vary.

3.9.1 Performance Analysis on Large Scale

While our end-to-end implementation demonstrates that our protocol remains practical when $n = 2^8$, we further tested the computation time of our protocol and estimated the communication cost on larger scales ranging from $n = 2^9$ to $n = 2^{15}$, this range covers the sizes of most PoS chain validators.

Broadacst cost. We calculate the total number of bits to be sent via the broadcast channel. We compare our protocol and KZG in terms of it, ranging from $n = 2^9$ to $n = 2^{15}$, considering both the good case and the bad case with the maximal number of complaints. Note that in KZG, a share along with the proof for validating has the size of 224 Bytes; in the bad case, there are $n^2/2$ shares (with their proofs) to be broadcasted for public verification.

As shown in Figure 3.6, for our protocol, the costs in the good case and the worst case are very close and grow steadily. For $n = 2^9$, the cost is around 1.05 MB, while for $n = 2^{15}$, the cost is approximately 61.1 MB. In contrast, while the good-case KZG has very low broadcast costs, its worst-case costs grow quadratically and would require over 120 GB when $n = 2^{15}$.

Computation time. We conducted tests to measure the computation time for generating a secret-sharing transcript (Deal) and reaching an agreement on a qualified set (Verify) in both good case and bad case on AWS c5a.large (AMD EYPC 7002 CPU with 2 cores and 4 GB RAM). We compared our results with KZG, utilizing the reported findings from (Zhang et al., 2022) for the good case while estimating the worst-case scenario by assuming that $n^2/2$ shares need to be verified (each share verification takes 1.3 ms). As illustrated in Fig.3.7, in the good case, our protocol's performance is comparable to or even better than KZG, although their programming language (C++) and environment (AWS c5a.24xlarge, AMD EYPC 7002 CPU with 96 cores, and 187 GB RAM) are supposed to be superior to ours. However, in the worst-case scenario, our protocol remains efficient while KZG becomes infeasible.

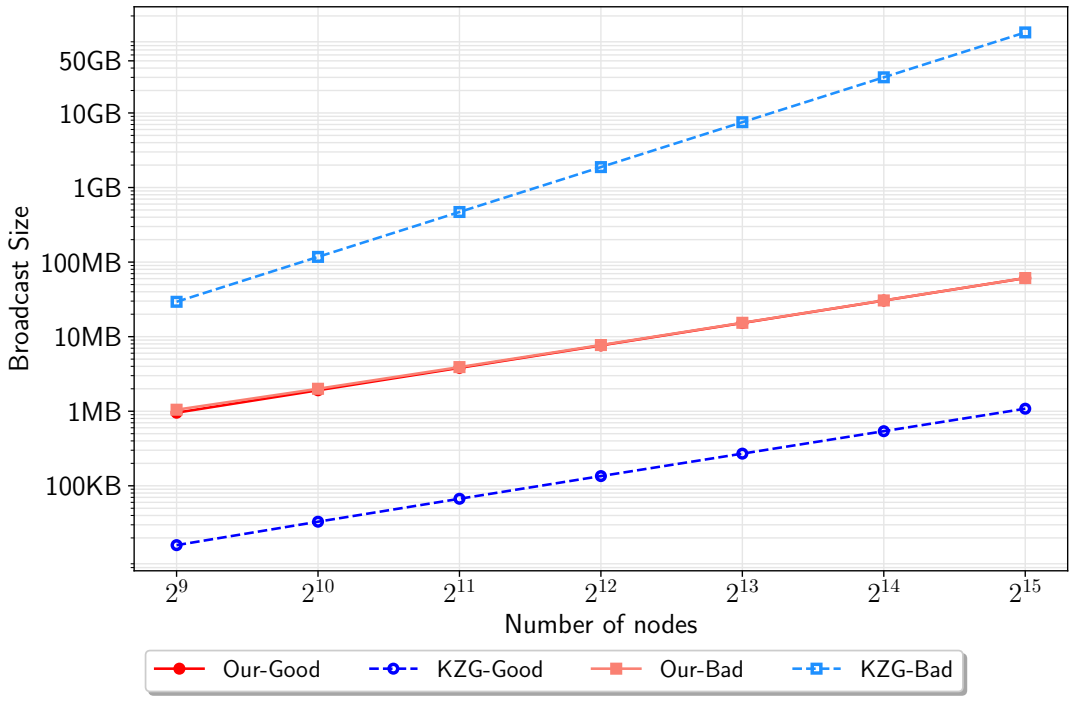


FIGURE 3.6: Broadcast channel overhead.

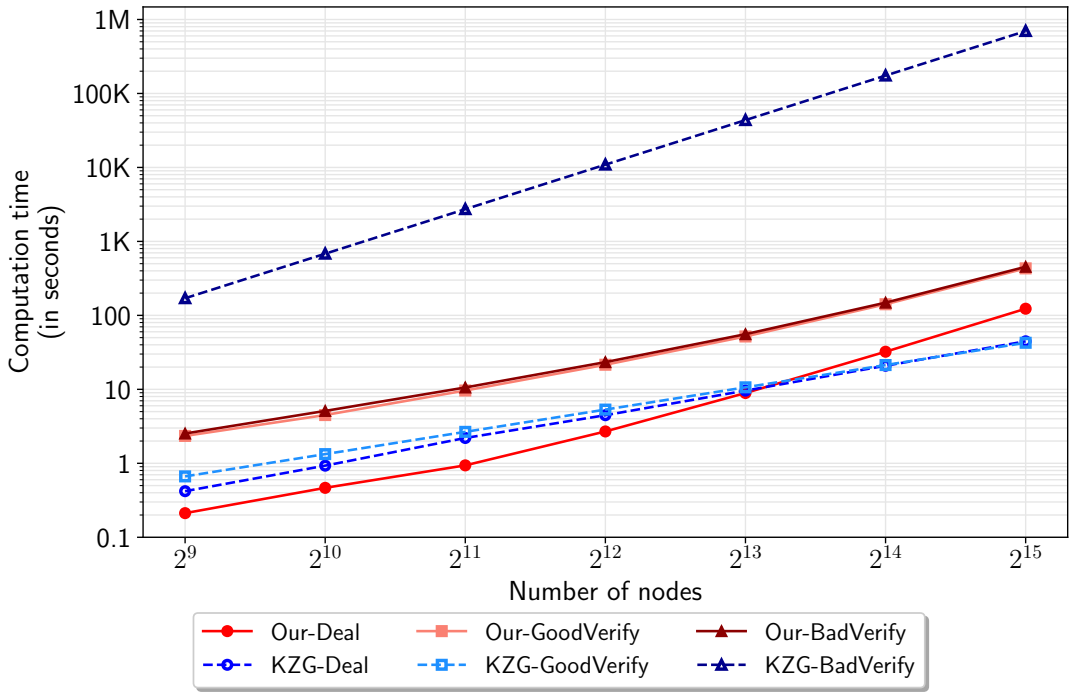


FIGURE 3.7: Computation overhead.

Note that our computation time grows faster than KZG's, which we believe is due to the use of a naïve implementation of multi-point polynomial evaluation. The complexity of our current implementation

is $O(n^2)$ for evaluating an $O(n)$-degree polynomial at $O(n)$ points. In contrast, the implementation in (Zhang et al., 2022) employs an optimized algorithm whose complexity is $O(n \log^2 n)$. However, it is important to highlight that our DKG protocol can benefit from the $O(n \log^2 n)$ polynomial evaluation algorithm as well, and our implementation can be enhanced if a Java implementation for the algorithm becomes available.

CHAPTER 4

Circular Dragon: $\widetilde{O}$ptimal Adaptively Secure Distributed Key and Randomness Generation

---

4.1 Introduction

Distributed key generation (DKG) (Pedersen, 1991; Gennaro et al., 2007; Canetti et al., 1999; Fouque and Stern, 2001; Katz, 2024) is a classic problem of generating a common public key and distributing the corresponding secret key's shares across a network of n nodes collectively. DKG eliminates the need for a trusted setup in threshold cryptographic systems (Komlo and Goldberg, 2020; Bacho and Loss, 2022; Das and Ren, 2024; Boldyreva, 2003; Baek and Zheng, 2003; Gennaro and Goldfeder, 2018). It has also proven crucial for distributed coin flipping (Cachin et al., 2000; DRand, 2023), Byzantine consensus (Guo et al., 2020; Lu et al., 2020), secure multi-party computation (Cramer et al., 2001), securing blockchains (Azouvi and Vukolic, 2022; Feng et al., 2024b) and more. Due to its emerging real-world deployments (potentially on a large scale) (DRand, 2023; Azouvi and Vukolic, 2022), there has been a resurgence of interests (Tomescu et al., 2020; Gurkan et al., 2021; Feng et al., 2024a; Bacho et al., 2024) in designing more efficient DKG.

In this work, we study DKG in the discrete logarithm setting, which has been a major focus of the topic. We focus on the synchronous setting where there is a known upper bound on message delivery time. Our goal is to design such a DKG with a low communication cost and low round complexity, while retaining optimal resilience (tolerating up to $n/2$ corruption) and adaptive security. As we will discuss below, despite a long line of research and recent advancements in both cryptography and distributed computing, significant gaps remain.

"Classical" DKG with $O(n^3)$ communication cost and $O(n)$ rounds. Most DKG protocols (Pedersen, 1991; Gennaro et al., 2007; Canetti et al., 1999; Fouque and Stern, 2001; Katz, 2024) follow a common

paradigm where each node verifiably shares a secret to the network, ensuring every other honest node has a corresponding share. For convenience, we call them joint-VSS DKG. For verifiable secret sharing (VSS) (Pedersen, 1991; Feldman, 1987; Gentry et al., 2022), there is an $O(n^2)$ communication lower bound, leading to an $O(n^3)$ overall communication for joint-VSS DKG protocols. It is easy to see a trivial $O(n^2)$ communication cost lower bound, as every node needs to receive at least $O(n)$ bits indicating all other nodes have received a share. Although honest nodes may keep silent when they receive a share, a malicious dealer can still trigger $O(n)$ nodes to report not receiving.

There is another challenge for ensuring that all nodes have a *consistent* view on which nodes have properly done their VSS. All these DKG protocols handle this issue by assuming a *broadcast channel*. One may realize this assumption by implementing Byzantine Broadcast (BB) protocols (Dolev and Strong, 1983; Nayak et al., 2020; Momose and Ren, 2021)[1], which itself is expensive. Particularly, for adaptive security[2], a BB instance has to incur $\Omega(n^2)$-bit communication cost (Abraham et al., 2023a), so the DKG will incur $\Omega(n^3)$-bit communication cost as it invokes $O(n)$ BB instances. Moreover, for BB in the absence of DKG/trusted setup, recent progresses in BB (Nayak et al., 2020; Momose and Ren, 2021; Elsheimy et al., 2024) and cryptographic tools (Attema et al., 2021; Qiu and Tang, 2023; Garg et al., 2024) realize deterministic quadratic-communication BB protocols. However, deterministic protocols necessitates $O(n)$ rounds due to the classical lower bound by Fischer and Lynch in (Fischer and Lynch, 1982).

Recent sub-cubic DKG still requires $O(n)$ rounds. As discussed, the cubic communication cost is "inherent" to the joint-VSS DKG paradigm; achieving sub-cubic DKG requires a new design. Gurkan et al. made an early attempt (Gurkan et al., 2021), where nodes first somehow *aggregate* their secrets, and only very few nodes are responsible for sharing an aggregated secret. However, their protocol tolerates only $O(\log n)$ corrupted nodes. Two recent works, by Feng et al. (Feng et al., 2024a) and Bacho et al. (Bacho et al., 2024), finally provided DKG protocols with *sub-cubic* communication complexity and optimal resilience. At a high level, the network is arbitrarily partitioned into small groups; within each group, member nodes somehow aggregate their secrets into a "group secret". Then, each group, by using the delicately designed protocols in (Feng et al., 2024a; Bacho et al., 2024), can act as one single node to verifiably share its "group secret" to the whole network. On rough terms, the number of VSS instances

[1]It is possible to realize a broadcast channel through external facilities such as a national TV channel, trusted website, or blockchains, which however introduce additional trust assumptions. Here we focus on the standard point-to-point network.

[2]Throughout this work, we focus on strongly adaptive adversaries (Abraham et al., 2023a) who can retract messages sent by a newly corrupted node in the same round of corruption.

is the number of groups, which is $O(\sqrt{n})$ in (Feng et al., 2024a) and $O(1)$ in (Bacho et al., 2024); The communication cost reduces to $O(n^{2.5})$ and $O(n^2 \log n)$, respectively.

Nonetheless, the network needs a consistent view of which groups have properly finished the secret sharing. To align with their efficiency improvement, both works adopt deterministic consensus, which again inherently incurs $O(n)$ rounds.

Lowering round complexity with $O(n^3)$ communication. To circumvent the barrier of $O(n)$ rounds, we need randomized consensus protocols, which, compared with their deterministic analogues, usually require a stronger setup or more communication. Indeed, many sub-linear-round consensus protocols (Abraham et al., 2019; Cachin et al., 2000; Lu et al., 2020) rely on unique threshold signatures (Boldyreva, 2003) for providing a *common coin*, which in turn requires a trusted setup or a DKG setup. The alternative efforts without a DKG setup involve a subroutine known as *weak common coin* (or oblivious leader election) (Feldman and Micali, 1988; Katz and Koo, 2006; Shrestha et al., 2023; Gao et al., 2022). Weak common coin is weaker than common coin with regard to agreement, i.e., there is a constant probability that honest nodes see different values. Compromising agreement makes it easier to instantiate. However, existing instantiations of weak common coin, again, rely on (variants of) VSS and require at least $O(n^3)$ communication in the absence of DKG setup. In particular, the early constructions by Feldman and Micali (Feldman and Micali, 1988) and Katz and Koo (Katz and Koo, 2006) use $O(n^2)$ VSS instances, so their communication cost is $\Omega(n^4)$. Recently, Shrestha et al. (Shrestha et al., 2023) constructed a weak common coin using $O(n)$ VSS instances, from which, they further gave an $O(1)$-round DKG, but at the cost of $O(n^3)$ communication. Their results can be seen as an extension to the asynchronous techniques from (Gao et al., 2022) for tolerating up to $n/2$ corruptions.

Since existing adaptively secure weak common coin protocols already incur $O(n^3)$ communication cost, existing randomized consensus cannot improve the round complexity of the recent sub-cubic DKG protocols.

Gaps regarding functionality. While the above discussions highlight a major gap in terms of efficiency, there are also substantial gaps regarding functionality. In particular, for secure and efficient secret aggregation, the current sub-cubic DKG protocols (Gurkan et al., 2021; Bacho et al., 2024; Feng et al., 2024a) rely on a special variant of VSS, known as aggregatable publicly verifiable secret sharing (APVSS) from (Gurkan et al., 2021). APVSS currently only has efficient instantiations for secrets which are elements

of a pairing-friendly Elliptic Curve (ECC) group rather than a scalar field, making the APVSS-based DKG incompatible with the standard discrete-logarithm (DL) cryptosystems.[3]

Given these limitations, we explore the following question in this work:

Can we design an adaptively secure DKG (for DL) with field-element secrets, $\widetilde{O}(n^2)$ communication cost, $\widetilde{O}(1)$ rounds, and optimal resilience?

4.1.1 Our Main Result

We answer the above question affirmatively. We present a new DKG protocol ("This work-II" in Table 4.1) for DL with field-element secrets, which also attains $O(\lambda\kappa n^2 \log n)$ communication and terminates in $O(\kappa \log n)$ rounds with an overwhelming probability, where λ and κ are the computational and statistical security parameters, respectively. As building blocks and byproducts, we give the first near-optimal group-element DKG ("This work-I" in Table 4.1), which directly yields a coin-flipping protocol. Our DKG is robust against adaptive adversaries, ensuring that honest nodes always obtain valid key shares. Regarding secrecy, we prove our protocol satisfies *oracle-aided algebraic simulatability* against adaptive adversaries, proposed by Bacho and Loss in (Bacho and Loss, 2022) to capture the adaptive security of many classic DKG protocols including Pedersen's (Pedersen, 1991), GJKR (Gennaro et al., 2007), and Bingo (Abraham et al., 2023b). As a bonus, we also prove our DKG satisfies *full secrecy* (Gennaro et al., 2007; Katz, 2024) against *static* adversaries, ensuring the generated key is not biased.

We remark that the static full secrecy matches the security of GJKR (Gennaro et al., 2007) and the recent work of Katz (Katz, 2024), while for most DKG protocols, including Pedersen (Pedersen, 1991) and the recent sub-cubic DKG protocols (Feng et al., 2024a; Bacho et al., 2024; Gurkan et al., 2021) cannot achieve it. Actually, for DKG with public PK shares, no existing protocol achieves adaptive full secrecy.

[3]We note that (Feng et al., 2024a) also presented a sub-cubic DKG without using APVSS, but it still is unclear how to reach near-quadratic communication cost without APVSS.

TABLE 4.1: Comparison with existing synchronous DKG protocols

Protocol	Resi.	Comm.Cost	Round	Field?	Adap.?	Assumptions
Pedersen (Pedersen, 1991)	$1/2$	$O(\lambda n^3 \log n)$	$O(n)$	✓	✓	DL
GJKR (Gennaro et al., 2007)	$1/2$	$O(\lambda n^3 \log n)$	$O(n)$	✓	✓	DL
Katz (Katz, 2024)	$1/2$	$O(\lambda n^3 \log n)$	$O(n)$	✓	✗	PKE+NIZK
CGJ+ (Canetti et al., 1999)	$1/2$	$O(\lambda n^3 \log n)$	$O(n)$	✓	✓*	DL + Erasure
GJM+ (Gurkan et al., 2021)	$\log n/n$	$O(\lambda n^2 \log^2 n)$	$O(n)$	✗	✗	APVSS
SBKN (Shrestha et al., 2023)	$1/2$	$O(\lambda n^3 \log n)$	$O(1)$	✓	✗	DDH
GRandLine (Bacho et al., 2024)	$1/2$	$O(\lambda n^2 \log n)$	$O(n)$	✗	✓	APVSS
Dragon-I (Feng et al., 2024a)	$1/2$	$O(\lambda n^{2.5} \log n)$	$O(n)$	✗	✓	APVSS
Dragon-II (Feng et al., 2024a)	$1/2$	$O(\lambda \kappa n^{2.5} \log n)$	$O(n)$	✓	✗	PVSS
This work-I (§4.6)	$1/2$	$O(\lambda n^2 \log n)$	$O(\kappa \log n)$	✗	✓	APVSS
This work-II (§4.7)	$1/2$	$O(\lambda \kappa n^2 \log n)$	$O(\kappa \log n)$	✓	✓	APVSS + Erasure

`Resi.`: the maximal fraction of Byzantine nodes the protocol can tolerate.
`Comm.Cost` measures the number of bits sent by all honest nodes. λ and κ are the computational security parameter and the statistical security parameter, respectively.
`Round`: SBKN(Shrestha et al., 2023) requires $O(1)$ rounds in *expect*, while ours can terminate within $O(\kappa \log n)$ rounds with an overwhelming probability.
`Adap.?` asks if the protocol is adaptively secure. Most protocols achieve the relaxed adaptive security definition by (Bacho and Loss, 2022), but GGJ+ achieves the standard one.
`Field?` asks if the secret key is in a scalar field.
`Assumptions`: We consistently omit the assumptions needed for efficient consensus protocols, including SXDH and RO. PVSS relies on PKE and NIZK. APVSS relies on SXDH + BDH (Gurkan et al., 2021) for static security and COMDL + AGM (Bacho and Loss, 2023b) for adaptive security. (Canetti et al., 1999) and ours assume memory erasures (Canetti et al., 1996) for adaptive security.
`Notes`: All protocol listed require a standard PKI and a uniform random string (URS). The comparison assumes the state-of-the-art transparent-setup consensus protocols for implementing broadcast channels. A detailed discussion is available in Sect 4.2.2. We also defer the discussions for asynchronous protocols to Sect 4.2.1: They all incur $\Omega(\lambda n^3)$ communication cost, except two recent concurrent works (Feng and Tang, 2024; Abraham et al., 2025), which however both require stronger setups and heavier tools, and cannot handle 1/2 corruption.

4.2 Related Works

4.2.1 Asynchronous Distributed Key Generation

Existing asynchronous DKG (ADKG) schemes (Abraham et al., 2021a; Das et al., 2022; Kokoris-Kogias et al., 2020; Gao et al., 2022; Abraham et al., 2023b; Groth and Shoup, 2023) follow the joint-verifiable secret sharing (VSS) paradigm, where each node contributes consistent secret shares via asynchronous VSS. Unlike classical DKG protocols, which can rely on deterministic protocols to reach consensus, ADKG protocols must generate a (weak) common coin to enable randomized consensus protocols. This challenge is similar to that of constant-round synchronous DKG protocols, as discussed in the introduction.

The ADKG protocol by Kokoris-Kogias et al. (Kokoris-Kogias et al., 2020) incurs a communication cost of $O(\lambda n^4)$ and requires $O(n)$ rounds, as it relies on $O(n^2)$ instances of asynchronous VSS. Later, leveraging the aggregatable PVSS scheme from (Cascudo and David, 2017; Gurkan et al., 2021), Abraham et al. (Abraham et al., 2021a) proposed a more efficient randomized consensus, yielding an ADKG with $O(\lambda n^3 \log n)$ communication complexity and an expected round complexity of $O(1)$. Gao et al. (Gao et al., 2022) further optimized this by removing the $O(\log n)$ multiplicative factor from Abraham et al.'s result. However, both protocols (Abraham et al., 2021a) and (Gao et al., 2022) generate secret keys in an elliptic curve cryptography (ECC) group. Notably, these ADKG protocols can directly benefit from the adaptively secure aggregatable PVSS (APVSS) scheme (Bacho and Loss, 2023b), thereby gaining adaptive security.

Das et al. (Das et al., 2022) introduced a field-element DKG protocol with $O(\lambda n^3)$ communication complexity and $O(\log n)$ round complexity. Additionally, the Bingo DKG protocol by Abraham et al. (Abraham et al., 2023b) provides a field-element ADKG protocol with $O(\lambda n^3)$ communication complexity, $O(1)$ round complexity, and adaptive security, but it relies on the KZG polynomial commitment (Kate et al., 2010), which necessitates a common reference string (CRS) setup.

Groth and Shoup (Groth and Shoup, 2023) presented a batched protocol capable of generating $O(n)$ keys in a single run. On the other hand, while ADKG protocols typically tolerate up to t corrupted nodes in a system with $n = 3t + 1$ nodes, recent works (Das et al., 2023b; Abraham et al., 2023b) have explored high-threshold ADKG, requiring $2t + 1$ shares for secret key reconstruction.

4.2.2 Relevant Consensus Protocols

We discuss consensus protocols that can be used to implement broadcast channels in existing Distributed Key Generation (DKG) protocols. The most standard approach is to invoke a Byzantine broadcast (BB) (Dolev and Strong, 1983) instance whenever a sender wishes to broadcast a message to the network. The validity property of BB ensures that if the sender is honest, every participant receives the message as it was broadcast. The agreement property guarantees that all receivers decide on the same message, even if the sender is malicious. Finally, the termination property requires that the protocol concludes within a bounded number of rounds.

Note that a PKI setup for digital signatures is necessary for BB and Byzantine Agreement (BA) protocols that tolerate more than $n/3$ corrupted nodes (Fischer et al., 1985). Dolev and Strong (Dolev and Strong, 1983) introduced the first BB protocol capable of tolerating up to $n-1$ corrupted nodes. With standard digital signatures, their BB protocol incurs a communication cost of $O(n^2\ell + \lambda n^3)$ and requires $O(n)$ rounds to broadcast an ℓ-bit input to a network of n nodes. If an aggregatable signature scheme like BLS is employed, the communication cost can be reduced to $O(n^2\ell + \lambda n^2 + n^3)$. Since a classical DKG scheme invokes $O(n)$ BB instances, the total communication cost is considered $O(n^4)$, as noted in (Shrestha et al., 2023; Bacho et al., 2024).

A few recent works have pushed the communication complexity of BB to be nearly optimal. In particular, Momose and Ren (Momose and Ren, 2021) gave a BA protocol with $O(n^2\ell + \lambda n^2)$ communication cost and $O(n)$ rounds, tolerating any $t < n/2$ corrupted nodes. Their BB protocol assumes a constant-sized threshold signature scheme. However, the threshold signature scheme in their protocol is only for compressing a certificate consisting of $O(n)$ digital signatures, so we can use the recent DKG-free threshold signature schemes to instantiate, without causing a circularity issue in DKG constructions. As we have discussed in Section 2.2, under a transparent setup, we have a threshold signature scheme whose size is $O(\lambda \log n)$. Therefore, it leads to a BA protocol with $O(n^2\ell + \lambda n^2 \log n)$ communication complexity under a transparent setup. Additionally, Nayak et al. (Nayak et al., 2020) gave an extension technique, which can adapt a BA for short inputs like Momose and Ren (Momose and Ren, 2021) for handling long inputs. Notably, combining Momose and Ren (Momose and Ren, 2021) and Nayak et al. (Nayak et al., 2020) gives us a BA protocol with $O(n\ell + \lambda n^2 \log n)$ communication complexity and $O(n)$ rounds. Note that for $t < n/2$, there is a trivial reduction from BB to BA, where the sender just needs to multicast its input, and then the network runs a BA for reaching a consensus. Therefore, it

naturally follows a BB protocol with the same complexity. If a CRS such as the q-SDH parameter is allowed, we can remove the factor of $O(\log n)$ from the communication cost.

We remark that Bacho et al. (Bacho et al., 2023a) also gave a BB protocol with $\widetilde{O}(n\ell + \lambda n^2)$ communication cost and $O(n)$ rounds, which does not require threshold signatures. However, their protocol only achieves static security.

Alternatively, since DKG involves $O(n)$ broadcasters, one can use a parallel Byzantine broadcast (PBB) protocol to implement the broadcast channel. For deterministic protocols, Feng et al. (Feng et al., 2024a) proposed a PBB with a communication complexity of $O(n^{2.5}\ell + \lambda n^{2.5} \log n)$, while Civit et al. (Civit et al., 2024) introduced one with a complexity of $O(n^2\ell + \lambda n^2 \log n)$, under a transparent setup. However, in classical DKG schemes such as those by Pedersen (Pedersen, 1991) and Gennaro et al. (Gennaro et al., 2007), each node may broadcast $O(\lambda n)$ bits, meaning that advanced PBB protocols are unlikely to outperform the cubic communication complexity. On the other hand, the DKG protocol by Gurkan et al. (Gurkan et al., 2021) can benefit from the improved PBB protocols, achieving a reduced communication complexity of $O(\lambda n^2 \log^2 n)$.

4.3 Technical Overview

We now outline the challenges and provide a brief overview of our technique suite, called "Circular Dragon", behind our nearly-optimal DKG protocol. The core technique for reducing the round complexity while preserving the sub-cubic communication complexity can be seen as an extension to the "Dragon" methodology, which is formally introduced in (Feng et al., 2024a) and also implicitly used in (Bacho et al., 2024).

4.3.1 Remaining Challenges in the Dragon Method (Feng et al., 2024a)

We start with a brief overview of the "Dragon" method (Feng et al., 2024a) that achieves sub-cubic communication, and examine its bottleneck of further realizing sublinear rounds.

The Dragon method (Feng et al., 2024a), short for Decentralization at the cost of Representation after Arbitrary GrOupiNg, proceeds in two steps: First, partition the honest-majority network arbitrarily into a few small groups (or called consortiums), and ensure that *at least one* group has an honest majority (following from the pigeonhole principle); Then, each group executes small-scale *intra-group* protocols to form a group secret (usually shared through some VSS variant), followed by certain new protocols to do *inter-group* communication (from a group to all) *as if* a single representative group member communicates with the whole network.[4] A component called consortium-sender Byzantine broadcast (CSBB) (Feng et al., 2024a) realizes the latter step, and it costs only $O(\lambda n^2)$ communication for a group to broadcast an $O(\lambda n)$-bit message (as efficient as *one* normal BB). In (Feng et al., 2024a), each group size is $O(\sqrt{n})$, leading to $m = O(\sqrt{n})$ groups and subsequent $O(\lambda n^{2.5})$-bit total communication.

However, CSBB relies on deterministic Byzantine agreement (Momose and Ren, 2021) running across all n nodes, introducing an $O(n)$ round complexity for the entire DKG.

4.3.2 Circular Dragon Approach: Near-optimal DKG/Coin-flipping

As a crucial building block of our DKG scheme and a necessary byproduct of our design, we first propose an adaptively secure coin-flipping protocol with near-optimal complexities without strong setups, following from a recursive DKG for *group*-element secrets. This enables us to initialize a particular adaptively secure unique threshold signature (Bacho et al., 2024). See Table 2.1 in Section 2.3 for a

[4] Another recent work GRandLine (Bacho et al., 2024) can be similarly interpreted with a different intra-group treatment.

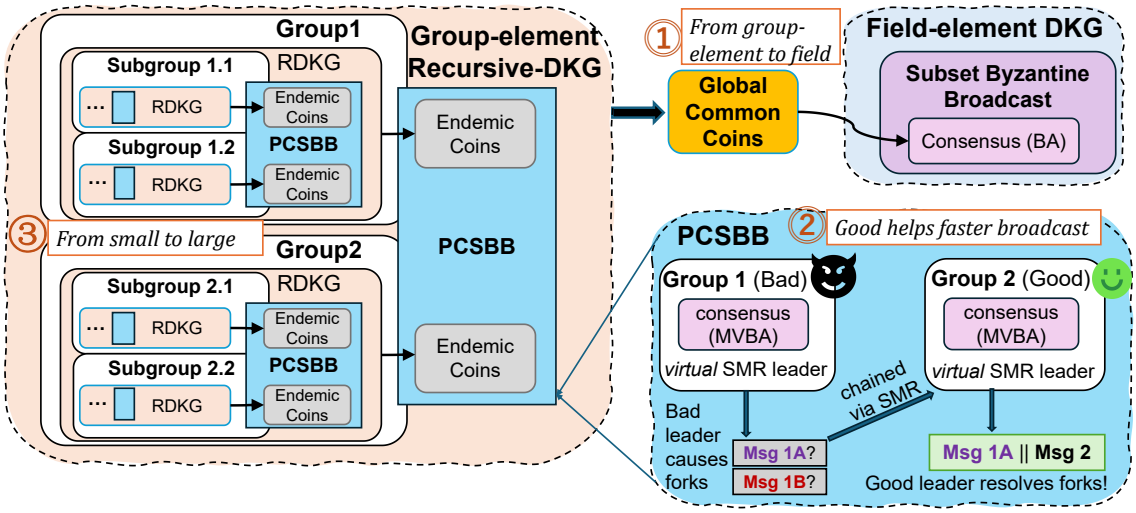


FIGURE 4.1: The technique suite of "Circular Dragon".

summary of this result. Jumping ahead, our Circular Dragon method involves three *different* kinds of "self-boosting" techniques (thus circular), as shown pictorially in ①,②,③ in Fig.4.1.

Parallel CSBB as consensus component. We first go over the consensus building block (Part ② in Fig.4.1) used in our new group-element DKG. Recall that the reason for linear rounds in recent sub-cubic DKG protocols (Feng et al., 2024a; Bacho et al., 2024) is m concurrent instances of CSBB for group-to-network broadcasting where m denotes the number of groups. Particularly, inside *each* CSBB, there is a deterministic consensus running among the entire network of n nodes. To solve the efficiency problem, we formulate parallel CSBB (PCSBB) and construct it more efficiently than a composition of multiple independent CSBB instances executing in parallel. At the heart of our design is a sublinear-round consensus without going through a weak common coin; instead, we leverage the grouping structure of participants to pool m PCSBB instances together, resulting in more efficient group-to-network broadcasting with a round complexity linear only in m instead of n, where m is sublinear in n.

As briefly recalled in Section 4.3.1, after arbitrarily partitioning the network, there are a smaller number of $m = o(n)$ groups/consortiums, and each group has a small size of $\eta = o(n)$ nodes, with at least one *good* group having an honest majority. If viewing each consortium as a "virtual party", then there are m such parties, one of which must be honest. The Dragon method invokes m CSBB instances, where each instance allows one virtual party (representing a group of η nodes) to broadcast across all n nodes.

To reduce the round complexity of m consensus instances among all n parties, we make one critical observation: one honest "virtual party" can help all consensus instances reach agreement faster. An encouraging fact is that a rotating-leader state machine replication (SMR) (Abraham et al., 2020; Bhat et al., 2021) implements a stack of "parallel" Byzantine broadcasts. A rotating-leader SMR executes

epoch by epoch and each epoch sees a new leader. Once an honest leader becomes the leader of an epoch, the whole network not only agrees on the value proposed by the current leader (within constant rounds at the cost of merely $O(n^2)$ communication), but also simultaneously agrees on the values proposed in all previous epochs even with malicious leaders. Suppose there are $m = o(n)$ leaders, with at least one honest leader. The rotating-leader SMR naturally realizes m BB instances after $2m = o(n)$ epochs, where leaders propose their input messages in the first m epochs, and in the latter m epochs, the whole network reaches agreement on what has been committed in the first m epochs.

Drawing from the above inspiration, we can stack m CSBB instances in a faster way by emulating a rotating-leader SMR with m virtual leaders, where each consortium emulates a leader and performs two tasks as if a single leader node: (1) form a proposal block that contains the input message and extends the latest block; (2) sign the proposal block and disseminate it to all n nodes.

For task (1), members of the current leader consortium invoke an intra-consortium consensus to agree on which block to extend, as different nodes might have different views about the latest block. A straightforward idea is to let each consortium member use a deterministic intra-consortium BB to broadcast its candidate block to extend, which however causes large cubic communication and linear rounds in the consortium size. Our construction instead adopts a *single* intra-consortium instance of complexity-optimal multi-valued validated Byzantine agreement (MVBA) (Abraham et al., 2023c), as MVBA enables us to choose a carefully designed (external) validity condition to pick up the globally latest block endorsed by sufficient signatures. Moreover, such intra-consortium MVBA can terminate in constant rounds, using only endemic setup for intra-consortium common coins, which can be achieved from an intra-consortium DKG within at most η rounds (and can be further reduced to nearly constant rounds by technique ③ illustrated in Fig.4.1).

For task (2), we adopt multiverse threshold signature (MTS) with silent setup to reduce the size of proposal blocks, and adopt the standard techniques of cryptographic accumulator and erasure code to minimize the communication overhead of disseminating the proposal blocks from leader consortiums to the entire network. More details about PCSBB are presented in Section 4.5.

Recursively applying PCSBB to realize nearly-optimal group-element DKG. If using our new PCSBB to replace $m = O(\sqrt{n})$ CSBB instances in Feng et al.'s DKG (Feng et al., 2024a) (where $m \approx \eta = O(\sqrt{n})$), we already have an adaptively secure protocol with $\widetilde{O}(\lambda n^{2.5})$ communication and $O(\kappa\sqrt{n})$

rounds. We detail this construction in Section B.2. However, to realize nearly-quadratic communication, we further apply PCSBB to reduce the rounds of Bacho et al.'s group-element DKG (Bacho et al., 2024) (Part ③ in Fig.4.1), which has a recursive structure causing more challenges for achieving nearly-constant round instantiation.

Before diving into our recursive group-element DKG protocol, we first briefly reintroduce the (aggregatable) publicly verifiable secret sharing (PVSS), which is the core component in existing sub-cubic DKG schemes. On rough terms, a PVSS transcript for a network $\mathcal{P} = \{P_1, \ldots, P_n\}$ consists of a sequence of ciphertexts $(\mathsf{Enc}(ek_1, f(1)), \ldots, \mathsf{Enc}(ek_n, f(n)))$ along with a valid proof, where f is a t-degree secret polynomial, and ek_i is the public key of P_i. If the PVSS scheme is aggregatable, a transcript with f_1 and another transcript with f_2 can be aggregated into one transcript with $f_1 + f_2$.

Bacho et al.'s DKG works in a recursive manner for network "partition". At the first layer, the entire network $\mathcal{P}$ is partitioned into two groups $\mathcal{D}^{1,0}$ and $\mathcal{D}^{1,1}$. Each group $\mathcal{D}^{1,b}$ is supposed to generate an aggregatable PVSS (APVSS) transcript $\tau^{1,b}$ (encrypting secret shares for the network $\mathcal{P}$) and broadcasts $\tau^{1,b}$ to the entire network using CSBB; so the network will obtain at most two valid APVSS transcripts $\tau^{1,0}$ and $\tau^{1,1}$, and the aggregation of them defines the DKG output. Then, each group is recursively partitioned to generate its APVSS transcript: the group $\mathcal{D}^{1,b}$ is partitioned into two subgroups, each subgroup generates an APVSS transcript and uses CSBB to broadcast within the group; so the group $\mathcal{D}^{1,b}$ generates its APVSS transcript $\tau^{1,b}$ by aggregating two subgroup's transcripts. This recursion occurs over $O(\log n)$ layers.

To reduce the round complexity of Bacho et al.'s DKG, we need to apply PCSBB to replace CSBB instances for *every* recursion layer. However, in PCSBB, each consortium runs a randomized consensus, requiring endemic common coins within the consortium and thus an intra-consortium DKG. Since the consortium size is up to $n/2$, no existing DKG protocol satisfies our efficiency requirements, as they either incur cubic communication or linear rounds.

We address this issue by further leveraging the recursive structure to bootstrap each layer's intra-group common coins: the recursion not only generates the final DKG output but also provides DKG setups at every layer. Specifically, at the first layer, when group $\mathcal{D}^{1,b}$ is generating the APVSS transcript $\tau^{1,b}$ for the network $\mathcal{P}$, it uses the same process to additionally generate an APVSS transcript for its own participants. The latter APVSS transcript is only for the group $\mathcal{D}^{1,b}$'s members thus provides a DKG setup for endemic common coins within group $\mathcal{D}^{1,b}$. Given these intra-group common coins, $\tau^{1,0}$ and

$\tau^{1,1}$ can therefore be broadcast via PCSBB in constant rounds, and then aggregated to derive the final DKG output. Assuming the recursion occurs over $\ell = O(\log n)$ layers, at the bottom layer, each node generates $\ell + 1$ APVSS transcripts, with sizes $O(\lambda n), \ldots, O(\lambda n/2^{\ell})$, respectively, and the total size remains $O(\lambda n)$. Essentially, the aggregation of APVSS transcripts of length $O(\lambda n/2^k)$ enables a DKG setup for a group at the k-th layer. More details can be found in Section 4.6.

Lifting group-element DKG to field-element DKG, efficiently. Although our new recursive DKG protocol only generates group-element secrets, it provides an efficient distributed setup for a coin-flipping protocol. Given that, we present a field-element DKG protocol in the coin-aided model in Section 4.7 (Part ① in Fig.4.1), preserving adaptive security and the *same complexities*.

The reason that only group-element secrets can be generated in our group-element DKG construction is the use of APVSS. Currently, APVSS lacks efficient constructions that support field-element secrets with adaptive security. Hence, we turn to PVSS.

Let us examine a simple PVSS-based DKG (Fouque and Stern, 2001): Everyone generates a PVSS transcript locally for the network and then broadcasts it to the network; Then, all nodes will have the same view and can obtain their secret key share by aggregating all secret shares decrypted from valid PVSS transcripts. As noted in (Feng et al., 2024b,a), for DKG security, it suffices to ensure that at least one honest node contributes to the final secret key. Intuitively, a more efficient approach is to leverage the common coin to randomly select κ relay nodes, ensuring at least one is honest, and then only let the selected relay nodes to broadcast PVSS.

However, challenges remain for adaptive security: (1) An adaptive adversary can corrupt all κ selected nodes, learn their secrets, and even retract their PVSS broadcasts; (2) Current field-element PVSS constructions only have static security; (3) These issues must be solved with nearly-optimal complexities.

Use SBC to resolve adaptive corruption of committee. For (1), we still ask all honest nodes compute a PVSS transcript to broadcast, and let them erase secrets before broadcasting. However, extra techniques are required to ensure efficiency.

To avoid trivially broadcasting n PVSS transcripts, we introduce *subset Byzantine broadcast* (SBC), in which all participants first "lock" their inputs across the network via efficient information dispersal; and at a later point, a subset of κ inputs is selected to deliver, when it is too late for the adversary to retract. Our construction utilizes a couple of known techniques: each node erasure encodes its input value into

many fragments and to disperse all inputs to the network, while only values of the selected subset will be reconstructed. In addition, to ensure a consistent view on the reconstructed messages, we use a constant-round Byzantine Agreement protocol from (Abraham et al., 2019) for each message. Further details are presented in Section 4.7.1. When applying SBC to broadcast the PVSS transcripts of $O(\kappa)$ randomly selected nodes, the communication cost is $O(\kappa\lambda n^2 \log n)$, and the round complexity is $O(\log \kappa)$.

Adaptive security via interactive PVSS. While non-interactive PVSS for field-element secrets exists (Schoenmakers, 1999; Gentry et al., 2022; Fouque and Stern, 2001; Groth, 2021), they are only proven to be statically secure. There appears to be a significant barrier, as the underlying PKE needs stronger selective-opening security in the face of adaptive corruptions. We turn our attention to *interactive* PVSS. Nonetheless, existing constructions such as the recent one from Das et al. (Das et al., 2023c) have efficiency issues: the dealer sends a polynomial commitment PCom having $O(n)$ bits to all nodes, so directly using it during n interactive PVSS instances still incurs cubic communication (as everyone receives n such commitments). [5] Moreover, it was only proven statically secure.

We modify the synchronous variant of Das et al.'s interactive PVSS to meet our efficiency requirement, similar to how (Feng et al., 2025) modifies Das et al.'s asynchronous version. In particular, since the size of PCom is $O(\lambda n)$, if every node needs to send such a PCom to all other nodes, the overall communication cost is already $O(\lambda n^3)$. Instead of directly compressing the polynomial commitment, observe that there is no need for the dealer to send the entire PCom to each P_i, and the dealer can just send each party a small corresponding part of PCom with constant size. Given the modification, the collective communication cost for interactively generating n PVSS transcripts becomes $O(\lambda n^2)$. In addition, we formally prove the modified version is secure against adaptive adversaries (as part of our DKG's security proof). More details are in Section 4.7.2 and 4.7.3.

[5] One may reduce the size of polynomial commitment (e.g., using KZG commitment), but relies on a stronger setup and may cause issues of public key reconstruction.

4.4 Modeling

Communication model. The network consists of n nodes $\mathcal{P} = \{P_i\}_{i\in[n]}$ and is assumed to be synchronous and fully connected, meaning that the protocol proceeds in "rounds", and during every round, each uncorrupted node can confidentially send messages to any uncorrupted recipient, with delivery ensured when the next round begins. To measure the communication complexity, we count the number of bits sent by honest nodes. We also consider n to be some unfixed polynomial in λ and κ, where λ and κ represent the cryptographic and statistical security parameters, respectively.

Threat model. We assume a standard PKI and a uniform random string (URS), such that all public keys and URS are accessible to everyone. The adversary $\mathcal{A}$ is assumed to be probabilistic polynomial time (PPT) bounded, and it can corrupt up to t participants (where $t < n/2$) and arbitrarily control the behavior of corrupted nodes. We consider an *adaptive* adversary $\mathcal{A}$ that can gradually choose the participating nodes to corrupt during the initial phase and protocol execution. Once a node corrupted, the adversary gets access to its local states and controls its subsequent behaviors. The adaptive adversary is also "rushing" to perform the after-the-fact removal of messages, meaning that it is the first party observing the messages sent in each round, such that if the adversary adaptively corrupts a node during some round R, it can immediately retract the messages that were just sent by this node in round R.

4.5 Parallel Consortium-Sender Byzantine Broadcast

This section introduces parallel consortium-sender Byzantine broadcast (PCSBB)—our core consensus component towards realizing our more efficient DKG protocol.

4.5.1 The Definition

Consortium-sender Byzantine broadcast (CSBB) is a central consensus primitive used in existing sub-cubic DKG protocols (Feng et al., 2024a; Bacho et al., 2024). At a high level, CSBB enables a group of senders, called the sender consortium, to broadcast a common input message to a large set of receiving nodes $\mathcal{P}$, at the cost as if the sender consortium is a single node. CSBB guarantees that if the sender consortium has an honest majority and all honest senders propose the same input message $\texttt{msg}$, then all receivers in $\mathcal{P}$ will terminate with $\texttt{msg}$. Moreover, even if the sender consortium does not have an honest majority, CSBB ensures that all receivers will still terminate with the same message, preserving consistency across the network.

Similar to how parallel Byzantine broadcast (PBB) generalizes Byzantine broadcast (BB) by enabling multiple senders to broadcast their distinct input messages in parallel, PCSBB extends CSBB to support multiple consortium senders. As briefly mentioned, PCSBB can be constructed more efficiently than multiple PBB instances when the sender consortiums are partitioned from an honest-majority network, leveraging the fact that at least one sender consortium has an honest majority. PCSBB can be formally defined as follows.

DEFINITION 6. *A PCSBB protocol involves n participants $\mathcal{P} = \{P_1, \ldots, P_n\}$ and is paired with m consortium-sender groups $\left\{\mathcal{D}^{(i)}\right\}_{i\in[m]}$. A consortium-sender group is said to be* **honest***, if it has an honest majority, otherwise called* **corrupted***. A PCSBB protocol is (n, m, t)-secure, if the following properties can hold with overwhelming probability against any adaptive PPT adversary $\mathcal{A}$ corrupting up to t parties in $\mathcal{P}$:*

- ***Termination.*** *All honest nodes output some m-element vector $\mathcal{V} = [v_1, \ldots, v_m]$.*
- ***Agreement.*** *In the existence of an honest consortium-sender group whose honest participants also share the common input, all honest nodes in $\mathcal{P}$ output the same vector $\mathcal{V}$.*
- ***Validity.*** *Conditioned on that the honest nodes in an honest consortium $\mathcal{D}^{(i)}$ share the same input msg_i, then $\mathcal{V}[i] = \mathsf{msg}_i$.*

4.5.2 Construction Overview

A straightforward approach to implementing PCSBB is to run m CSBB instances in parallel when there are m sender consortiums. However, the existing CSBB construction (Feng et al., 2024a) requires the entire network $\mathcal{P}$ to perform a deterministic Byzantine agreement protocol, which incurs $O(n)$ rounds of communication. While randomized Byzantine agreement can reduce the round complexity, it requires a (weak) common coin accessible to all nodes in $\mathcal{P}$, leading to a circular dependency since our goal is precisely to establish such a global common coin.

We propose an efficient PCSBB construction that terminates in sub-linear rounds and relies solely on *endemic* common coins within each consortium. Since each consortium can independently run a DKG protocol on its own scale to set up endemic common coins, which is significantly more efficient than executing a DKG protocol across the entire network to enable global common coins.

In the following, we overview the idea of our construction, which is based on an observation that some leader-based state machine replication (SMR) protocol (Abraham et al., 2021b; Bhat et al., 2021) can be easily adapted to realize a round-efficient parallel broadcast, and we can obtain a PCSBB by letting each consortium emulate a SMR leader within constant $O(\kappa)$ rounds by leveraging endemic coins.

Observation: Rotating-Leader SMR as Parallel Broadcasts. An SMR protocol enables participants to agree on a sequence of proposed messages. One well-established method for synchronous SMR is the *rotating-leader* paradigm (Abraham et al., 2021b; Bhat et al., 2021). In this approach, protocol execution is divided into successive epochs, with each epoch assigned to a leader in a round-robin manner. The leader proposes an input message, and participants validate it by performing checks, such as ensuring it does not fork any previously confirmed proposals. Once the proposal is validated, participants vote and disseminate the result to the entire network. When enough votes for the same proposal are collected, a certificate for this proposal is formed, a node waits for an additional round-trip to detect potential equivocation before committing the message and advancing to the next epoch.

In such an SMR protocol, disagreement can occur if the current epoch has a malicious leader. For instance, some honest nodes might only commit a prefix of another honest node's output. However, such disagreement is temporary. As the leader rotates, there will eventually be an epoch with an honest leader, after which all participants will reach agreement and have the same committed blocks.

We can use this rotating-leader SMR protocol to realize parallel broadcast among multiple senders. Suppose there are m senders, with at least one being honest. During the first m epochs, each sender acts as the leader in turn, proposing its message. After this in another m successive epochs, each sender proposes a default message. At the conclusion of these $2m$ epochs, all nodes terminate and output the committed messages from the first m epochs. Since at least one sender is honest, there must be an epoch with an honest leader during the latter m epochs. Consequently, all honest nodes will have committed the same messages for the first m epochs. With an appropriate SMR instantiation, this approach allows m senders to broadcast their ℓ-bit messages with $O(mn\ell + \lambda mn^2 \log n)$ communication cost and $O(m)$ rounds.

Technique: Emulating SMR Leaders with Consortiums. The core idea behind CSBB is to have the sender consortium emulate a single sender in a conventional Byzantine broadcast protocol, maintaining efficiency comparable to a single sender. We adopt a similar strategy for PCSBB by letting each sender consortium emulate a leader in the SMR-based parallel broadcast.

In the rotating-leader SMR protocol, a leader performs two main tasks: (1) Form a proposal block containing its input message, which extends the latest certified block known to any honest node; (2) Sign the proposal block and send it to all nodes in the network. To enable a sender consortium to emulate an SMR leader, we require efficient protocols for these tasks within the consortium.

For the first task, a single leader in the conventional SMR collects certified blocks from all nodes and extend the latest block with the highest rank. However, within a leader consortium, different nodes may receive blocks with differing ranks. To resolve this, they must run an intra-consortium consensus to agree on which block to extend. Designing this consensus while ensuring efficiency is non-trivial. In particular, if the consortium collectively executes a Byzantine agreement, it might not terminate as honest nodes' inputs are different. Alternatively, if every node in the consortium invokes a BB to broadcast the highest-ranked block, the communication cost becomes cubic in the consortium size, creating a communication bottleneck. Moreover, for achieving constant round complexity, endemic common coins within each consortium are also required.

We address this by carefully employing a (randomized constant-round) multi-valued validated Byzantine agreement (MVBA) protocol (Abraham et al., 2023c). Specifically, the consortium follows a carefully designed mechanism to exchange signatures on candidates of the latest block. This ensures that all honest nodes eventually gather enough signatures for a candidate block, and any block with sufficient

signatures is guaranteed to be the latest certified block known to any honest node. Once this condition is met, the consortium can invoke the MVBA protocol to reach consensus on a block that has been signed by enough nodes.

For the second task, we utilize a multiverse threshold signature (MTS) scheme to generate a compact signature for the proposal block. To efficiently disseminate the block, we implement a data delivery component similar to those used in previous works, such as (Feng et al., 2024a; Bhat et al., 2021). This component leverages erasure coding and cryptographic accumulators to reduce communication costs. For nodes outside the leader consortium, their actions mostly inherit the classic rotating-leader SMR protocol (Bhat et al., 2021), except that while receiving the proposal block from the leader consortium, they follow the verification of MTS and the optimized data delivery component for communication efficiency[6].

[6]We remark that compared with the classic rotating-leader synchronous SMR instantiations (Abraham et al., 2021b; Bhat et al., 2021), our SMR-inspired PCSBB, for presentation and conceptual simplicity, is not optimized for *good-case latency*.

4.5.3 Building Blocks

We introduce the ingredients used in our PCSBB protocol below, and recall their formal definition and instantiation in Section 2.

MTS. In a multiverse threshold signature (MTS) scheme, each user P_i can locally sign a message $\mathtt{msg}$ using its won secret key sk_i via the algorithm $\mathsf{MTS.PartSign}$, and the resulting partial signature σ_i can be verified under vk_i via $\mathsf{MTS.PartVrfy}$. Given any set of public keys $\{vk_i\}_{i\in U}$ and a threshold number $k \leq |U|$, the algorithm $\mathsf{MTS.UniverseSetup}$ can deterministically compute a combine key CK_U and a verification key VK_U. Then, given k valid partial signatures by distinct nodes within U, the algorithm $\mathsf{MTS.Combine}$ can produce a threshold signature σ for the message $\mathtt{msg}$ using $\mathsf{CK_U}$, which can be verified under VK_U via the algorithm $\mathsf{MTS.Vrfy}$.

We consider the MTS scheme from (Attema et al., 2021), which relies on the SXDH assumption in the random oracle model, solely assumes a conventional PKI setup, and gives a size of $O(\lambda \log |U|)$ threshold signature where $|U|$ is the size of a specific universe. In our PCSBB, we use $(\mathsf{VK}_{\mathcal{P}}, \mathsf{CK}_{\mathcal{P}})$ to denote the keys of the network $\mathcal{P}$ (as a universe), and $(\mathsf{VK}_{\mathcal{D}_i}, \mathsf{CK}_{\mathcal{D}_i})$ for a consortium $\mathcal{D}_i$.

ACCUMULATOR. In a cryptographic accumulator scheme ACC, one can compute a compact accumulator value u for a set $S = \{s_j\}$ via a deterministic algorithm $\mathsf{Acc.Eval}$; and for each element $s_j \in S$, an algorithm $\mathsf{Acc.Wit}$ can produce a witness w_j, such that an algorithm $\mathsf{Acc.Vrfy}$ can verify if s_j is in the set accumulated by u^* given w_j. We consider the Merkle-tree based construction, whose accumulator value has $O(\lambda)$ bits, while the witness has $O(\lambda \log |S|)$ bits.

ERASURE CODE. A (k, n)-erasure code (Blahut, 1983) comprises two deterministic algorithms: $\mathsf{EC.Encode}$ and $\mathsf{EC.Decode}$. The $\mathsf{EC.Encode}$ takes a value $\mathbf{m}$ as input and outputs n fragments $\mathbf{c} = (\mathtt{c}_1, \ldots, \mathtt{c}_n)$. Any k elements in $\mathbf{c}$ can reconstruct the $\mathbf{m}$ using the $\mathsf{EC.Decode}$. Throughout this thesis, we focus on (k, n)-erasure code with $k = \lfloor \frac{n}{2} \rfloor + 1$.

MVBA. A multi-valued validated Byzantine agreement (MVBA) allows a group of nodes, each having an input v_i satisfying a pre-defined external predicate, to agree on v satisfying this predicate. We denote an execution of MVBA as $\mathsf{MVBA}\langle\{P_i(v_i)\}\rangle \rightarrow v$. We consider the synchronous MVBA protocol from (Abraham et al., 2023c), which incurs $O(\eta\ell + \lambda\eta^2 \log \eta)$ communication cost for agreeing on a ℓ-bit value among a group of η nodes, when instantiating with the MTS scheme, accumulator scheme, and

DD.send[sid](msg, u^*). Calling party P_i, on input (msg, u^*), performs the following steps:

// Encode the message and send fragments to different parties

EC.Enc(msg) $\rightarrow (c_{i,1}, \ldots, c_{i,n})$; Acc.Eval$(ak, \{(j, c_{i,j})\}_{j\in[n]}) \rightarrow u_i$

terminate if $u^* \neq u_i$

for $j \in [n]$, Acc.Wit$(ak, u_i, (j, c_{i,j})) \rightarrow w_{i,j}$ and **send** $(\mathsf{dd}, \mathsf{sid}, \mathsf{propose}, u_i, c_{i,j}, w_{i,j})$ to P_j

DD.relay[sid](u^*). Calling party P_i, on input u^*, performs the following steps:

// Echo the fragments that match the given accumulator

receive $\{(\mathsf{dd}, \mathsf{sid}, \mathsf{propose}, u_j, c_{j,i}, w_{j,i})\}_{j\in\mathbb{J}}$

for $j \in \mathbb{J}$ **do**

if $u_j = u^* \wedge$ Acc.Vrfy$(ak, u_j, (j, c_{j,i}), w_{j,i}) = 1$ **then**

multicast $(\mathsf{dd}, \mathsf{sid}, \mathsf{relay}, c_{j,i}, w_{j,i})$, **break**

DD.recover[sid](u^*). Calling party P_i, on input u^*, performs the following steps:

// Reconstruct the message with fragments that match the given accumulator

receive $\{(\mathsf{dd}, \mathsf{sid}, \mathsf{relay}, c_k, w_k)\}_{k\in\mathbb{K}}$, $C \leftarrow \emptyset$

for $k \in \mathbb{K}$ **do**

if Acc.Vrfy$(ak, u^*, (k, c_k), w_k) = 1$ **then** $C \leftarrow C \cup \{(k, c_k)\}$

if $|C| = t + 1$, **then** `msg` $\leftarrow$ EC.Decode(C), **break output** `msg`

FIGURE 4.2: The Data Deliver protocol.

the erasure code introduced above, and assuming a common coin. In addition, there is a predetermined number $\Delta_{\mathsf{MVBA}} = O(\kappa)$, so that the MVBA protocol can terminate within Δ_{MVBA} rounds except with a probability $2^{-\kappa}$, as shown in Lemma 2.

DATA DELIVERY (DD). DD is a two-round data delivery subroutine (Fig. 4.2) designed to efficiently disseminate long messages across the network from one or multiple senders. It is based on erasure coding and an accumulator scheme, adapted from the deliver subroutine in (Bhat et al., 2021), and provides three APIs: DD.send, DD.relay, and DD.recover. These APIs handle message dispersal, fragment relaying, and message reconstruction, respectively. Notably, DD.relay takes an additional input u, representing the accumulator value of a message, ensuring that only fragments consistent with this value are relayed.

The guarantee is that if all honest nodes use the same u in DD.relay and an honest node invokes DD.send with msg whose accumulator value matches u, then every honest node in the network can receive msg. The communication complexity for a single DD instance is $O(n\ell + \lambda n \log n)$, where ℓ is the message length, irrespective of how many nodes invoke DD.send for the same message.

4.5.4 The Construction

Formal description. We present PCSBB in Fig. 4.3. It requires a setup phase in which (1) the public parameters for the underlying MTS, MVBA and Acc are honestly generated and available to all nodes, (2) the PKI setup for the underlying MTS has been established, and (3) within each consortium $\mathcal{D}_i$ the setup for enabling one-round coin flipping has been established.

Terminology explanation: Following the conventional terminology of SMR protocols, we use the following variables in our PCSBB protocol.

- *Epoch*: Time is evenly sliced into synchronous periods called epochs. In each epoch e, one sender consortium is assigned as the leader consortium in a round robin manner.
- *Blocks and Certificates*: When proposing a message msg_i as a leader consortium in the e-th epoch, the sender consortium $\mathcal{D}_i$ needs to pack their input message into a *block*, which is a tuple $B := (e, \mathsf{msg}, e', u', \sigma)$, where (e', u', σ) can be seen as a *certificate* of the previous block B' created at e'-th epoch. Specifically, u' is the accumulator value of B', while σ is a valid threshold signature on u' by the network $\mathcal{P}$. We say B extends B' if u' of B' is included in B. B'' is an *ancestor* of B', if B' extends B'', or an ancestor of B' extends B''. BLKs is a set of valid blocks, and CERTs is the set of valid certificates.
- *Rank*: For a block B or a certificate cert, its rank is the epoch number e it includes. A block B has a higher rank than B', if $e > e'$. Same rule applies to certificates.

Main protocol. As illustrated in Fig. 4.3, the leader consortium rotates every epoch. During the first m epochs, each consortium sequentially proposes its input message msg_i, while in the last m epochs, they propose only empty messages. The protocol concludes after $2m$ epochs, at which point every honest node P_i outputs the messages it has committed during the first m epochs.

Each epoch consists of a joint execution of PCSBB.Epoch, which follows a structured five-step process. We present the details in the next section.

4.5.5 The Details of PCSBB.Epoch

The details of PCSBB.Epoch are depicted in Fig. 4.4 and Fig. 4.5. We verbally describe PCSBB.Epoch below.

PCSBB[sid]. Each P_i performs the following steps:

CERTs $\leftarrow \{\}$, BLKs $\leftarrow \{\}$, STATE $\leftarrow$ (CERTs, BLKs)

for $e \in [1, 2m]$ **do**// s is the index of the leader consortium $\mathcal{L}_e$ in epoch e

$s \leftarrow e \mod m$, **if** $s = 0$ **then** $s \leftarrow m, \mathcal{L}_e \leftarrow \mathcal{D}^{(s)}$

if $e > m$ **then** msg $= \perp$// commit empty blocks in the last m epochs

else if $i = s \wedge P_i \in \mathcal{L}_e = \mathcal{D}^{(s)}$ **then** msg $=$ msg$_i$ **else then** msg $= \perp$

STATE $\leftarrow$ PCSBB.Epoch(sid, $e, \mathcal{P}, \mathcal{D}^{(s)}$, msg, STATE)

parse STATE $\rightarrow$ (CERTs, BLKs)

output messages in the **committed** blocks created in the first m epochs in BLKs

FIGURE 4.3: The PCSBB construction. Each epoch lasts for $\Delta_{\text{epoch}} = 8 + \Delta_{\text{MVBA}}$ time and each MVBA lasts for $\Delta_{\text{MVBA}} = O(\kappa)$ time.

PCSBB.Epoch$\langle$sid, $e, \mathcal{P}, \mathcal{D}$, msg$_i$, STATE$\rangle$

1 : // Party P_i has input msg$_i$ if $P_i \in \mathcal{D}$ and $e \leq m$

2 : Timer $T \leftarrow 0$; STATE $\rightarrow$ (CERTs, BLKs)

3 : highest-cert $\leftarrow$ last valid element in CERTs, votes $\leftarrow \emptyset$

Step 1 (Epoch initialization):

1 : **if** $e > 1$ **then** parse highest-cert $\rightarrow (e', u', \sigma')$

2 : receive $\{(\text{sid}, \text{new-cert}, e-1, \text{hcert}_j)\}_{j \in H_1}$

3 : **for** $j \in H_1$ **do** parse hcert$_j \rightarrow (e-1, u_j^*, \sigma_j)$

4 : **if** MTS.Vrfy(VK$_{\mathcal{P}}, \sigma_j, u_j^*) = 1 \wedge e - 1 > e'$

5 : **then** highest-cert $\leftarrow$ hcert$_j$; CERTs$[e-1] \leftarrow$ hcert$_j$

6 : **send** (sid, highest-cert, e, highest-cert) to all $P_j \in \mathcal{D}$

Step 2 (Propose): $T = 1$

1 : **if** $P_i \in \mathcal{D}$ **then** // if P_i is in the leader consortium

2 : receive $\{(\text{sid}, \text{highest-cert}, e, \text{hcert}_j)\}_{j \in H_2}$

3 : **update** highest-cert according to the received ones

4 : highest-cert $\leftarrow$ LeaderAgreeCert$\langle$sid, $e, \mathcal{D}, P_i$(highest-cert)$\rangle$

5 : $(B_e, u) \leftarrow$ buildBlock(e, cert, msg), long-msg $\leftarrow B_e$

6 : $\pi_i \leftarrow$ MTS.PartSign(MTS.sk_i, u)

7 : **multicast** (sid, propose-short, e, u, π_i)

8 : DD.send[sid, e, propose-long] (long-msg, u)

Function buildBlock(e, cert, msg)

1 : parse cert $\rightarrow (e', u', \sigma)$, $B_{\text{last}} \leftarrow$ **global** BLKs$[u']$

2 : $B_{\text{new}} \leftarrow (e, \text{msg}, e', u', \sigma)$, $\mathbf{m} \leftarrow$ EC.Enc($B_{\text{new}}, n, n-f$)

3 : $u \leftarrow$ Acc.Eval($ak, \mathbf{m}$), **return** (B_{new}, u)

FIGURE 4.4: The PCSBB Epoch construction: Step 1-2.

Step 3 (Vote and redeliver): $T = 5 + \Delta_{\mathsf{MVBA}}$

$u^* \leftarrow \perp, \pi \leftarrow \perp$ // u^* is the accepted accumulator

receive $\{(\mathsf{sid}, \mathsf{propose\text{-}short}, e, u_j, \pi_j)\}_{j \in H_3}$, **for** $j \in H_3 \subset D$ **do**

  if $\mathsf{MTS.PartVrfy}(\mathsf{MTS}.vk_j, \pi_j, u_j) = 1$ **then** $\mathsf{acc}[u_j] \leftarrow \mathsf{acc}[u_j] \cup \{\pi_j\}$

  if $|\mathsf{acc}[u_j]| = n - f$ **then** $u^* \leftarrow u_j, \pi \leftarrow \mathsf{MTS.Combine}(\mathsf{CK}_{\mathcal{D}}, \mathsf{acc}[u_j], u_j)$

    multicast $(\mathsf{sid}, \mathsf{propose\text{-}accept}, e, u^*, \pi)$

    $\mathsf{DD.relay}[\mathsf{sid}, e, \mathsf{propose\text{-}long}](u^*)$

    // $T = 6 + \Delta_{\mathsf{MVBA}}$

    long-msg $\leftarrow \mathsf{DD.recover}[\mathsf{sid}, e, \mathsf{propose\text{-}long}]\ (u^*)$

    parse long-msg $\rightarrow B_e, B_e \rightarrow (e, \mathsf{msg}, e', u', \sigma)$

    $\mathsf{BLKs}[u^*] \leftarrow B_e$, update highest-cert by (e', u', σ) if necessary

    $\mathsf{DD.send}[\mathsf{sid}, e, \mathsf{redeliver\text{-}long}]\ (\mathsf{sid}, e, \text{long-msg}, u^*)$

  if $\pi \neq \perp \wedge B_e \neq \perp$ **then**

    if $\mathsf{validateBlock}(B_e, \text{highest-cert}) = 1$ **then**

      $\rho_i \leftarrow \mathsf{MTS.PartSign}(\mathsf{MTS}.sk_i, u^*)$, **multicast** $(\mathsf{sid}, \mathsf{vote}, e, u^*, \rho_i)$

Step 4 (Accept): $T = 6 + \Delta_{\mathsf{MVBA}}$

receive $\{(\mathsf{sid}, \mathsf{propose\text{-}accept}, e, u_j^*, \pi_j^*)\}_{j \in H_4}$, **for** $j \in H_4$ **do**

  if $u^* = \perp$ **then** $u^* \leftarrow u_j^*, \pi \leftarrow \pi_j^*$

  else if $u_j^* \neq u^* \wedge \mathsf{MTS.Vrfy}(\mathsf{VK}_{\mathcal{D}}, \pi_j^*, u_j^*) = 1$

    then multicast $(\mathsf{sid}, \mathsf{equivocate}, e, u^*, \pi, u_j^*, \pi_j^*)$

  if long-msg $= \perp$ **then**

    long-msg $\leftarrow \mathsf{DD.recover}[\mathsf{sid}, e, \mathsf{redeliver\text{-}long}]\ (u^*)$

    parse long-msg $\rightarrow (e, B_e), \mathsf{BLKs}[u^*] = B_e$

Step 5 (Commit): $T = 7 + \Delta_{\mathsf{MVBA}}$

receive $\{(\mathsf{sid}, \mathsf{vote}, e, u_j^*, \rho_j, \pi_j^*)\}_{j \in H_5}$, **for** $j \in H_5$ **do**

  if $u_j^* = u^* \wedge \mathsf{MTS.PartVrfy}(\mathsf{MTS.sk}_j, \rho_j, u^*) = 1$

    then votes $\leftarrow$ votes $\cup \{\rho_j\}$

  if $|\mathsf{votes}| = n - f$ **then** $\sigma \leftarrow \mathsf{MTS.Combine}(\mathcal{P}, \mathsf{votes}, u^*)$

    highest-cert $\leftarrow (e, u^*, \sigma), \mathsf{CERTs}[e] \leftarrow (e, u^*, \sigma)$

    multicast $(\mathsf{sid}, \mathsf{new\text{-}cert}, e, \text{highest-cert})$

    commit B_e and all uncommitted ancestor blocks

STATE' $\leftarrow (\mathsf{CERTs}, \mathsf{BLKs})$, **output** STATE'

FIGURE 4.5: The PCSBB Epoch construction: Step 3-5.

STEP 1: EPOCH INITIALIZATION. Except for the first epoch, all nodes should receive certificates shared by others and updates its highest-ranked block certificate accordingly. Each node also prepare to join in two data deliver DD instances that are to disperse long messages later in this epoch. At the end of this step, each party sends the highest-ranked certificate to its knowledge to every node in the leader consortium.

STEP 2: PROPOSE. Only nodes in the leader consortium execute this step. Each leader node receives certificates and updates its knowledge of the highest-ranked certificate. This ensures that honest nodes within the consortium acknowledge the highest-ranked certificate known to any honest party in the network.

Within the consortium, nodes invoke the intra-consortium consensus protocol LeaderAgreeCert, which ensures agreement on a certificate ranked higher than any certificate known to honest nodes in the previous epoch. Next, nodes construct the new block B_e using the agreed highest certificate. The block is encoded into n code words $\mathbf{m}$ via the erasure code algorithm EC.Enc, and its cryptographic accumulated value u is computed using Acc.Eval. Each node then partially signs u, producing a signature π_i, and proposes the new block by multicasting a propose-short message containing u and π_i.

Remarkably, our LeaderAgreeCert is based on MVBA, which can terminates within $\Delta_{\mathsf{MVBA}} + 3$ rounds. This step is where we leverage endemic coins.

STEP 3: VOTE AND REDELIVER. When each node receives $n - f$ propose-short messages that agree on the same accumulated value u^* and provides valid partial signatures, the node will accept the short proposal u^*, aggregate $n - f$ valid partial signatures into a threshold signature π to u^*, and announce its approval on u^* to others by multicasting propose-short message with π. When a node accepts u^*, it will do two more things before it actually votes for u^*. Firstly, it calls DD.relay[sid, e, propose-long] to allow actively relay data fragments corresponding to u^*. Secondly, when the long proposal is successfully recovered it updates the highest-ranked certificate accordingly and invokes DD.send[sid, e, redeliver-long] to ensure availability of long proposal among all honest nodes. Note that it is important to update the highest certificate otherwise the current node may refuse to vote a valid proposal since the highest-ranked certificate prior to the update cannot be extended by the proposal. Finally, it votes for u^* by multicasting a partial signature ρ_i.

STEP 4: ACCEPT. Nodes receive propose-accept messages and validate the corresponding proof π_j^*. When conflicting acceptance of two different blocks are identified, a node alerts others by multicasting the equivocating leader consortium threshold signatures on different accumulated values. If equivocation does not happen however the node has not accepted any proposal, it adopts other's acceptance over u^* and expects a successfully redeliver of the long proposal.

STEP 5: COMMIT. When a node receives $n - f$ valid votes towards its accepted u^*, it aggregates these votes into a block certificate and announces the new certificate to others. If no equivocation is detected 2 round (Δ) time after the node votes, it commits the block corresponding to u^* and move on to the next epoch. Otherwise, do not commit B and move on.

STEP EQUIVOCATION. If an honest party detects equivocation in Step 4, it multicasts the proof of equivocation at time $T = 6 + \Delta_{\mathsf{MVBA}}$. Hence at time $T = 7 + \Delta_{\mathsf{MVBA}}$ honest parties will receive the proof of equivocation and abort actions in epoch e. See Fig.4.6 (in Section 4.5.5) for the detailed actions.

We present the five steps of PCSBB.Epoch in Fig.4.4, Fig.4.5. We describe the details of the equivocation step in Fig.4.6.

Step Equivocation: $T = 7 + \Delta_{\mathsf{MVBA}}$

1 : **receive** $(\mathsf{sid}, \mathsf{equivocate}, e, u[1], \pi[1], u[2], \pi[2])$ **then**
2 : **if** $u[1] \neq u[2] \wedge \mathsf{MTS.Vrfy}(\mathsf{VK}_{\mathcal{D}}, \pi[1], u[1]) = 1 \wedge$
3 : $\mathsf{MTS.Vrfy}(\mathsf{VK}_{\mathcal{D}}, \pi[2], u[2]) = 1$ **then**
4 : **abort** any further actions and discard certificates in epoch e

FIGURE 4.6: The PCSBB Epoch construction: Step Equivocate.

REMARK 1. *In case an adversarial leader consortium does not lure honest parties to actually vote for equivocating blocks, equivocating proposals alone will not trigger a real equivocation. For example, if an adversarial lead consortium delays sending the proof of an equivocation to disrupt the protocol so that honest parties receive the equivocation later than time $T = 7 + \Delta_{MVBA}$, such proof will be disregarded and that epoch will not be treated as being equivocated, since a late equivocation is not a sufficient sign of that honest parties have accepted and voted for equivocating proposals.*

4.5.6 The Details of LeaderAgreeCert Intra-consortium Consensus

In the PCSBB, a critical step is to ensure the nodes within the same leader consortium have the same view of about the previous blocks, so that they can jointly act as a leader to propose a new block and help the network commit previous blocks. However, in Step 1, malicious nodes may send different certificates with different ranks to different consortium members, creating inconsistencies in their views.

We introduce an intra-consortium consensus LeaderAgreeCert to solve this issue. In LeaderAgreeCert, each $P_i \in \mathcal{D}$ provides the highest-ranked certificate cert_i to its knowledge as input, and will agree on a certificate cert, which is ranked higher than any certificate known to honest nodes in the previous

epoch. This component is used in Step 2 of our PCSBB.Epoch, and it invokes MVBA as a subroutine. LeaderAgreeCert terminates within $\Delta_{\mathsf{MVBA}} + 3$ rounds, incurring $O(\eta \log n\lambda + \eta^2 \log \eta\lambda)$-bit communication.

At a high level, a certificate with a valid threshold signature confirms it has a rank equal to or higher than the highest certificate received by an honest leader. Since all honest nodes multicast their highest-ranked certificates to all leaders, this ensures that the selected certificate is at least as high-ranked as those held by any honest node.

We present both the pseudocode in Fig. 4.7 and verbal description in the following.

LeaderAgreeCert$\langle \mathsf{sid}, e, \mathcal{D}, P_i(\mathsf{cert}_i)\rangle$

send $(\mathsf{sid}, e, \mathsf{request}, \mathsf{cert}_i)$ to all $P_j \in \mathcal{D}$, **receive** $\{(\mathsf{sid}, e, \mathsf{request}, \mathsf{cert}_j)\}_{j\in C_1}$

for $j \in C_1$ **do** **if** cert_j has an equal or a higher rank than cert_i **then**

$\gamma_{i,j} \leftarrow \mathsf{MTS.PartSign}(\mathsf{MTS}.sk_i, \mathsf{cert}_j)$, **send** $(\mathsf{sid}, e, \mathsf{signed}, \gamma_{i,j})$ to P_j

// assume these partial signatures have been verified already

receive $\{(\mathsf{sid}, e, \mathsf{signed}, \gamma_{j,i})\}_{j\in C_2}$

if $|C_2| \geq \lfloor |\mathcal{D}|/2 \rfloor + 1$ **then** $\alpha_i \leftarrow \mathsf{MTS.Combine}(\mathsf{CK}_{\mathcal{D}}, \{\gamma_{j,i} | j \in C_2\}, \mathsf{cert}_i)$

send $(\mathsf{sid}, e, \mathsf{tsig}, \alpha_i, \mathsf{cert}_i)$ to all $P_j \in \mathcal{D}$

receive $\mathsf{TIGs} = \{(\mathsf{sid}, e, \mathsf{tsig}, \alpha_j, \mathsf{cert2}_j)\}_{j\in C_3}$

compute $(\alpha, \mathsf{cert2})$ where cert2 is the highest-ranked in TSIGs and α is the corresponding valid threshold signature

// Let $Q((\cdot,\cdot))$ be a predicate attests the validity of (a, b) tuples where b is a threshold signature of a

$(\beta, \mathsf{cert3}) \leftarrow \mathsf{MVBA}\langle P_i((\alpha, \mathsf{cert2})), \mathsf{predicate} = Q\rangle$, **return** cert3

FIGURE 4.7: Intra-consortium consensus LeaderAgreeCert.

First, each member proposes its highest certificate to the group. Upon receiving a certificate of equal or higher rank, a node echoes a partially signed receipt. A node then aggregates valid partial signatures into a threshold signature, attesting to its proposal being approved by the majority of $\mathcal{D}$. An honest consortium must certify C' since it genuinely has a higher rank. Consortium members multicast their proposals and the corresponding proofs of approval within the group, locking onto the highest-ranked certificate with a valid proof. Finally, they invoke a randomized MVBA protocol using the locked certificate and proof, applying a predicate to validate the proof over the certificate.

Detailed Analysis about View Inconsistency within a Leader Consortium. Honest parties in the sender/leader consortium may have inconsistent views on the highest-ranked certificates within the network, and before that they must also ensure, by the time they propose a new block, they have received a certificate that has a rank equal to or higher than the highest-ranked certificate known to all honest parties by the end of last epoch. Suppose that an honest party P has the highest-ranked certificate C known to all honest parties by the end of Step 1 in epoch e. Then, since Step 1 incurs an all-to-all multicast to update the local highest certificate, C must have an equal or even a higher rank than the highest-ranked certificate C known to all honest parties by the end of epoch $e-1$. Furthermore, P must send C to all members of in epoch e's leader consortium $\mathcal{D}$. Therefore, by the beginning of Step 2, honest nodes within the sender consortium $\mathcal{D}$ should have received C and update their highest certificate. Though honest node may receive certificates with even higher rank than C from adversarial nodes, it is guaranteed that all honest nodes in the consortium observe some certificate that has a rank equal to or higher than the highest-ranked certificate known to all honest parties.

In the case where adversarial nodes aim to create conflict between honest nodes and send a certificate $C' \neq C$ selectively to honest consortium members assuming C' has never been seen by any honest nodes in the network nor has it been equivocated, we investigate three cases: (1) when C' has a strictly lower rank than C, honest nodes will not update its highest certificate to C' so distributing C' is meaningless; (2) when C' and C has an equal rank r, the corresponding block accumulated value u' and u must be voted by at least one honest party respectively in epoch r, thus there must be an equivocation in epoch r, since two conflicting accumulated values were accepted by honest parties before being voted. This contradicts the fact that C' has not been equivocated and never happens; (3) when C' has a higher rank than C, honest nodes in the consortium will receive C' and update their highest certificate from C to C'.

The inconsistent view issue only arises from the third case.

4.5.7 The Analysis

Efficiency Analysis. For round complexity, each epoch takes $8 + \Delta_{\mathsf{MVBA}}$ rounds. As an PCSBB instance consists of $2m$ epochs, it takes $8m + m\Delta_{\mathsf{MVBA}}$ rounds in total. As $\Delta_{\mathsf{MVBA}} = O(\kappa)$, the round complexity is $O(\kappa m)$.

Regarding the communication complexity, during each epoch, the intra-consortium consensus LeaderAgreeCert incurs $O(\eta\lambda \log n + \lambda\eta^2 \log \eta)$ bits in expectation (due to running MVBA) to agree on a certificate whose size is $(\lambda \log n)$. Other than that, each node multicasts a constant number of messages to the network, and the size of each message is bounded by $O(\ell/n + \lambda \log n)$. Therefore, the communication complexity for each epoch is $O(n\ell + \lambda n^2 \log n)$. So the overall communication complexity is $O(nm\ell + \lambda mn^2 \log n)$. A detailed complexity analysis is provided below.

Security Analysis. We establish the security in the following theorem.

THEOREM 2 (PCSBB Security). *Let* PCSBB[sid] *be an instance of PCSBB operating under the setup for the underlying cryptographic primitives and one-round coin flipping within honest-majority consortiums. Under the SXDH assumption and the security of the coin flipping instances within honest-majority consortiums,* PCSBB[sid] *guarantees termination, agreement, and validity in the random oracle model. Moreover,* PCSBB[sid] *remains secure even if the adversary queries the signing oracle of the MTS scheme for messages that do not have the prefix* sid.

SKETCH. First, under the SDXH assumption, the underlying MTS scheme is secure in the random oracle model. The security of coin flipping within $\mathcal{D}_i$ enables $\mathcal{D}_i$ to execute the MVBA protocol securely. In addition, even the adversary has the signing oracle access described in the theorem, the adversary cannot forge signatures for messages in the PCBSS instance. Under the above facts, in Lemma 11, we prove that an honest-majority $\mathcal{D}_i$ can always propose a block such that every honest node commits this block and its uncommitted ancestors, which ensures the validity; In lemma 13, we prove that al honest nodes, at the end of the protocol, have the same committed blocks created in the first m epochs, which ensures the agreement. □

Complexity Analysis. For communication complexity, we analyze the complexity by steps. In Step 1, it needs to send a certificate to all members of the leader consortium. Since the size of a certificate is $O(\lambda \log n)$ and the size of the consortium is η, Step 1 incurs $O(\lambda n \eta \log n)$ bit communication for all honest parties.

For Step 2, if an honest node P_i does not belong to the current leader consortium, it incurs no communication overhead. Otherwise, it executes subroutine LeaderAgreeCert, multicast a short proposal and invokes DD.send of a new block of size $O(\ell + \lambda \log n)$. Within the subroutine LeaderAgreeCert, it multicasts a certificate of size $O(\lambda \log n)$ within the consortium, echos partial signatures of size $O(\lambda)$ to consortium members, multicasts a message of size $O(\lambda \log \eta + \lambda \log n)$ within the consortium and finally invoke MVBA with an input size of $O(\lambda \log n)$. The MVBA has a cost of $O(\eta\lambda \log n + \lambda\eta^2 \log \eta)$ bits in expectation. For DD primitive, the overall execution incurs $O(n(\ell + \lambda \log n) + \lambda n^2 \log n)$ bit of total communication. In summary, Step 2 incurs $O(\eta^2 \lambda \log n + \eta\lambda \log n + \lambda\eta^2 \log \eta + \eta n\lambda + n\ell + \lambda n^2 \log n) = O(n\ell + \lambda n^2 \log n)$ bits communication.

For Step 3, 4 and 5, each honest party at most multicasts message of total length of $O(\lambda \log n)$ bits, adding another $O(\lambda n^2 \log n)$ to the overall communication. DD primitive is executed one more time with the same input size, so it does not affect the asymptotic complexity.

Since the equivocation step incurs no extra communication, the communication complexity of an epoch is $O(n\ell + \lambda n^2 \log n)$. Therefore the PCSBB total communication complexity becomes $O(mn\ell + \lambda mn^2 \log n)$ since it runs exactly $2m$ epochs.

For round complexity, concretely it takes $8 + \Delta_{\mathsf{MVBA}}$ rounds to complete an epoch, hence the overall round complexity is $O(\kappa m)$.

4.5.8 Security Proof for PCSBB

Moreover, we formulate the security of PCSBB by the following lemmas. Note that we say a block is committed *directly* in epoch e if it is committed as a result of being proposed in epoch e. Otherwise, we say a block B is committed *indirectly* if it is a result of *directly* committing a block B' that extends B.

LEMMA 7. *No equivocating block certificate can be generated for two conflicting block proposed in the same epoch.*

PROOF. Assume equivocating blocks B and B' have been certified in epoch e. This implies at least one honest node P votes for B by its accumulated value u and another honest party Q votes for B' by its accumulated value u' at time $T = 6 + \Delta_{\mathsf{MVBA}}$. Moreover, P (Q) must accept the short proposal of B (B') and forward it to others at time $T = 5 + \Delta_{\mathsf{MVBA}}$. Hence, by time $T = T = 6 + \Delta_{\mathsf{MVBA}}$, both P and Q will detect an equivocation, and multicast the evidence immediately without voting any proposals. This contradicts the hypothesis that the P and Q votes in epoch e. □

LEMMA 8. *If an honest node commits a block B in epoch e, then (i) an equivocating block certificate does not exist in epoch e (ii) every honest node receives a certificate C corresponding to B before entering epoch $e + 1$.*

PROOF. We first discuss the case where a block is *directly* committed. An honest node P *directly* commits B at time $T = 7 + \Delta_{\mathsf{MVBA}}$ in epoch e, then it must successfully (1) recover the original content of B and vote for it by the accumulated value u at time $T = 6 + \Delta_{\mathsf{MVBA}}$; (2) accept the short proposal of B and forward it to others at time $T = 5 + \Delta_{\mathsf{MVBA}}$.

For (i), it follows Lemma 7.

For (ii), we know that P would multicast the certificate corresponding to B at time $T = 7 + \Delta_{\mathsf{MVBA}}$. Such a certificate will arrive all honest party by time $T = 8 + \Delta_{\mathsf{MVBA}}$ before the start of next epoch.

In case an honest party P *directly* commits block B and *indirectly* commits block B' such that B extends B', we show (i) and (ii) holds for B' respectively. Suppose B' is proposed in epoch e' and the corresponding certificate is $\mathsf{cert}_{B'}$. Similar to the arguments in the *directly* committing case (i): assume an equivocating certificates were formed in epoch e', then two honest parties will vote for distinct proposals. However, if two honest parties votes for two distinct proposals, they must have received equivocating proposal-accept messages before they vote, hence they will not vote. This already creates a contradiction.

For (ii), observe that the certificate $\mathsf{cert}_{B'}$ is included as a part of the descendant block B'' where B'' is also *indirectly* committed in epoch e or specifically $B = B''$. In both cases B'' has a corresponding certificate $\mathsf{cert}_{B''}$. This implies that by the end of epoch e, at least one honest node has voted for a proposal of B'' and thus it must have received the full content of block B''. Hence, it has received $\mathsf{cert}_{B'}$. □

LEMMA 9. *If an honest node directly commits B in epoch e, then any certified block C that ranks higher than B must extend B.*

PROOF. We induct on epoch $f > e$ where C is proposed in epoch f. For an epoch f, we prove that if C exists then it must extend B. For the base $f = e + 1$, observe that B is *directly* committed by an honest node P in epoch e. By Lemma 8, we know that all honest nodes receives the certificate cert_B corresponding to B by the start of epoch $f = e+1$ without equivocation. If a certificate cert_C is formed later, then at least one honest node votes for it in epoch f. Since cert_B is the highest-ranked certificate known to all honest parties in epoch f, honest node will only vote for a block that extends B. Hence, C extends B.

For induction step, assume that any certified block that ranks higher than e but strictly lower than f must extend B. We prove that a certified block C with rank f must extend B. Again, the necessary condition for C to be certified is that at least one honest party P votes for C in epoch $f > e$. Suppose $\mathsf{cert}_{e'}$ is the highest ranked certificate that P has and certified block B' in epoch $e' \geq e$. Since P is honest, C must extend B'. Also, P only updates its highest ranked certificate $\mathsf{highest\text{-}cert}$ when receiving a strictly higher ranked certificate. This implies the rank of $\mathsf{highest\text{-}cert}$ grows monotonically, thus $e \leq e' < f$. Therefore, by the induction hypothesis, block B' extends B. Since block C extends B', block C must extend B. □

LEMMA 10. *Honest nodes do not commit conflicting blocks for any epoch e.*

PROOF. Suppose for the sake of contradiction that two distinct block B_1 and B_2 are committed and they are both proposed in epoch e_0. Consider B_1 is committed as a result of block C_1 being *directly* committed in epoch $e_1 \geq e_0$ and B_2 is committed as a result of C_2 being *directly* committed in epoch $e_2 \geq e_0$. This implies that C_1 extends B_1 and C_2 extends B_2. Without loss of generality, assume $e_0 \leq e_1 < e_2$, that is C_1 is *directly* committed strictly earlier than C_2. By Lemma 9, C_2 extends C_1. Therefore C_2 extends both B_1 and B_2. Since both B_1 and B_2 are committed in epoch e_0, $B_1 = B_2$. □

LEMMA 11. *In epoch e, when an honest consortium-sender group $\mathcal{D}$ is the leader consortium and honest nodes in $\mathcal{D}$ share common input msg, the consortium-sender group proposes non-equivocating block B that includes msg and the rank of B is higher than the highest ranked certificate known to all honest parties before epoch e.*

PROOF. Suppose an honest party P has the highest-ranked certificate C_0 known to all honest nodes by the end of epoch $e-1$. We show B has an increasing rank compared with C_0. Following the PCSBB protocol, P receives highest-ranked certificate from other nodes and update its local record accordingly. Therefore P may receives C_1 which has a higher rank C_0 but is not known to any honest nodes before and update its highest ranked certificate to C_1. Otherwise, $C_1 = C_0$ and C_1 still ranks equally to or higher than C_0. Then, P will send C_1 to all nodes in the consortium group by the end of Step 1 in epoch e. By the start of Step 2, consortium members P_i receive certificates and may update their local highest certificate, denoted as $C_{i,2}$ after the update. Since P is honest, an honest node P_i in $\mathcal{D}$ must have received C_1. Again, P_i could receive a certificate that is even higher than C_1 (and unknown to any honest parties before this step), but regardless P_i will have a $C_{i,2}$ ranking equally to or higher than C_1. Thus, any honest node P_i in $\mathcal{D}$ has $C_{i,2}$ ranking equally to or higher than C_0. Honest consortium nodes continues to execute the protocol and call subroutine LeaderAgreeCert with $C_{i,2}$ as input. Honest consortium member first proposes its $C_{i,2}$ to others. When receiving a certificate $C_{j,2}$, an honest party echoes a partially signed receipt if $C_{j,2}$ ranks equally to or higher than $C_{i,2}$. When node P_i receives enough valid partial signatures, it aggregate them into a threshold signature attesting an approval from the majority of $\mathcal{D}$. Suppose honest party Q has the highest-ranked certificate C_3 known to all honest nodes in $\mathcal{D}$ before proposing it in subroutine LeaderAgreeCert. Then C_3 is highest ranked certificate among all $C_{i,2}$ for honest nodes and C_3 must rank equally to or higher than any $C_{i,2}$. Hence, all honest nodes must vote for C_3. Q obtains a quorum proof α_Q that approves C_3 and shares C_3 and α_Q with all consortium sender nodes. By line 7 in the subroutine, an honest node P_i in $\mathcal{D}$ will compute $(\alpha_i, \mathsf{cert2}_i)$ where $\mathsf{cert2}_i$ is the highest ranked certificate known to P_i and α_i is a valid quorum proof. Due to the existence of party Q and (α_Q, C_3), an honest node will obtain meaningful $(\alpha_i, \mathsf{cert2}_i)$ pair where the rank of $\mathsf{cert2}_i$ is at least the rank of C_3 and α_i is indeed a valid quorum proof. Finally, nodes input $(\alpha_i, \mathsf{cert2}_i)$ pair into a randomized MVBA primitive. Since the inputs from honest parties all satisfy the predicate, MVBA will terminates and outputs a common certificate cert3 for all honest parties where the rank of cert3 is at least the rank of C_3, hence C_0.

Since all honest nodes in the honest consortium $\mathcal{D}$ has the common input msg and agrees on the same highest ranked certificate cert3, they can build the same block B and propose it to the network. Since the sender consortium has an honest majority, no equivocating proposal can be generated. □

LEMMA 12. *All honest nodes commit* msg *in epoch* e *when an honest consortium-sender group* $\mathcal{D}$ *is the leader consortium in epoch* e *and honest nodes in* $\mathcal{D}$ *share common input* msg.

PROOF. Following Lemma 11, the honest sender consortium will form an unanimous propose for block B that includes msg. At time $T = 6 + \Delta_{\mathsf{MVBA}}$, all honest parties has received non-equivocating propose-accept messages, recovered block B in the long format and is ready to vote. Note that cert3 is also included in B. Since cert3 has a increasing rank compared with C_0, all honest party must update it highest certificate to cert3 (Line 14, Step 3) or otherwise they have a certificate of the same rank. This implies B extends the local highest certificate for all honest parties, by Lemma 7, honest nodes will vote the proposal of B. At time $T = 7 + \Delta_{\mathsf{MVBA}}$, an honest parties will receive enough votes for B and form a block certificate of B, multicast the certificate to the network, and commit B and all its uncommitted ancestor blocks. □

LEMMA 13. *All honest nodes committed the same blocks created in the first m epochs by the end of epoch $2m$, assuming existence of an honest consortium-sender group.*

PROOF. Since the leader consortium rotates in a round-robin manner, by pigeonhole principle, there must be an honest consortium leader elected in epoch $e \in [m + 1, 2m]$. Following Lemma 12, by the end of epoch e, all honest parties have committed the same blocks created in the first m epochs. □

4.6 Near-Optimal DKG for One-Round Coin Flipping

In this section, we present a DKG protocol with $\widetilde{O}(n^2)$ communication complexity and $\widetilde{O}(1)$ rounds, for the one-round coin flipping protocol, e.g., (Bacho et al., 2024).

4.6.1 The Protocol

Deterministic Recursive Partition. In our protocol, the network $\mathcal{P}$ is recursively partitioned into smaller groups according to an arbitrary deterministic rule. Specifically, at the first layer of recursion, $\mathcal{P}$ is divided into two consortiums, $\mathcal{D}^{(1,0)}$ and $\mathcal{D}^{(1,1)}$, such that $\mathcal{P} = \mathcal{D}^{(1,0)} \cup \mathcal{D}^{(1,1)}$, with their sizes approximately equal, i.e., $\lfloor \frac{|\mathcal{P}|}{2} \rfloor$ and $\lceil \frac{|\mathcal{P}|}{2} \rceil$. At the second layer of recursion, each consortium $\mathcal{D}^{(1,k)}$ is further partitioned into $\mathcal{D}^{(2,2k)}$ and $\mathcal{D}^{(2,2k+1)}$. This process continues until the d-th layer, where each consortium $\mathcal{D}^{(d,k)}$ contains a small constant number of nodes. For convenience, we denote the entire network as $\mathcal{D}^{(0,0)} := \mathcal{P}$.

This deterministic recursive partition ensures that if $\mathcal{P}$ has an honest majority, there exists a *forever-honest-majority path*. Specifically, let the adversary corrupt up to $\lfloor \frac{|\mathcal{P}|-1}{2} \rfloor$ nodes, forming the set $\mathcal{C}$. Then, there exists a path $(k_1, \ldots, k_d)$ through the recursive partition tree such that:

$$\mathcal{D}^{(d,k_d)} \subset \mathcal{D}^{(d-1,k_{d-1})} \cdots \subset \mathcal{D}^{(1,k_1)}, \quad \text{and} \quad |\mathcal{D}^{(r,k_r)} \cap \mathcal{C}| \leq \lfloor \frac{|\mathcal{D}^{(r,k_r)}| - 1}{2} \rfloor, \forall r \in [d].$$

This follows directly from the pigeonhole principle: since $\mathcal{P}$ has an honest majority, at least one of the top-level consortiums $\mathcal{D}^{(1,k_1)}$ must also have an honest majority. Similarly, at each subsequent level of the partition, one of the sub-consortiums will retain an honest majority.

Building Blocks. We use the following building blocks.

BYZANTINE BROADCAST FOR BASE RECURSION. Since a bottom-level consortium $\mathcal{D}^{(d,k_d)}$ only contains a small number of nodes, it can execute "inefficient" protocols. Specifically, we use an adaptively and concurrently secure deterministic Byzantine broadcast protocol BB (c.f. Section 2). In particular, we denote the execution of an instance of BB as $\mathsf{BB}[\mathsf{sid}|d|k_d|i]\langle P_i(v), \mathcal{D}^{(d,k_d)} \rangle \to v$, where $(\mathsf{sid}|d|k_d|i)$ is the unique identifier of this instance, and P_i is the sender who broadcasts the message v to the receiver group $\mathcal{D}^{(d,k_d)}$.

PCSBB. We use the PCSBB protocol introduced in Section 4.5. Looking ahead, the protocol PCSBB will be run within every consortium $\mathcal{D}^{(r,k_r)}$ with $r \neq d$, while the two sub-consortiums $\mathcal{D}^{(r+1,k'_{r+1})}$ and $\mathcal{D}^{(r+1,k''_{r+1})}$ serve as the consortiums of the PCSBB instances. We denote the execution of an instance of PCSBB as

$$\mathsf{PCSBB}[\mathsf{sid}|r|k_r]\langle \mathcal{D}^{(r+1,k'_{r+1})}(v'), \mathcal{D}^{(r+1,k''_{r+1})}(v'')\rangle \rightarrow \mathcal{V},$$

where $(\mathsf{sid}|r|k_r)$ is the unique identifier of the instance executed within $\mathcal{D}^{(r,k_r)}$, and $\mathcal{D}^{(r+1,k'_{r+1})}$ and $\mathcal{D}^{(r+1,k''_{r+1})}$ provides v' and v'' as inputs, respectively; $\mathcal{V}$ is the output.

Running PCSBB within $\mathcal{D}^{(r,k_r)}$ requires that the sub-consortiums $\mathcal{D}^{(r+1,k'_{r+1})}$ and $\mathcal{D}^{(r+1,k''_{r+1})}$ provide *endemic* coins. In our protocol, each sub-consortium will run the one-round coin flipping protocol from (Bacho et al., 2024) (cf. Appendix B.1.2) to provide the endemic coins needed by the PCSBB, while the setup required by the coin flipping protocol will be provided by our protocol.

APVSS. We use an aggregatable publicly verifiable secret sharing (APVSS) scheme PVSS from (Bacho and Loss, 2023b; Bacho et al., 2024). The full definition and construction of this scheme are recalled in Section 2.2 and Appendix B.1.1. In our protocol, we explicitly use the following algorithm of the APVSS scheme.

- $\mathsf{PVSS.Deal}((ek_j)_{j\in\mathcal{D}}, t, P_i) \rightarrow T$. It creates a PVSS transcript T under the encryption key set $(ek_j)_{j\in\mathcal{D}}$, while t is the specified threshold, such that using $t+1$ corresponding decryption keys can recover the secret of this transcript.
- $\mathsf{PVSS.Agg}(\{T_j\}_{j\in S}) \rightarrow T$. It aggregates a set of PVSS transcripts (under the same set of encryption keys) into one compact transcript T.
- $\mathsf{PVSS.Vrfy}(T) \rightarrow 0$ or 1. It validates a transcript.
- $\mathsf{PVSS.Dec}(dk_i, T) \rightarrow sk_i$. It decrypts a PVSS transcript T using a decryption key dk_i and yields sk_i, which is a secret share of the secret of this transcript.
- $\mathsf{PVSS.PubDriv}(T)$. It derives the public key and public key shares from T.

Overview. At a high level, our DKG protocol RDKG proceeds in two steps. First, all nodes in $\mathcal{P}$ agree on an aggregated PVSS transcript, denoted by $T^{(0)}$. Then, each honest node $P_i \in \mathcal{P}$ derives the public keys $pk, (pk_j)_{j\in[n]}$ from $T^{(0)}$ and obtains its secret key sk_i by decrypting $T^{(0)}$. This secret key serves as the output of P_i in the DKG execution.

For security, $T^{(0)}$ must be an aggregation of PVSS transcripts, with at least one transcript generated by a forever-honest node, ensuring that the secret key remains unknown to the adversary. Our protocol adopts a divide-and-conquer strategy to achieve this. Specifically, the top-level consortiums $\mathcal{D}^{(1,0)}$ and $\mathcal{D}^{(1,1)}$ each generate and agree on transcripts $T^{(0)}_{(1,0)}$ and $T^{(0)}_{(1,1)}$, respectively. The final transcript $T^{(0)}$ is obtained by aggregating these two. This strategy is similar to that employed in (Bacho et al., 2024; Feng et al., 2024a). Honest nodes in $\mathcal{P}$ must first agree on both $T^{(0)}_{(1,0)}$ and $T^{(0)}_{(1,1)}$ to ensure they can consistently agree on $T^{(0)}$. In (Bacho et al., 2024), this agreement is achieved through two parallel instances of the CSBB protocol (Feng et al., 2024a), which relies on a deterministic Byzantine Agreement protocol, incurring $O(n)$ rounds. Our protocol improves the round complexity by replacing the CSBB instances with our PCSBB protocol, which requires only $O(\kappa)$ rounds.

However, the improved round efficiency of PCSBB relies on efficient generation of endemic coins within each consortium. Therefore, in addition to generating $T^{(0)}_{(1,k)}$ (for $k \in \{0,1\}$), each top-level consortium $\mathcal{D}^{(1,k)}$ must also establish a setup for the coin-flipping protocol (we specifically consider the protocol from (Bacho et al., 2024)) within the consortium. This is achieved by generating and agreeing on another PVSS transcript, $T^{(1)}$, from which honest nodes in $\mathcal{D}^{(1,k)}$ can obtain their secret key shares. Consequently, each top-level consortium $\mathcal{D}^{(1,k)}$ needs to generate and agree on both $T^{(0)}_{(1,k)}$ and $T^{(1)}$. As the recursion proceeds, a consortium $\mathcal{D}^{(r,k_r)}$ at the r-th layer must generate all PVSS transcripts required by its parent consortium, denoted by $T^{(0)}_{(r,k_r)}, \ldots, T^{(r-1)}_{(r,k_r)}$, along with a transcript $T^{(r)}$ to establish the setup for its own endemic coin-flipping protocol.

At each layer $r \in [0,d]$, an honest node P_i must decrypt the PVSS transcript $T^{(r)}$ and use the decryption result to perform coin-flipping. For security, we require that transcripts $T^{(r)}$ and $T^{(r')}$ for $r \neq r'$ be generated with respect to different sets of encryption public keys, i.e., $(ek_i^{(r)})$ and $(ek_i^{(r')})$, such that $ek_i^{(r)} \neq ek_i^{(r')}$ for the same honest node P_i[7]. Therefore, each node needs to generate $d+1 = O(\log n)$ encryption public keys.

Protocol Description. We present the protocol RDKG in Fig.4.9. It requires a setup phase in which (1) the public parameters for PVSS, PCSBB, and BB are honestly generated and available to all nodes, and (2) each $P_i \in \mathcal{P}$ runs $(ek_i^{(r)}, dk_i^{(r)}) \leftarrow \mathsf{PVSS.KeyGen}(1^\lambda)$, for $r \in [0,d]$ and generates key pairs for PCSBB and BB, such that all public keys are available to all nodes. Under the setup, the protocol executes as follows:

[7]If the same public key is used across layers, the decryption procedure for establishing endemic coins might act as a decryption oracle, which may compromise security.

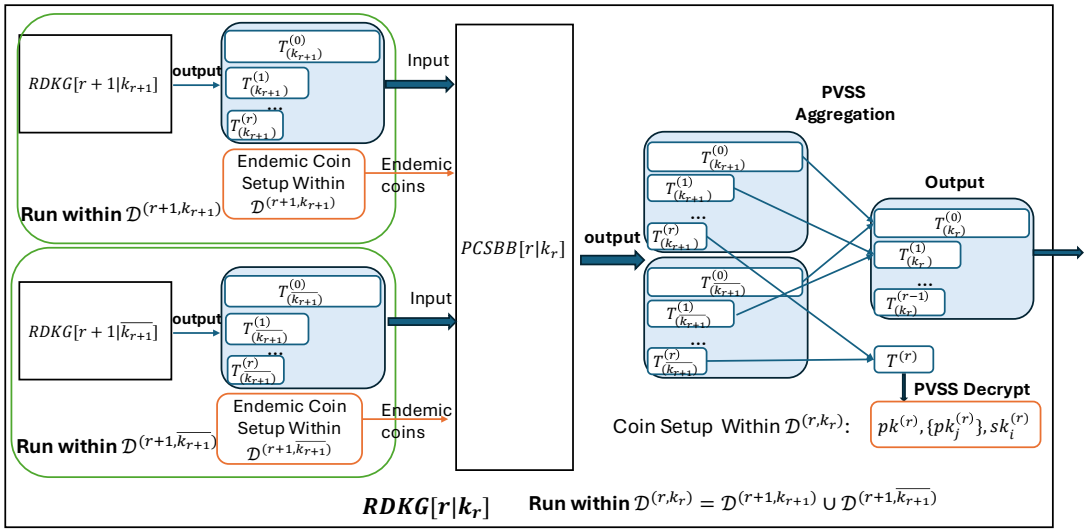


FIGURE 4.8: The execution flow of RDKG at r-th recursion.

- **Base case**. For the d-th recursive step, each consortium $\mathcal{D}^{(d,k_d)}$ executes $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$. Specifically, each node P_i compute $d+1$ PVSS transcripts in the base case, each of which encrypts a random secret to be shared among P_i's associated groups $\mathcal{D}^{(0,0)}$, $\mathcal{D}^{(1,k_1)}$, $\cdots$ and $\mathcal{D}^{(d,k_d)}$, respectively. Then, P_i broadcasts these $d+1$ PVSS transcripts across its smallest belonging group $\mathcal{D}^{d,k^{(d)}}$, then waits for such broadcasted PVSS messages from all nodes in $\mathcal{D}^{(d,k^{(d)})}$. Then, the valid transcripts for the current consortium $\mathcal{D}^{(d,k^{(d)})}$ are aggregated and decrypted, providing a setup for the endemic coin flipping within $\mathcal{D}^{(d,k_d)}$. The remaining PVSS transcripts are aggregated accordingly, forming a list $\mathbb{T}^{(d,k_d)}$.
- **Recursive execution**. At the r-th recursion, each consortium $\mathcal{D}^{(r,k_r)}$ executes $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$, whose execution flow is outlined in Fig.4.8. Specifically, the two sub-consortiums, $\mathcal{D}^{(r+1,k_{r+1})}$ and $\mathcal{D}^{(r+1,\overline{k_{r+1}})}$, run $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$ and $\mathsf{RDKG}[\mathsf{sid}|r+1|\overline{k_{r+1}}]$, respectively, which establish the setup for endemic coin flipping within each sub-consortium, and also produce two lists of PVSS transcripts, $\mathbb{T}^{(r+1,k_{r+1})}$ and $\mathbb{T}^{(r+1,\overline{k_{r+1}})}$. Then, $\mathcal{D}^{(r+1,k_{r+1})}$ and $\mathcal{D}^{(r+1,\overline{k_{r+1}})}$ jointly execute $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$ to broadcast the two lists of PVSS transcripts. Once the PCSBB outputs, $P_i \in \mathcal{D}^{(r,k_r)}$ can aggregate its results to obtain a list of further aggregated PVSS transcripts. The aggregate PVSS transcript for $\mathcal{D}^{(r,k_r)}$ can be used to establish a setup for the endemic coin flipping within $\mathcal{D}^{(r,k_r)}$, which can be used by the $(r-1)$-th recursion to set up its larger PCSBB. And the remaining $(r-1)$ aggregate PVSS transcripts will be passed to the $(r-1)$-th recursive execution, serving as the input of this larger PCSBB.

RDKG[sid]. Every $P_i \in \mathcal{D}^{(d,k_d)} \subset \cdots \subset \mathcal{D}^{(1,k_1)} \subset \mathcal{P}$ performs the following steps:

if sid doesn't contain the fields of r and k, $r \leftarrow 0$; **else** $r \leftarrow \mathsf{sid}.r$ and $k \leftarrow \mathsf{sid}.k$

if $r = d$ **then** **// Base case**

$\mathsf{PVSS.Deal}((ek_j^{(0)})_{j\in[n]}, \lfloor \frac{n}{2} \rfloor, P_i) \rightarrow T_i^{(0)}, \cdots,$

$\mathsf{PVSS.Deal}((ek_j^{(d)})_{j\in\mathcal{D}^{(d,k_d)}}, \lfloor \frac{|\mathcal{D}^{(d,k_d)}|}{2} \rfloor, P_i) \rightarrow T_i^{(d)}$

$\mathsf{BB}[\mathsf{sid}|d|k_d|i]\langle P_i((T_i^{(d)}, \cdots T_i^{(0)})), \mathcal{D}^{(d,k_d)}\rangle \rightarrow ((T_i^{(d)}, \cdots T_i^{(0)}))$

wait for receiving $\{(T_j^{(d)}, \cdots T_j^{(0)})\}_{j\in\mathcal{D}^{(d,k_d)}}$ from BBs until next step

$S \leftarrow \emptyset$; **for** $j \in \mathcal{D}^{(d,k_d)}$ **if** $\mathsf{PVSS.Vrfy}(T_j^{(z)}) = 1, \forall z \in [0,d]$ **then** $S \leftarrow S \cup \{j\}$

$\mathbb{T}^{(d,k_d)} \leftarrow \{\mathsf{PVSS.Agg}(\{T_j^{(d-1)}\}_{j\in S}), \cdots, \mathsf{PVSS.Agg}(\{T_j^{(0)}\}_{j\in S})\}$

$T^{(d)} \leftarrow \mathsf{PVSS.Agg}(\{T_j^{(d)}\}_{j\in S})$

$\mathsf{PVSS.Dec}(dk_i^{(d)}, T^{(d)}) \rightarrow sk_i^{(d)}, \mathsf{PVSS.PubDriv}(T^{(d)}) \rightarrow pk^{(d)}, (pk_j^{(d)})_{j\in\mathcal{D}^{(d,k_d)}}$

output $sk_i^{(d)}, pk^{(d)}, (pk_i^{(d)})_{j\in\mathcal{D}^{(d,k_d)}}, \mathbb{T}^{(d,k_d)}$

if $r \in \{0, 1, \cdots, d-1\}$ **// Recursive cases**

//RDKG within $\mathcal{D}^{(r+1,k_{r+1})}$, the output except $\mathbb{T}_i^{(r)}$ sets up the endemic coin

$\mathsf{RDKG}[\mathsf{sid}|(r+1)|k_{r+1}]\langle\mathcal{D}^{(r+1,k_{r+1})}\rangle \rightarrow$

$sk_i^{(r+1)}, pk^{(r+1)}, (pk_j^{(r+1)})_{j\in\mathcal{D}^{(r+1,k_{r+1})}}, \mathbb{T}^{(r+1,k_{r+1})}$

//The extra commitment setup for the endemic coin

$\mathsf{Coin.ComSetup}[\mathsf{sid}|(r+1)|k_{r+1}]\langle\mathcal{D}^{(r+1,k_{r+1})}\rangle$

//Broadcast $\mathbb{T}^{(r,k_{r+1})}$ to the entire $\mathcal{D}^{(r,k_r)} = \mathcal{D}^{(r+1,k_{r+1})} \cup \mathcal{D}^{(r+1,\overline{k_{r+1}})}$ via PCSBB

$\mathsf{PCSBB}[\mathsf{sid}|r|k_r]\langle\{\mathcal{D}^{(r+1,k_{r+1})}(\mathbb{T}^{(r+1,k_{r+1})}), \mathcal{D}^{(r+1,\overline{k_{r+1}})}(\mathbb{T}^{(r+1,\overline{k_{r+1}})})\rangle$

$\rightarrow \{\mathbb{T}^{(r+1,k_{r+1})}, \mathbb{T}^{(r+1,\overline{k_{r+1}})}\}$

//Process the output of PCSBB

parse $\mathbb{T}^{(r+1,k_{r+1})} := \{T_{k_{r+1}}^{(r)}, \cdots, T_{k_{r+1}}^{(0)}\}$ and $\mathbb{T}^{(r+1,\overline{k_{r+1}})} := \{T_{\overline{k_{r+1}}}^{(r)}, \cdots, T_{\overline{k_{r+1}}}^{(0)}\}$

$S \leftarrow \emptyset$; **for** $j \in \{k_{r+1}, \overline{k_{r+1}}\}$ **if** $\mathsf{PVSS.Vrfy}(T_j^{(z)}) = 1, \forall z \in [0,r]$ **then** $S \leftarrow S \cup \{j\}$

if $r > 0$ **then** $\mathbb{T}^{(r,k_r)} \leftarrow \{\mathsf{PVSS.Agg}(\{T_j^{(r-1)}\}_{j\in S}, \cdots, \mathsf{PVSS.Agg}(\{T_j^{(0)}\}_{j\in S}))\}$

$T^{(r)} \leftarrow \mathsf{PVSS.Agg}(\{T_j^{(r)}\}_{j\in S})$

$\mathsf{PVSS.Dec}(dk_i^{(r)}, T^{(r)}) \rightarrow sk_i^{(r)}, \mathsf{PVSS.PubDriv}(T^{(r)}) \rightarrow pk^{(r)}, (pk_j^{(r)})_{j\in\mathcal{D}^{(r,k)}}$

output $sk_i^{(r)}, pk^{(r)}, (pk_j^{(r)})_{j\in\mathcal{D}^{(r,k)}}$, and if $r > 0$, additionally output $\mathbb{T}^{(r,k_r)}$

FIGURE 4.9: A quasi-constant round recursive DKG (RDKG) that produces group-element secrets for establishing an adaptively-secure one-round coin flipping protocol (Bacho et al., 2024).

4.6.2 The Analysis

Efficiency Analysis. For communication complexity, in the base-case, an honest node P_i needs to broadcast $d+1$ PVSS transcripts $\{T_i^(0), \cdots, T_i^{(d)}\}$ to the consortium $\mathcal{D}^{(d,k_d)}$, while the size of $T_i^{(r)}$ is $O(n_r\lambda)$, where n_r denotes the size of $\mathcal{D}^{(r,k_r)}$, which is $\frac{n}{2^r}$. Thus, the total size of all $d+1$ transcripts is $O(\sum_{r\in(0,d)} n_r\lambda) = O(n\lambda)$. Recall that the underlying BB incurs $O(\eta\ell + \lambda\eta^2 \log \eta)$ bits when broadcasting ℓ bits to η nodes. Therefore, it incurs $O(n_d n\lambda + n_d^2 \log n_d \lambda)$-bit communication cost to broadcast these PVSS transcripts. Hence, the communication cost of each instance $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$, denoted by Cost_d, is $O(n_d^2 n\lambda + n_d^3 \log n_d \lambda)$ bits.

In the r-th recursion, the communication cost of $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$, denoted by Cost_r, is the sum of communication cost of $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$, $\mathsf{RDKG}[\mathsf{sid}|r+1|\overline{k_{r+1}}]$, and $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$. According to Theorem.2, the communication cost of $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$ is $O(\lambda n_r^2 \log(n_r) + n_r n\lambda)$, as the input size is $O(n\lambda)$. Thus,

$$\mathsf{Cost}_r = 2 \cdot \mathsf{Cost}_{r+1} + O(\lambda n_r^2 \log(n_r) + n_r n\lambda).$$

Thus, as $O(\frac{n}{2^d}) = O(1)$, and $n_r = \frac{n}{2^r}$, the communication cost of $\mathsf{RDKG}[\mathsf{sid}]$ is

$$\begin{aligned}
\mathsf{Cost}_0 =& 2 \cdot \mathsf{Cost}_1 + O(\lambda n^2 \log(n) + n^2\lambda) \\
=& 2 \cdot (2 \cdot \mathsf{Cost}_2 + O(\lambda n_1^2 \log(n_1) + n_1 n\lambda)) + O(\lambda n^2 \log(n) + n^2\lambda) \\
& \cdots \\
=& 2^d \mathsf{Cost}_d + \sum_{r\in[0,d]} \left(2^r \cdot O(\lambda n_r^2 \log(n_r) + n_r n\lambda)\right) = O(n^2 \log n\lambda).
\end{aligned}$$

For round complexity, we first decide the duration of epochs in all underlying PCSBB protocols as $O(\kappa)$ rounds, ensuring that all concurrent randomized PCSBB protocols can terminate within the time with all but negligible probability. Consequently, every recursive step in our protocol (including the base case and all other step involving PCSBB) has $O(\kappa)$ rounds, and, therefore, the round complexity of our recursive protocol is $\sum_{r=0}^{\log n} O(\kappa) = O(\kappa \log n)$.

Security Analysis. We establish the following theorem for our RDKG protocol.

THEOREM 3. *Under the SXDH assumption and the co-OMDL assumption, the one-round coin flipping protocol from (Bacho et al., 2024) (recalled in Appendix B.1.2) using* RDKG *as its DKG setup satisfies*

the availability, agreement, bias-resistance, and unpredictability in the AGM and ROM, against any PPT adversary which can adaptively corrupt up to $\lfloor \frac{n-1}{2} \rfloor$ *nodes.*

In the following, we sketch our security proof through several key steps, with the detailed proof presented in Section 4.6.3.

STEP 1: GENERALIZING THE SECURITY PROOF FROM (BACHO ET AL., 2024). Our objective is to prove the security of the coin-flipping protocol from (Bacho et al., 2024) when using our RDKG protocol as the setup. Ideally, if the coin-flipping protocol has already been proven secure under a DKG with certain security properties, we could establish security by simply demonstrating that RDKG satisfies these required properties. However, (Bacho et al., 2024) provides a "holistic" security proof tied specifically to their DKG setup, making it necessary for us to delve into the details of the coin-flipping protocol to analyze its security.

Fortunately, we observe that the security proof in (Bacho et al., 2024) can be generalized to accommodate a broader class of DKG protocols. At a high level, this class consists of DKG protocols that are based on the APVSS scheme from (Bacho and Loss, 2023b) and share the following structure: (1) The network, through any mechanism, agrees on a valid PVSS transcript, which is an aggregation of a set of valid PVSS transcripts, containing at least one generated by a forever-honest node. (2) Each honest node decrypts this aggregated transcript to obtain its secret key share.

We formalize this class of protocols through an *abstract* DKG (described in Fig. 4.10 in Section 4.6.3). In Lemma 14, we show that, under the co-OMDL assumption (recalled in Definition 9), the coin-flipping protocol is secure under any DKG that conforms to this abstract model, in the AGM and the ROM.

STEP 2: RDKG "IMPLEMENTS" THE ABSTRACT DKG. Next, we demonstrate that RDKG serves as an "implementation" of the abstract DKG. Specifically, we show that an adversary $\mathcal{S}$ against the abstract DKG can perfectly simulate an execution of RDKG with any PPT adversary $\mathcal{B}$, such that the simulated execution terminates with an *identical* setup to that of the abstract DKG. Consequently, if $\mathcal{B}$ can successfully attack the coin-flipping protocol under this setup, $\mathcal{S}$ can easily break the security of the coin flipping under the abstract DKG setup.

The core of this proof is to ensure that in RDKG, the honest nodes in $\mathcal{P}$ agree on a PVSS transcript, which is an aggregation of transcripts containing at least one generated by a forever-honest node. Intuitively, this property is ensured by our RDKG via the following arguments: First, the deterministic recursive

partition ensures the existence of a forever-honest-majority path, say

$$\mathcal{D}^{(d,k_d)} \subset \mathcal{D}^{(d-1,k_{d-1})} \ldots \subset \mathcal{P}.$$

Along this path, each consortium maintains a majority of forever-honest nodes. Under the SDXH assumption, the underlying BB and PCSBB (given proper setup) are secure in the random oracle model. Then, ensured by the validity of BB, the PVSS transcripts generated by forever-honest nodes in $\mathcal{D}^{(d,k_d)}$ are received by all honest nodes in $\mathcal{D}^{(d,k_d)}$ and subsequently aggregated into $\mathbb{T}^{(d,k_d)}$. By the validity of PCSBB, $\mathbb{T}^{(d,k_d)}$ is propagated to all honest nodes in $\mathcal{D}^{(d-1,k_{d-1})}$ and aggregated into $\mathbb{T}^{(d-1,k_{d-1})}$. Repeating this argument up the recursion layers, the final transcript $T^{(0)}$ becomes an aggregation of PVSS transcripts, including those generated by forever-honest nodes.

One subtle issue is that the security of $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$ depends on the secure execution of the endemic coin-flipping protocol within the honest-majority sub-consortium $\mathcal{D}^{(r+1,k_{r+1})}$. This, in turn, requires that $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$ provides a secure DKG setup for coin flipping within $\mathcal{D}^{(r+1,k_{r+1})}$. We formalize this dependency with a layer-by-layer security analysis.

- **Defining the security of subroutines** $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$**:** For the analysis, we define the security of each subroutine $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ in Definition 7. In particular, the subroutine is secure if, when $\mathcal{D}^{(r,k_r)}$ has an honest majority: (1) All honest nodes within $\mathcal{D}^{(r,k_r)}$ agree on the same tuple $\mathbb{T}^{(r,k_r)}$, which consists of valid PVSS transcripts, and each of them is an aggregation of a set of transcriptions containing at least one generated by a forever honest node. (2) The setup provided by $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ ensures secure coin flipping within $\mathcal{D}^{(r,k_r)}$.
- **Security analysis for the base case** $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$**:** In Lemma 16, we prove that $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$ satisfies the security requirements in Definition 7. Specifically, the security of BB ensures that all honest nodes in $\mathcal{D}^{(d,k_d)}$ agree on a set of PVSS transcripts, including those sent by honest nodes. The aggregated transcript contains contributions from honest nodes, and the setup derived from $T^{(d)}$ guarantees secure coin flipping within $\mathcal{D}^{(d,k_d)}$.
- **Security analysis for the** r**-th recursion** $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$**:** In Lemma 17, we prove that if the subroutines $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$ and $\mathsf{RDKG}[\mathsf{sid}|r+1|\overline{k_{r+1}}]$ are secure, then $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ is secure. When $\mathcal{D}^{(r,k_r)}$ has an honest majority, at least one sub-consortium, say $\mathcal{D}^{(r+1,k_{r+1})}$, also has an honest majority. This enables secure coin flipping within $\mathcal{D}^{(r+1,k_{r+1})}$, ensuring the security of $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$. As a result, the aggregated PVSS transcripts $\mathbb{T}^{(r+1,k_{r+1})}$ are

propagated and aggregated into $\mathbb{T}^{(r,k_r)}$, which forms the basis for secure coin flipping within $\mathcal{D}^{(r,k_r)}$.

By combining Lemma 16 and Lemma 17, we conclude that $\mathsf{RDKG}[\mathsf{sid}]$ establishes a secure DKG setup for coin flipping within $\mathcal{P}$, completing the proof.

SECURITY JUSTIFICATIONS FOR COMPONENT COMPOSITION. We now justify the security of multiple subroutines when they are composed inside $\mathsf{RDKG}[\mathsf{sid}]$. This follows from the structure of our protocol, where different subroutines are either executed in parallel by disjoint sets of nodes (e.g., $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ and $\mathsf{RDKG}[\mathsf{sid}|r|\overline{k_r}]$) or sequentially composed (e.g., $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ and $\mathsf{PCSBB}[\mathsf{sid}|r-1|k_{r-1}]$). However, the only piece of secret information reused across subroutines is the secret signing key for the underlying MTS scheme (used inside PCSBB). An adversary might thus obtain valid signatures from honest nodes across different subroutines. Nevertheless, since every subroutine has a unique identifier and all messages in a subroutine must begin with this identifier, signatures obtained in one subroutine cannot help the adversary forge signatures in another.

Formally, in Lemma 15, we prove that if a subroutine $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ is secure in the stand-alone setting against an enhanced adversary (as defined in Remark 2), it remains secure when composed with other subroutines. This enhanced adversary can control nodes outside the sub-consortium $\mathcal{D}^{(r,k_r)}$ and query a signing oracle for messages not prefixed by $(\mathsf{sid}|r|k_r)$. Since such an adversary can simulate all other subroutines except $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$, any attack on the composition can be reduced to the stand-alone case. We apply a similar approach to justify the security of PCSBB and BB when composed with other components. The unique identifiers and message isolation ensure these subroutines retain their security properties in the composed protocol.

4.6.3 Full Security Proof

The original security proof in (Bacho et al., 2024) shows the coin-flipping protocol in Fig.B.1 is secure under the specific recursive DKG protocol presented in (Bacho et al., 2024). Since we give an improved recursive DKG (RDKG) protocol in §4.6 to replace its original DKG, it becomes necessary to analyze its security under our new DKG, despite the generation of same key structure.

Security under Abstract DKG. We proceed with the security proof via the following two steps. First, we generalize the original security proof in (Bacho et al., 2024) to show that the coin flipping protocol

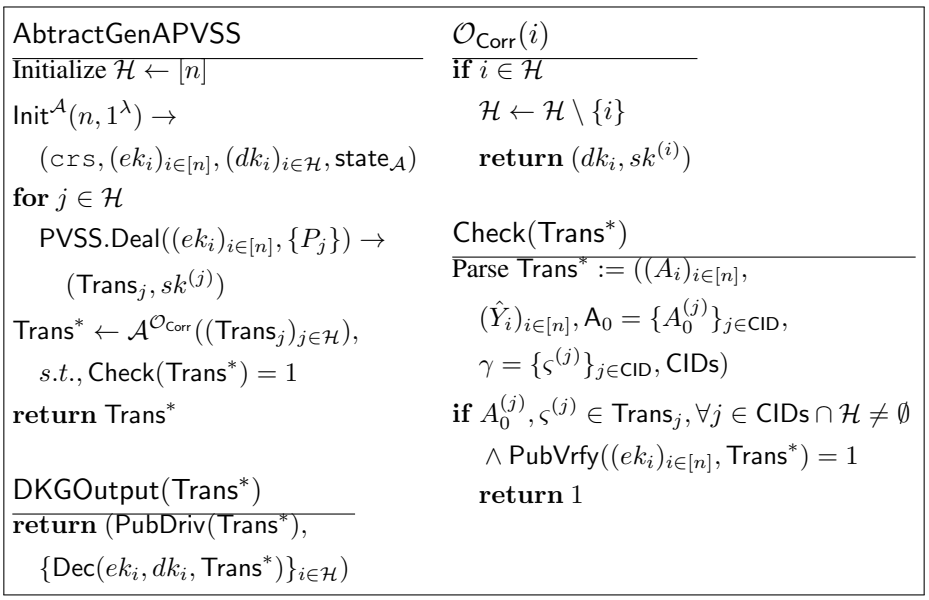


FIGURE 4.10: Abstract DKG Setup using the APVSS scheme in Appendix B.1.1. It first runs AbstractGenAPVSS with $\mathcal{A}$ to obtain Trans^* and then outputs $\mathsf{DKGOuput}(\mathsf{Trans}^*)$.

as long as the DKG protocol can be "captured" by the abstract DKG setup we defined in Fig.4.10. Next, we show our RDKG in Fig.4.9 indeed satisfies this abstraction.

Availability and agreement. We show that when the APVSS transcript Trans^* is valid, the coin flipping protocol satisfies the availability and agreement. Let $(pk, (pk_i)_{i\in[n]}, (sk_i)_{i\in\mathcal{H}}) \leftarrow \mathsf{DKGOuput}(\mathsf{Trans}^*)$, and when Trans^* is valid, there exists a t-degree polynomial f, such that $pk_i = g^(f(i)) \in \mathbb{G}_1$ for all $i \in [n]$, $sk_i = h^{f(i)} \in \mathbb{G}_2$ for all $i \in \mathcal{H}$, and $pk = g^{f(0)}$.

First, it is easy to see that during the $\mathsf{ComSetup}$, cm_i produced by an honest participant P_i can pass the verification:

$$e(pk_i, h) = e(g, h)^{f(i)}, \text{ while } e(g^{-\alpha_i}, h) \cdot e(g, h^{-\alpha}) \cdot e(g, h^{f(i)}) = e(g, h)^{f(i)}.$$

Therefore, for any honest P_i, its local set $\mathcal{G}$ contains all forever honest participants, which is at least $t + 1$.

Then, for every $\mathsf{Coin}^V[\mathsf{sid}]$, the tuple (σ_i, π_i) produced by an honest P_i can pass the verification. Therefore, every honest P_i can receive $t+1$ valid pairs from the nodes in $\mathcal{G}$. Thus, every honest P_i can produce an output in $\mathsf{Coin}^V[\mathsf{sid}]$, which satisfies the availability.

If (σ_j, π_j) passes the verification, then, $\sigma_{j,1} = H_1(\mathsf{sid})^{\alpha_j}$ and $\mathsf{cm}_{j,1} = g^{\alpha_j}$ (ensured by the Dleq proof π_j). When $P_j \in \mathcal{G}$, it holds that $\mathsf{cm}_{j,2} = h^{-\alpha_i} sk_i$. Therefore, $\sigma_{j,2} = e(H_1(\mathsf{sid}), h^{-\alpha_i} sk_i) \cdot e(H_1(\mathsf{sid})^{\alpha_j}, h) = e(H_1(\mathsf{sid}), h)^{f(j)}$. Thus, σ reconstructed from any set of $t+1$ valid (σ_j, π_j) is $e(H_1(\mathsf{sid}), h)^{f(0)}$, which ensures the agreement.

Bias-resistance and unpredictability. We establish the following lemma.

LEMMA 14. *Assume that there are a set of polynomially many instances* $\{\mathsf{Coin}^V[\mathsf{sid}_k]\}_{k\in S}$ *running concurrently under the same abstract DKG setup (c.f. Fig.4.10), where* $\mathcal{A}$ *outputs* Trans^* *in the* AbstractGenAPVSS *subroutine. Under the condition that there exists at least one forever honest* $j^* \in \mathsf{getCID}(\mathsf{Trans}^*)$*, then, despite that* $\mathcal{A}$ *might take any attacking strategies, for every* $k \in S$*, the bias-resistance and unpredictability are ensured.*

The proof largely follows the original proof in (Bacho et al., 2023c), which concludes that "... the keys output by GRand (their DKG name) are directly derived from the final APVSS transcript which is just an aggregation of several initially sampled PVSS transcripts with contribution from at least one honest party...", and the abstract DKG surely satisfies the condition.

In the following, we show how to slightly modify the original proof for proving the security under the abstract DKG setup.

First, if there exists an instance $\mathsf{Coin}^V[\mathsf{sid}_k]$ for some $k \in S$ for which the adversary $\mathcal{A}$ can violate the bias-resistance and unpredictability, then adversary $\mathcal{A}$ can generate $\sigma = e(H_1(\mathsf{sid}_k), h^{f(0)})$ before enough honest participants activating this instance. In particular, let $\mathsf{state}_{\mathcal{A}}^{(k),\mathsf{before}}$ be the state of $\mathcal{A}$ at the time that less than $t+1-|\mathcal{C}|$ honest parties have generated the corresponding $\{\sigma_j, \pi_j\}$. Note that the coin value v_{sid_k} is $H_2(\sigma)$, so any polynomial algorithm $\mathcal{D}$ given $\mathsf{state}_{\mathcal{A}}^{(k),\mathsf{before}}$ cannot distinguish it from a uniformly sampled value r, if $\mathcal{D}$ cannot compute σ. Therefore, it suffices to show that any PPT $\mathcal{A}$ cannot output (σ, k) at the time that less than $t+1-|\mathcal{C}|$ honest parties have participated the instance $\mathsf{Coin}^V[\mathsf{sid}_k]$.

Next, we go through the following hybrid games, where **Game 0** is the "real" game with an adversary aiming at producing (σ, k) without enough honest nodes activated in the instance of $\mathsf{Coin}^V[\mathsf{sid}_k]$. We prove that if the advantage of the adversary in **Game 0** is non-negligible, then its advantage must be non-negligible in the last game. After that, we show its advantage in the last game must be negligible, which evidences that the advantage in **Game 0** must be negligible as well.

Game 0: After the setup phase, $\mathcal{A}$ can determine when an honest participant P_i is activated in an instance $\mathsf{Coin}^V[\mathsf{sid}_k]$ and observe the communication transcripts. It can also choose which participant to corrupt at any time. When $\mathcal{A}$ terminates, $\mathcal{A}$ outputs (k, σ). We say $\mathcal{A}$ wins the game, if $\sigma = e(H_1(\mathsf{sid}_k), h^{f(0)})$, and we denote the probability that $\mathcal{A}$ wins the game by $\mathsf{Adv}^0_{\mathcal{A}}(\lambda)$.

Game 1: It is identical to Game 0, except that it adds an additional abort condition. It randomly samples $k^* \leftarrow\!\!\$\ S$ in the beginning and aborts when $k \neq k^*$ for (k, σ) returned by $\mathcal{A}$. Note that the choice of k^* is independent of $\mathcal{A}$'s view. Therefore, the advantage of $\mathcal{A}$ in this game, $\mathsf{Adv}^1_{\mathcal{A}}(\lambda)$, is at least $\frac{\mathsf{Adv}^0_{\mathcal{A}}(\lambda)}{|S|}$, where $|S|$ is the number of all Coin instances, which is assumed to be polynomial in λ.

Game 2: It is identical to Game 1, except that it reprograms the random oracle H_1 as follows. For input sid_{k^*}, program $H_1(\mathsf{sid}_{k^*}) := \xi_{1,1}$, where $(\xi_1, \ldots, \xi_n)$ is the co-OMDL challenge. For other inputs, randomly sample $\gamma \leftarrow\!\!\$\ \mathbb{Z}_p^*$, and return g^γ. Note that the standard techniques are applied to ensure the consistency of queries. At the point of $\mathcal{A}$'s view, this game is identical to the last game, and thus $\mathsf{Adv}^2_{\mathcal{A}}(\lambda) = \mathsf{Adv}^1_{\mathcal{A}}(\lambda)$.

Game 3: It is identical to Game 2, except that it adds an additional abort condition. It randomly sample $i^* \leftarrow\!\!\$\ [n]$ in the beginning and aborts when P_{i^*} is corrupted or $i^* \notin \mathsf{getCID}(\mathsf{Trans}^*)$. Note that we already required the existence of at least one forever honest node in $\mathsf{getCID}(\mathsf{Trans}^*)$. Therefore, the advantage of $\mathcal{A}$ in this game, $\mathsf{Adv}^3_{\mathcal{A}}(\lambda)$, is at least $\frac{\mathsf{Adv}^2_{\mathcal{A}}(\lambda)}{n}$.

Game 4: It is identical to Game 3, except that it adds an addition abort condition. Namely, when the adversary can violate the Sim-Ext security of the SoK in the APVSS scheme, it aborts.

Game 5: It is identical to Game 4, except that it adds an addition abort condition. Namely, when the adversary can violate the soundness of Dleq in the coin flipping protocol, it aborts.

Game 6: It is identical to Game 5, except that it adds an addition abort condition. Namely, when the adversary can find a collision for H_1, it aborts.

Given the security of the underlying building blocks, it is easy to see the advantage of $\mathcal{A}$ in Game 6, denoted by $\mathsf{Adv}^6_{\mathcal{A}}(\lambda)$, equals to $\frac{\mathsf{Adv}^0_{\mathcal{A}}(\lambda)}{n \cdot |S|} - \mathsf{negl}(\lambda)$. Then, if $\mathsf{Adv}^0_{\mathcal{A}}(\lambda)$ is non-negligible, $\mathsf{Adv}^6_{\mathcal{A}}(\lambda)$ is also non-negligible.

It remains to prove $\mathsf{Adv}^6_{\mathcal{A}}(\lambda)$ is negligible. The part exactly follows the original proof in (Bacho et al., 2024, Page 24 to Page 29), which shows the adversary's advantage in their game $\mathbf{G}_7$ is negligible, under the co-OMDL assumption (cf. Def.9). Indeed, all information the adversary can learn from our **Game 6**, including (1) the group descriptions, (2) the random oracle queries, (3) the PVSS transcripts generated by honest nodes, (4) the "partial signatures" (σ_i, π_i) generated by honest nodes, and the secret keys of honest nodes, are available to the adversary in their game $\mathbf{G}_7$, and vice versa. The only difference is how the final Trans^* is generated. In our **Game 6**, the adversary seems to be more powerful as it can specify Trans^*, as long as it satisfies that $\mathsf{Check}(\mathsf{Trans}^*) = 1$, and that a forever honest node $P_{i^*} \in \mathsf{getCID}(\mathsf{Trans}^*)$; in their game $\mathbf{G}_7$, the adversary can interfere the generation of Trans^* by corrupting participants in their DKG protocol. However, after a careful inspection, we can see that the original proof actually works for the seemingly more powerful adversary.

In particular, the original proof gives six PPT algorithms $A_1, \dots A_6$ for solving the co-OMDL problem, which use four simulation strategies $\{\mathsf{Sim}_1, \dots, \mathsf{Sim}_4\}$ to invoke a PPT adversary $\mathcal{A}$ against the coin flipping protocol. It shows that, if an algebraic adversary $\mathcal{A}$ can produce the valid (k, σ) such that $\sigma = e(H_1(\mathsf{sid}_k), h^{f(0)})$ with a non-negligible probability, then at least one of the six algorithms can solve the co-OMDL problem with a non-negligible probability.

We can build PPT algorithms $B_1, \dots, B_6$ accordingly for solving the co-OMDL problem by using an adversary $\mathcal{A}$ against the coin flipping protocol under the abstract DKG.

- B_1, B_2 and B_3: Note that A_1, A_2, and A_3 only use the robustness of their DKG protocol, i.e., all honest nodes will output consistent secret key shares at the end of the DKG. In particular, A_1 (resp. A_2 or A_3) applies the simulation strategy Sim_1 (resp. Sim_2 or Sim_3), where the simulation algorithm acts on the behalf of all honest nodes during the DKG protocol by honestly following the protocol specification.

  B_1, B_2 and B_3 are identical to A_1, A_2, and A_3, respectively, except that during the DKG part, B_1, B_2 and B_3 honestly generate the PVSS transcripts on the behalf of all honest nodes and let $\mathcal{A}$ specify the final Trans^*.

We note that only the simulation strategy Sim_4, used by A_4, A_5, A_6, utilizes "properties" of their DKG protocol.

- Sim_4 (Bacho et al., 2024, Page 26): On input the co-OMDL challenge $(\xi_1, \dots, \xi_n)$, it queries DL_g with $(\xi_{t+2}, \dots, \xi_n)$ and gets $(z_{t+2}, \dots, z_n)$. Then, during the DKG protocol, it honestly

generates the PVSS transcripts on the behalf of all honest nodes except P_{i^*} (that contributes to the final aggregated PVSS transcript and remains honest). For P_{i^*}, its PVSS transcript is generated as follows: however, it generates the degree-t polynomial $f_{i^*} = d_0 + d_1 X + \ldots + d_t X^t \in \mathbb{Z}_p[X]$ such that $g^{d_j} = \xi_{j+1}$ for all $j \in [t]$ (i.e., the $t+1$ coefficients of the polynomial are given by the discrete logarithm values of $\xi_1, \ldots, \xi_{t+1}$). From this, it can generate the commitments and encrypted shares of party P^*'s PVSS transcript by Lagrange interpolation in the exponent and knowledge of the secret keys sk_j of all parties. Since Sim_4 generates the public keys pk_i of honest parties faithfully and the adversary A is algebraic, it outputs the (updated) public keys of corrupt parties as a linear combination of known values, which enables Sim_4 to compute the respective secret keys from this linear combination along with the algebraic representation. In addition, the signature of knowledge w.r.t. $A_0^{(i^*)} = g^{f_{i^*}(0)}$ is generated with the simulation algorithm of the underlying SoK.

- $A_4, A_5,$ and A_6: The three algorithms use Sim_4 to simulate their game $\mathbf{G}_7$ for invoking the adversary $\mathcal{A}$. To convert the forgery (k, σ) outputted by $\mathcal{A}$ into a solution of the co-OMDL challenge, the requirement on the DKG part is that the public key pk derived from Trans^* can be expressed as $\xi_{1,1} \cdot g^{\alpha}$ for some $\alpha \in \mathbb{Z}_p$.
- $B_4, B_5,$ and B_6: They are almost identical to $A_4, A_5,$ and A_6, respectively, except that the during the DKG part they generate all PVSS transcripts of honest nodes like Sim_4 does but let $\mathcal{A}$ to decide the final aggregated transcript Trans^*.

  Note that in our **Game 6**, there exists the designated party P_{i^*} which is forever honest and included in $\mathsf{getCID}(\mathsf{Trans}^*)$. Therefore, the final aggregated PVSS Trans^* is in the form of $((A_i)_{i\in[n]}, (\hat{Y}_i)_{i\in[n]}, \mathsf{A}_0 = \{A_0^{(j)}\}_{j\in\mathsf{CID}\setminus\{i^*\}} \cup \{\xi_{1,1}\}, \gamma = \{\varsigma^{(j)}\}_{j\in\mathsf{CID}}, \mathsf{CIDs})$. According to the algorithm $\mathsf{PubDriv}$, we have that $pk = \xi_{1,1} \cdot \prod_{j\in\mathsf{CIDs}\setminus\{i^*\}} A_0^{(j)}$. Note $A_0^{(j)}$ for $j \neq i^*$ comes with a valid signature of knowledge $\varsigma^{(j)}$. Ensured by the simulation extraction property of the underlying SoK, we can obtain $\alpha^{(j)}$ such that $A^{(j)} = g^{\alpha^{(j)}}$. So, $pk = \xi_{1,1} \cdot g^{\sum_j \alpha^{(j)}}$.

By following the exactly same analysis in (Bacho et al., 2024), it can be established that one of the six algorithms $B_1, \ldots, B_6$ can solve the co-OMDL problem with a non-negligible probability, if the algebraic adversary $\mathcal{A}$ can produce a valid tuple (k, σ) at the time that no enough honest nodes have activated the instance $\mathsf{Coin}^V[\mathsf{sid}_k]$.

Security under Our DKG in §4.6. Note that our group-element DKG protocol in Fig.4.9 has a recursive construction. At the every recursion, every consortium has an intra-consortium DKG setup and runs multiple instances of the one-round coin-flipping protocol within the consortium.

DEFINITION 7 (Subroutine security). *Let $\mathcal{D}^{(r,k_r)}$ be a consortium at the r-th recursion, and let $\mathsf{RDKG}[\mathsf{sid}|r|k_r]\langle\mathcal{D}^{(r,k_r)}\rangle$ be an instance running within $\mathcal{D}^{(r,k_r)}$. We say that $\mathsf{RDKG}[\mathsf{sid}|r|k_r]\langle\mathcal{D}^{(r,k_r)}\rangle$ is secure if, when $\mathcal{D}^{(r,k_r)}$ has an honest majority (i.e., at least $\lfloor\frac{|\mathcal{D}^{(r,k_r)}|}{2}\rfloor+1$ nodes in $\mathcal{D}^{(r,k_r)}$ are forever honest), the following properties hold except with negligible probability.*

- ***Agreement:*** *If honest nodes P_i and P_j in the consortium output $\mathbb{T}_i^{(r,k_r)}$ and $\mathbb{T}_j^{(r,k_r)}$ respectively, then $\mathbb{T}_i^{(r,k_r)}=\mathbb{T}_j^{(r,k_r)}$.*
- ***Validity:*** *Within a predetermined Δ_r rounds, each honest node P_i can output*

$$(sk_i^{(r)}, pk^{(r)}, (pk_j^{(r)})_{j\in\mathcal{D}^{(r,k_r)}}, \mathbb{T}^{(r,k_r)}),$$

  where the tuple $\mathbb{T}^{(r,k_r)}$ can be parsed as $\{\mathbf{T}_{k_r}^{(r-1)},\ldots,\mathbf{T}_{k_r}^{(0)}\}$. For all $z\in[0,r-1]$, it holds that $\mathsf{PVSS.Vrfy}((ek_j)_{j\in\mathcal{D}^{(z,k_z)}},\mathbf{T}_{k_r}^{(z)})=1$. Moreover, there exists a forever honest node P_{i^*} that contributes to all $\mathbf{T}_{k_r}^{(z)}$. Specifically, if we parse

$$\mathbf{T}_{k_r}^{(z)} := ((A_j)_{j\in\mathcal{D}^{(z,k_z)}}, (\hat{Y}_j)_{j\in\mathcal{D}^{(z,k_z)}}, \mathsf{A}_0=\{A_0^{(k)}\}_{k\in\mathsf{CIDs}}, \gamma=\{\varsigma^{(k)}\}_{k\in\mathsf{CIDs}}, \mathsf{CIDs}),$$

  then $P_{i^*}\in\mathsf{getCID}(\mathbf{T}_i^{(z)})$, and $A_0^{(i^*)}\in\mathsf{A}_0$ was generated by P_{i^*}.
- ***Secure coin flipping:*** *If each node $P_i\in\mathcal{D}^{(r,k_r)}$ uses $(pk^{(r)},(pk_j^{(r)})_{j\in\mathcal{D}^{(r,k_r)}},sk_i^{(r)})$ as the setup key for the coin-flipping protocol, and there are polynomially many instances of the coin-flipping protocol running concurrently, then every instance satisfies the properties of availability, agreement, bias-resistance, and unpredictability.*

REMARK 2 (Enhanced stand-alone security). *For ease of analysis, we consider the security of $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ in the stand-alone setting, which involves the following adversary $\mathcal{A}$: The adversary $\mathcal{A}$ can observe all public parameters and public keys of $\mathcal{P}$ and the communication transcripts generated during the execution of $\mathsf{RDKG}[\mathsf{sid}|r|k_r]\langle\mathcal{D}^{(r,k_r)}\rangle$. Furthermore, it can activate any node $P_i\in\mathcal{D}^{(r,k_r)}$ in instances of the coin-flipping protocol under the setup provided by $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$. $\mathcal{A}$ can adaptively corrupt up to $\lfloor\frac{|\mathcal{D}^{(r,k_r)}|}{2}\rfloor$ nodes within $\mathcal{D}^{(r,k_r)}$. We consider $\mathcal{A}$ to have succeeded if it terminates within Δ_r rounds and one of the following events occurs:*

- *The agreement or validity property of $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ is violated.*

- *$\mathcal{A}$ outputs $(\mathsf{sid}', r_{\mathsf{sid}'})$ such that fewer than $\lfloor \frac{|\mathcal{D}^{(r,k_r)}|}{2} \rfloor + 1 - |\mathcal{C}^{\mathcal{A}}|$ honest nodes in $\mathcal{D}^{(r,k_r)}$ have been activated in the coin-flipping instance with session identifier sid', yet $r_{\mathsf{sid}'}$ becomes the common coin output of that instance, i.e.,*

$$r_{\mathsf{sid}'} = e(H_1(\mathsf{sid}'), sk^{(r)}),$$

where $sk^{(r)}$ is the Lagrange interpolation of all secret key shares $sk_i^{(r)}$.

The adversary $\mathcal{A}$ is considered enhanced in the sense that it can corrupt all nodes in $\mathcal{P} \setminus \mathcal{D}^{(r,k_r)}$, since these nodes are not participants in the execution of $\mathsf{RDKG}[\mathsf{sid}|r|k_r]\langle\mathcal{D}^{(r,k_r)}\rangle$. Additionally, the decryption keys $\{dk_i^{(r')}\}_{r'\in[0,r-1]}$ of any honest node $P_i \in \mathcal{D}^{(r,k_r)} \setminus \mathcal{C}_{\mathsf{init}}^{\mathcal{A}}$ are available to $\mathcal{A}$, where $\mathcal{C}_{\mathsf{init}}^{\mathcal{A}}$ denotes the set of nodes initially corrupted by $\mathcal{A}$. Moreover, $\mathcal{A}$ can query a partial signing oracle $\mathcal{O}_{\mathsf{PartSign}}(\cdot)$ associated with the underlying MTS scheme. However, $\mathcal{A}$ is restricted from querying messages that begin with the session identifier "$\mathsf{sid}|r|k_r$".

REMARK 3 (Composition Security). *To demonstrate the security of the final protocol $\mathsf{RDKG}[\mathsf{sid}]$, we need to establish that the security of each subroutine holds when composed with other subroutines in the protocol execution. Specifically, we consider the following adversary model.*

Let $\mathcal{A}$ be a PPT adversary that can adaptively corrupt participants in $\mathcal{P}$ during the execution of $\mathsf{RDKG}[\mathsf{sid}]$ and observe all communication transcripts generated by honest parties. $\mathcal{A}$ can utilize the information learned from any subroutine to influence the behavior of other subroutines.

The winning condition for $\mathcal{A}$ in the composition setting is the same as that for an enhanced stand-alone adversary described in Remark 2. This means that $\mathcal{A}$ succeeds if it violates the agreement, validity, or secure coin-flipping properties of $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ under the conditions outlined in Definition 7.

Fortunately, for our protocol, we can show when $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ is secure in the stand-alone setting against an enhanced adversary, it remains secure in the $\mathsf{RDKG}[\mathsf{sid}]$.

LEMMA 15. *Assume the network $\mathcal{P}$ runs $\mathsf{RDKG}[\mathsf{sid}]$, and let $\mathsf{RDKG}[\mathsf{sid}|r|k_r]\langle\mathcal{D}^{(r,k_r)}\rangle$ be a subroutine of it. For every $r \in [0, d]$ and $k_r \in [2^r]$, if $\mathsf{RDKG}[\mathsf{sid}|r|k_r]\langle\mathcal{D}^{(r,k_r)}\rangle$ is secure in the stand-alone setting against any enhanced adversary, then it remains secure in the composition.*

PROOF. Let $\mathcal{A}$ be a PPT adversary which violates one of the security properties of $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ in the composition. Then, we can build an enhanced PPT adversary $\mathcal{B}$ which violates the security of $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ in the stand-alone setting. In particular, $\mathcal{B}$ works as follows:

Setup phase: $\mathcal{B}$ takes the public parameter pp as input and provides pp to $\mathcal{A}$. Upon $\mathcal{A}$ provides the set of initially corrupted nodes $\mathcal{C}^{\mathcal{A}}_{\mathsf{init}}$, $\mathcal{B}$ corrupts $\mathcal{C}^{\mathcal{B}}_{\mathsf{init}} := \mathcal{C}^{\mathcal{A}}_{\mathsf{init}} \cup (\mathcal{P} \setminus \mathcal{D}^{(r,k_r)})$. Then, $\mathcal{B}$ receives all public keys and the decryption keys $\{dk_i^{(r')}\}_{r' \in [0,r-1]}$ of all $P_i \in \mathcal{D}^{(r,k_r)} \setminus \mathcal{C}^{\mathcal{A}}_{\mathsf{init}}$, and it generates the key pairs for all $\mathcal{C}^{\mathcal{B}}_{\mathsf{init}} \setminus \mathcal{C}^{\mathcal{A}}_{\mathsf{init}}$. $\mathcal{B}$ provides the public keys of all $P_i \in \mathcal{P} \setminus \mathcal{C}^{\mathcal{A}}_{\mathsf{init}}$ to $\mathcal{A}$. Then, $\mathcal{A}$ returns the public keys for nodes in $\mathcal{C}^{\mathcal{A}}_{\mathsf{init}}$. Finally, $\mathcal{B}$ returns the public keys of all $P_i \in \mathcal{C}^{\mathcal{B}}_{\mathsf{init}}$.

Execution: $\mathcal{B}$ simulates an execution of $\mathsf{RDKG}[\mathsf{sid}]$ (denoted by $\mathsf{Sim}[\mathsf{sid}]$) for $\mathcal{A}$, while engaging in an execution of $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ (denoted by $\mathsf{Real}[\mathsf{sid}|r|k_r]$) with the aim of breaking its security. Let $\mathcal{C}^{\mathcal{B}}$ be the set of nodes so-far corrupted by $\mathcal{B}$ in $\mathsf{Real}[\mathsf{sid}|r|k_r]$, and let $\mathcal{C}^{\mathcal{A}}$ be the set of nodes so-far corrupted by $\mathcal{A}$ in $\mathsf{Sim}[\mathsf{sid}]$. $\mathcal{B}$ works as follows during the execution phase.

First, for any $P_j \in \mathcal{C}^{\mathcal{A}} \cap \mathcal{D}^{(r,k_r)}$, $\mathcal{B}$ forwards every message sent by $\mathcal{A}$ on the behalf of P_j in $\mathsf{Sim}[\mathsf{sid}]$ to $\mathsf{Real}[\mathsf{sid}|r|k_r]$ (also on the behalf of P_j). For any P_j, *s.t.*, $j \in \mathcal{D}^{(r,k_r)} \setminus \mathcal{C}^{\mathcal{A}}$ (the honest nodes), $\mathcal{B}$ forwards every message sent by P_j in $\mathsf{Real}[\mathsf{sid}|r|k_r]$ to $\mathsf{Sim}[\mathsf{sid}]$ (also on the behalf of P_j).

While the above strategy states how $\mathcal{B}$ simulates messages sent by P_j for $j \in \mathcal{P} \setminus \mathcal{C}^{\mathcal{A}}$ in the subroutine $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$, however, in $\mathsf{Sim}[\mathsf{sid}]$, $\mathcal{B}$ also needs to simulate messages in subroutines other than $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$. We can classify the messages into two categories: (1) those sent by $P_j \in \mathcal{C}^{\mathcal{B}} \setminus \mathcal{C}^{\mathcal{A}}$ (the nodes corrupted by $\mathcal{B}$ but not by $\mathcal{A}$), and (2) those sent by $P_j \in \mathcal{D}^{(r,k_r)} \setminus \mathcal{C}^{\mathcal{A}}$ (the honest nodes in $\mathcal{D}^{(r,k_r)}$).

For (1), $\mathcal{B}$ can simulate these messages by honestly following the protocol specification, as $\mathcal{B}$ knows the secret states of all $P_j \in \mathcal{C}^{\mathcal{B}} \setminus \mathcal{C}^{\mathcal{A}}$. For (2), according to the protocol description, $P_j \in P_j \in \mathcal{D}^{(r,k_r)} \setminus \mathcal{C}^{\mathcal{A}}$ needs to perform actions in $\mathsf{RDKG}[\mathsf{sid}|r-1|k_{r-1}], \ldots, \mathsf{RDKG}[\mathsf{sid}|1|k_1], \mathsf{RDKG}[\mathsf{sid}]$, if $P_j \in \mathcal{D}^{(r-1,k_{r-1})}, \ldots, \mathcal{D}^{(1,k_1)}$. In the following, we illustrate how $\mathcal{B}$ can simulate the actions of P_j in $\mathsf{RDKG}[\mathsf{sid}|r-1|k_{r-1}]$, while the simulation for other subroutines can be done via a similar approach.

- In $\mathsf{RDKG}[\mathsf{sid}|r-1|k_{r-1}]$, following the subroutine $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$, P_j needs to perform the commitment setup of the coin flipping instances $\mathsf{Coin.ComSetup}[\mathsf{sid}|r|k_r]$ within $\mathcal{D}^{(r,k_r)}$. Then, in the subroutine $\mathsf{PCSBB}[\mathsf{sid}|r-1|k_{r-1}]$, P_j needs to participant in a few coin flipping instances to generate common coins within $\mathcal{D}^{(r,k_r)}$. $\mathcal{B}$ can simulate these actions in $\mathsf{Sim}[\mathsf{sid}]$

by activating P_j in the corresponding coin flipping instances in $\mathsf{Real}[\mathsf{sid}|k|r]$ and forwarding the messages sent by P_j to $\mathsf{Sim}[\mathsf{sid}]$ (on the behalf of P_j).

- In subroutine $\mathsf{PCSBB}[\mathsf{sid}|r-1|k_{r-1}]$, the messages sent by $P_i \in \mathcal{D}^{(r,k_r)} \setminus \mathcal{C}^{\mathcal{A}}$ (except the messages for coin flipping) can be perfectly simulated with the help of a partial signing oracle $\mathcal{O}_{\mathsf{PartSign}}(\cdot)$ w.r.t. the MTS scheme. Therefore, since $\mathcal{B}$ can query the partial signing oracle with messages that do not start with "$\mathsf{sid}|r|k$", $\mathcal{B}$ can simulate all messages within $\mathsf{PCSBB}[\mathsf{sid}|r-1|k_{r-1}]$.
- Following the subroutine $\mathsf{PCSBB}[\mathsf{sid}|r-1|k_{r-1}]$, P_j needs to process the output of the PCSBB instance, which requires P_j to decrypt the PVSS transcript $T^{(r-1)}$. Since $\mathcal{B}$ is given the decryption keys $\{dk_i^{(r')}\}_{r'\in[0,r-1]}$, $\mathcal{B}$ can perform these actions honestly.

Handle corruption: When $\mathcal{A}$ corrupts P_i in $\mathsf{Sim}[\mathsf{sid}]$, if $i \in \mathcal{D}^{(r,k_r)}$, $\mathcal{B}$ corrupts P_i in $\mathsf{Real}[\mathsf{sid}|r|k_r]$; Otherwise, $\mathcal{B}$ returns the internal states of P_i to $\mathcal{A}$, as this node is controlled by $\mathcal{B}$ in $\mathsf{Sim}[\mathsf{sid}]$.

Output: $\mathcal{B}$ terminates when $\mathcal{A}$ terminates, and it outputs what $\mathcal{A}$ outputs.

It is easy to check that at the point of $\mathcal{A}$'s view, the execution of $\mathsf{Sim}[\mathsf{sid}]$ simulated by $\mathcal{B}$ is identical to a real execution of $\mathsf{RDKG}[\mathsf{sid}]$. Therefore, if $\mathcal{A}$ wins with a non-negligible probability, $\mathcal{B}$ can win with a non-negligible probability. □

LEMMA 16 (Base Security). *Assume the network $\mathcal{P}$ runs $\mathsf{RDKG}[\mathsf{sid}]$, and let $\{\mathcal{D}^{(d,k)}\}_{k\in[0,2^d-1]}$ be all consortiums at d-th recursion (the base case). For each $k \in [0, 2^d-1]$, the instance $\mathsf{RDKG}[\mathsf{sid}|r|k_d]\langle\mathcal{D}^{(d,k)}\rangle$ is secure. I.e., when $\mathcal{D}^{(d,k)}$ has an honest majority, the instance satisfies agreement and validity, and enables secure coin flipping, as per Def.7.*

PROOF. We analyze the security of $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$ against *enhanced* stand-alone adversaries (c.f. Remark.2), which implies its security when it is composed with other subroutines in $\mathsf{RDKG}[\mathsf{sid}]$, following Lemma 15.

First, we examine the security of $\mathsf{BB}[\mathsf{sid}|d|k_d|i]$ (for $P_i \in \mathcal{D}^{(d,k_d)}$) when composed with other subroutines in $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$ under an enhanced adversary. The enhanced adversary can corrupt all parties outside $\mathcal{D}^{(d,k_d)}$, but this does not affect the security of $\mathsf{BB}[\mathsf{sid}|d|k_d|i]$, as those parties are irrelevant to the broadcast instance. Additionally, the adversary can access the signing oracle for the underlying MTS scheme, but only for messages that do not begin with the identifier "$\mathsf{sid}|d|k_d$". Since these signatures cannot be used to forge valid messages for the broadcast instance, the adversary gains no advantage

from this capability. Within $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$, other subroutines are separate BB instances with different identifiers, which do not interfere with each other. As a result, the security of $\mathsf{BB}[\mathsf{sid}|d|k_d|i]$ is not compromised by its composition with these subroutines. Therefore, $\mathsf{BB}[\mathsf{sid}|d|k_d|i]$ remains secure even when executed within the full protocol and under an enhanced adversary.

Given the security of the underlying BB in the composition, we analyze each property in the following.

Agreement: Ensured by the agreement of BB, all honest nodes output the same tuple $\{(T_j^{(d)}, \cdots T_j^{(0)})\}_{j\in\mathcal{D}^{(d,k_d)}}$. Since $\mathsf{PVSS.Agg}$ is a deterministic algorithm, all participants return the same $\mathbb{T}_i^{(d,k_d)}$

Validity: It is easy to verify that all honest nodes in $\mathcal{D}^{(d,k)}$ can terminate in $\mathsf{RDKG}[\mathsf{sid}|r|k_d]$ within Δ_{BB} rounds, where Δ_{BB} is the number of rounds of $\mathsf{BB}[\mathsf{sid}|r|k_d|i]$ (for $P_i \in \mathcal{D}^{(d,k)}$). Ensured by the validity of BB, each honest P_i can receive a tuple $(T_j^{(d)}, \cdots T_j^{(0)})$ generated by an honest P_j. So, the aggregated transcripts by P_i satisfy the required form.

Secure coin flipping: Finally, we show the output of $\mathsf{RDKG}[\mathsf{sid}|d|d_r]$ provides a secure setup for the one-round coin flipping protocol (cf. Fig.B.1) within $\mathcal{D}^{(d,k_d)}$. Namely, let $\mathbf{T}^{(d)}$ be the aggregated PVSS transcript an honest node P_i obtained at the end of $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$, and let $(pk^{(d)}, (pk_j^{(d)})_{j\in\mathcal{D}^{(d,k_d)}}, sk_i^{(d)})$ be the public key shares and the secret key share that P_i obtained by decrypting $\mathbf{T}^{(d)}$. Then, the coin flipping protocol is secure under the set up of $(pk^{(d)}, (pk_j^{(d)})_{j\in\mathcal{D}^{(d,d_r)}}, sk_i^{(d)})$.

We prove this fact by showing that $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$ can be "captured" by our abstract DKG in Fig.4.10. Specifically, we construct a PPT algorithm $\mathcal{S}$, which is an adversary in the abstract DKG and also *perfectly* simulates an execution of $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$, such that the transcript $\mathbf{T}^{(d)}$ in the simulated execution is identical to the transcript Trans^* returned by $\mathcal{S}$ in the abstract DKG, which means the simulated execution and the abstract DKG give an identical setup for the coin flipping protocol. Therefore, if there is a PPT adversary $\mathcal{B}$ which can violate the security of the coin flipping under the setup of the simulated execution, it can be trivially "translated" into an attack against the coin flipping protocol under the abstract DKG setup.

For clarity, let $\mathcal{B}$ be a PPT adversary against $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$, and we can build $\mathcal{S}$ in Fig.4.11. It is easy to verify that, at the point of $\mathcal{B}$'s view, the simulated execution is identical to a real execution of $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$. Therefore, as we analyzed above, this simulated execution should produce $\mathbf{T}^{(d,k_d)}$, which is a valid, and there is a forever honest P_{i^*} has contributed to $\mathbf{T}^{(d,k_d)}$. Thus, as $\mathcal{S}$ returns Trans^*,

Initial Phase: Once receiving $\texttt{crs}$ from the environment of abstract DKG, $\mathcal{S}$ provides $\texttt{crs}$ to $\mathcal{B}$. Then $\mathcal{B}$ should declare the initially corrupted set $\mathcal{C}^{\mathcal{B}}_{\mathsf{init}}$, such that $\mathcal{C}_{\mathsf{init}} = \mathcal{C}^{\mathcal{B}}_{\mathsf{init}} \cap \mathcal{D}^{(r,k_r)} \leq \lfloor \frac{|\mathcal{D}^{(r,k_r)}|-1}{2} \rfloor$. We assume w.l.o.g. that all nodes outside $\mathcal{D}^{(r,k_r)}$ are included in $\mathcal{C}^{\mathcal{B}}_{\mathsf{init}}$, as $\mathcal{B}$ is enhanced. $\mathcal{S}$ then passes $\mathcal{C}_{\mathsf{init}}$ to the environment. Next, $\mathcal{S}$ receives the public encryption keys $(ek_i)_{i\in\mathcal{D}^{(r,k_r)}\setminus\mathcal{C}_{\mathsf{init}}}$. Then, $\mathcal{S}$ sets $ek_i^{(r)} = ek_i$ for all $i \in \mathcal{D}^{(r,k_r)} \setminus \mathcal{C}_{\mathsf{init}}$, and the generates the key pairs $(ek_i^{(z)}, dk_i^{(z)})$ for all $z \in [0, d-1], z \neq r$. $\mathcal{S}$ also generates the key pairs w.r.t. the MTS scheme and the digital signature scheme for all nodes in $\mathcal{D}^{(r,k_r)} \setminus \mathcal{C}_{\mathsf{init}}$. Then, $\mathcal{S}$ provides all public keys of nodes in $\mathcal{D}^{(r,k_r)} \setminus \mathcal{C}_{\mathsf{init}}$ to $\mathcal{B}$, waits $\mathcal{B}$ to provide the public keys of $\mathcal{C}_{\mathsf{init}}$, and then provides $(ek_i^{(r)})_{i\in\mathcal{C}_{\mathsf{init}}}$ to the environment.

Input: $\mathcal{S}$ receives $(\mathsf{Trans}_j)_{j\in\mathcal{D}^{(r,k_r)}\setminus\mathcal{C}_{\mathsf{init}}}$ from the environment of abstract DKG.

Simulating $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$**:** $\mathcal{S}$ simulates the execution as follows.

- During the base case (d-th recursion), $\mathcal{S}$ on the behalf of all honest nodes generates the PVSS transcripts. In particular, for $j \in \mathcal{D}^{(r,k_r)} \setminus \mathcal{C}_{\mathsf{init}}$, $\mathcal{S}$ honestly generates the PVSS transcripts $\mathbf{T}_j^{(z)}$, for all $z \in [0, d-1]$ but $z \neq r$; $\mathcal{S}$ sets $\mathbf{T}_j^{(r)} := \mathsf{Trans}_j$ which it received from the environment of the abstract DKG.

  For all the subsequent operations of $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$, $\mathcal{S}$ acts on the behalf of all honest nodes by honestly executing the protocol. We remark that $\mathcal{S}$ can do this, because all secret keys of an honest node P_i, except $(ek_i^{(r)}, dk_i^{(r)})$, are generated by $\mathcal{S}$, and $dk_i^{(r)}$ is not needed to generate any messages in $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$.
- When $\mathcal{B}$ requests to corrupt a node P_i, $\mathcal{S}$ queries the oracle $\mathcal{O}_{\mathsf{Corr}}$ with i and obtains $(dk_i^{(r)}, sk^(r, i))$, which are the decryption key w.r.t. $ek_i^{(r)}$, and the secret of the PVSS transcript $\mathbf{T}_i^{(r)}$. Note that the secret keys of all other public keys of P_i are generated by $\mathcal{S}$, and the secrets of all $\mathbf{T}_i^{(z)}$ for $z \neq r$ are known to $\mathcal{S}$. Therefore, $\mathcal{S}$ can return all secret internal states of P_i, including $(dk_i^{(r)}, sk^(r, i))$ and other secret information, to $\mathcal{B}$.

Output: When an honest node in the simulated execution obtains the aggregated PVSS transcript $\mathbf{T}^{(r,k_r)}$, $\mathcal{S}$ returns $\mathsf{Trans}^* = \mathbf{T}^{(r,k_r)}$ to the environment of abstract DKG.

FIGURE 4.11: The description of $\mathcal{S}$ which is an adversary in abstract DKG and simulates $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ for $r \in [0, d]$.

it follows that $\mathsf{Check}(\mathsf{Trans}^*) = 1$ as required by the abstract DKG in Fig.4.10. Therefore, the abstract DKG and the simulated $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$ provide identical setup for the one-round coin flipping protocol.

Subsequently, if there is a PPT adversary $\mathcal{B}'$ which on the input of $\mathcal{B}$'s internal state can violate the security of the coin flipping protocol under the setup of the simulated $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$, there is a PPT adversary $\mathcal{S}'$ which on the input of $\mathcal{S}$'s internal state can violate the coin flipping protocol under the setup of the abstract DKG. However, as we analyzed in Lemma 14. Under the setup of abstract DKG, conditioned on Trans^* has been contributed by a forever honest P_{i^*}, the coin flipping protocol is secure. Therefore, the coin flipping protocol under the setup of $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$ is secure. □

Next, we analyze the security of a recursion $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ for $r < d$, under the condition that its subroutines $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$ are secure.

LEMMA 17 (Recursion Security). *Assume the network $\mathcal{P}$ runs $\mathsf{RDKG}[\mathsf{sid}]$, and let $\{\mathcal{D}^{(r,k_r)}\}_{k\in[0,2^r-1]}$ be all consortiums at r-th recursion, for some $r \in [0, d-1]$. Assuming that its subroutines $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$ and $\mathsf{RDKG}[\mathsf{sid}|r+1|\overline{k_{r+1}}]$ are secure in the composition, $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ is secure for $k \in [0, 2^r-1]$. I.e., when $\mathcal{D}^{(r,k_r)}$ has an honest majority, the instance satisfies termination, agreement, and validity, and enables secure coin flipping, as per Definition 7.*

PROOF. $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ has the following subroutines: $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$, $\mathsf{RDKG}[\mathsf{sid}|r+1|\overline{k_{r+1}}]$, and $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$. We first examine the security of $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$ when it is composed of the other two subroutines. Recall the security results of PCSBB in Theorem 2, which states an instance $\mathsf{PCSBB}[\mathsf{sid}']$ satisfies its security properties when (1) one consortium has an honest majority, (2) the coin flipping instances within the honest-majority consortium is secure, and (3) the adversary cannot forge an MTS signature for messages starting with "sid'". When $\mathcal{D}^{(r,k_r)}$ has an honest majority, one of its two sub-consortiums $\mathcal{D}^{(r+1,k')}$ and $\mathcal{D}^{(r+1,k'')}$ must have an honest majority. Given the security of $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$ and $\mathsf{RDKG}[\mathsf{sid}|r+1|\overline{k_{r+1}}]$, the coin-flipping instances within the honest-majority sub-consortium are secure. In addition, the information that an adversary can learn from $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r_{r+1}}]$ and $\mathsf{RDKG}[\mathsf{sid}|r+1|\overline{k_{r+1}}]$ will not help the adversary to forge an MTS signature for messages starting with "$\mathsf{sid}|r|k$".

Given the security of all subroutines when they are composed with each other, we analyze each property in the following. Without loss of generality, we assume that the sub-consortium $\mathcal{D}^{(r+1,k_{r+1})}$ has an honest majority.

Agreement: The agreement follows the agreement property of the underlying PCSBB. In particular, $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$ returns the same vector $\mathcal{V}^{(r)}$ to all nodes in $\mathcal{D}^{(r,k_r)}$. Since all honest nodes follow a deterministic rule to parse $\mathcal{V}^{(v)}$ to PVSS transcripts, to verify these transcripts, and to aggregate valid transcripts, all honest nodes have the same aggregated transcripts $\mathbb{T}^{(r,k_r)}$.

Validity: Note that $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$ can terminate within Δ_{r+1} rounds, such that all honest participants have a valid output $\mathbb{T}^{(r+1,k_{r+1})} = \{T^{(r)}_{k_{r+1}}, \cdots, T^{(0)}_{k_{r+1}}\}$. Then, it takes one round to finish the commitment setup for the endemic coin within $\mathcal{D}^{(r+1,k_{r+1})}$. Therefore, the protocol can be configured to start $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$ at the beginning of round $\Delta_{r+1}+2$, where the consortium $\mathcal{D}^{(r+1,k_{r+1})}$

(resp. $\mathcal{D}^{(r+1,\overline{k_{r+1}})}$) provides $\mathbb{T}^{(r+1,k_{r+1})}$ (resp. $\mathbb{T}^{(r+1,\overline{k_{r+1}})}$) as the input. Ensured by Theorem 2, $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$ can terminate within $\Delta_{\mathsf{PCSBB},r}$ rounds, where $\Delta_{\mathsf{PCSBB},r}$ is a predetermined constant. Hence, honest parties can receive the output of $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$ by the end of round $\Delta_{r+1} + 1 + \Delta_{\mathsf{PCSBB},r}$. Ensured by the validity of PCSBB, all honest nodes can receive the input of the honest-majority consortium, i.e., $\mathbb{T}^{(r+1,k_{r+1})}$, and optionally the input from another consortium. Then, the honest nodes can verify the PVSS transcripts and aggregate them accordingly. Ensured by the validity of $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$, all PVSS transcripts in $\mathbb{T}^{(r+1,k_{r+1})}$ are valid and have been contributed by a forever honest node $P_{i^*} \in \mathcal{D}^{(r+1,k_{r+1})}$. Then, according to the aggregation algorithm $\mathsf{PVSS.Agg}$, the aggregated PVSS transcripts are valid and have contributions by P_{i^*}.

Secure coin flipping: This can be proved through the exactly the same argument in the proof of Lemma 16. In particular, the simulator algorithm $\mathcal{S}$ in Fig.4.11 applies to simulating $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ for any $r \in [0, d]$. □

4.7 Near-Optimal DKG for DLog Cryptosystems

Though the DKG scheme in the previous section achieves nearly optimal communication and round complexities, it however only generates secret keys as Elliptic Curve group elements, which is incompatible with the majority of DLog-based cryptosystems where the secret key is in a scalar field $\mathbb{F}$. To close this gap, in this section, we present a DKG protocol for field-element secrets, where our group-element DKG only serves as a setup for a coin-flipping protocol. Moreover, we prove the DKG scheme achieves oracle-aided algebraic simulatability (Bacho and Loss, 2022) against adaptive adversaries and full secrecy (Gennaro et al., 2007) against static adversaries, making our protocol broadly applicable and as useful as the classical DKG protocols like the one by Gennaro et al. (Gennaro et al., 2007).

4.7.1 Subset Byzantine Broadcast

Following the high-level idea in Section 4.3.2, the communication cost could be saved if only $O(\kappa)$ randomly selected nodes contribute secrets to the final secret key, which suffices for secrecy as there exists one honest node among the $O(\kappa)$ selected nodes with an overwhelming probability. However, if naively using $O(\kappa)$ BB instances for delivering the selected nodes' PVSS transcripts, a rushing adaptive adversary can corrupt the senders and violate robustness. Alternatively, using $O(n)$ BB instances to let all nodes deliver PVSS transcripts (before selecting $O(\kappa)$-size committee) is robust against rushing adversaries but causes cubic communication. To address this issue, we introduce *Subset Byzantine Broadcast* (SBC), which begins with a propose phase, in which all nodes disperse their inputs to the network, followed by another delivery phase, where a selected subset of dispersed values will be reconstructed and delivered to the entire network. Intuitively, this simultaneously achieves adaptive security and low communication complexity, because, after the communication-efficient propose phase, it is "too late" for any rushing adversary to prevent the dispersed inputs from being reconstructed. Formally, SBC can be defined as follows.

DEFINITION 8 (Subset Byzantine broadcast). *An* (n, t, ℓ)-SBC *protocol involves* n *participants* $\mathcal{P} = \{P_1, \ldots, P_n\}$ *where at most* t *of them can be corrupted. It syntactically consists of two subsequent phases:*

- $\mathsf{SBC.propose}$: Every P_i has an ℓ-bit input msg_i to this phase, and would (implicitly) return certain protocol transcripts while P_i terminates.

- SBC.deliver: After the propose phase is terminated, each P_i is given the same $\mathbb{S}$ (i.e., a subset of $[n]$) as input for the deliver phase, where $\mathbb{S}$ can be provided by some higher level protocol (e.g., determined by a common coin). At the end of the deliver phase, each P_i will output the corresponding subset of input values $\{\mathsf{msg}'_i\}_{i\in\mathbb{S}}$.

An (n, t, ℓ)-secure SBC protocol shall satisfy the following properties.

- **Agreement:** Every two honest parties output the same set $\{\mathsf{msg}'_i\}_{i\in\mathbb{S}}$ in the deliver phase.
- **Validity:** Let $\mathbb{H}$ be the set of "so-far-honest" parties at the end of the propose phase. Then, with an overwhelming probability, for every $j \in \mathbb{H} \cap \mathbb{S}$, we have $\mathsf{msg}'_j = \mathsf{msg}_j$. I.e., the network can deliver the original input value of P_j, if P_j has not been corrupted before the end of the propose phase.
- **Termination:** Both propose and deliver phases can terminate.

SBC Construction. We first provide a high-level overview of this protocol. In the propose phase, every node P_i encodes its input msg_i into n fragments $(c_{i,1}, \ldots, c_{i,n})$ using an erasure coding scheme (Blahut, 1983), and applies an accumulator scheme to all fragments, so that the fragments correctly committed to the accumulator can later be identified; P_i then sends $c_{i,j}$ and its accumulator witness to each P_j. During the delivery phase, given a subset $\mathbb{S} \subset [n]$, every P_i multicasts the fragments and witnesses received from $\{P_j\}_{j\in\mathbb{S}}$ to the network, so that every node would reconstruct some values associated with $\{P_j\}_{j\in\mathbb{S}}$. Then all participants use $|\mathbb{S}|$ instances of BA (Abraham et al., 2019) to reach a consensus on the reconstructed values. The underlying BA protocol (Abraham et al., 2019) requires common coins, so does our SBC protocol.

Next, we provide a detailed construction in Fig.4.12 and formalize analysis of Subset Byzantine Broadcastin Lemma 18.

Given a cryptographic accumulator $\mathsf{Acc} = \{\mathsf{Acc.Gen}, \mathsf{Acc.Eval}, \mathsf{Acc.Wit}, \mathsf{Acc.Vrfy}\}$, a $(t + 1, n)$ erasure code scheme $\mathsf{EC} = \{\mathsf{EC.Enc}, \mathsf{EC.Dec}\}$, and an (n, t, ℓ)-Byzantine agreement protocol BA, one can construct an efficient (n, t, ℓ)-SBC protocol, as illustrated by Figure 4.12. Particularly, we assume the following setups as granted before the protocol execution: (1) The accumulator scheme's public parameter $ak \leftarrow \mathsf{Acc.Gen}(1^\lambda, n)$, and (2) all setup needed for the underlying BA, including a common (weak) coin within $\mathcal{P}$.

SBC.propose[sid]. Every $P_i \in \mathcal{P}$, on input msg_i, performs the following steps:

Step 1: // Encode the message and send fragments to different parties

$\mathsf{EC.Enc}(\mathsf{msg}_i) \rightarrow (c_{i,1}, \ldots, c_{i,n})$; $\mathsf{Acc.Eval}(ak, \{(j, c_{i,j})\}_{j\in[n]}) \rightarrow u_i$

for $j \in [n]$, $\mathsf{Acc.Wit}(ak, u_i, (j, c_{i,j})) \rightarrow w_{i,j}$ and **send** $(\mathsf{sid}, \mathsf{propose}, u_i, c_{i,j}, w_{i,j})$ to P_j

Step 2: **wait** for $(\mathsf{sid}, \mathsf{propose}, u_j, c_{j,i}, w_{j,i})$ from all $P_j \in \mathcal{P}$ until the end of Round 1.

then **terminate**

SBC.deliver[sid]. Every $P_i \in \mathcal{P}$, on input $\mathbb{S}$, performs the following steps:

Step 1: // Echo the fragments from the selected parties

multicast $(\mathsf{sid}, \mathsf{deliver}, \{u_z, (i, c_{z,i}, w_{z,i})\}_{z\in\mathbb{S}})$

Step 2: // Reconstruct the input messages of the selected parties

receive $(\mathsf{sid}, \mathsf{deliver}, \{u_z^{(j)}, (j, c_{z,j}, w_{z,j})\}_{z\in\mathbb{S}})$ from all $P_j \in \mathcal{P}$

for $z \in \mathbb{S}$, **if** $\exists \mathbb{J}_z \subset [n]$ and u_z, s.t. $|\mathbb{J}_z| = t + 1 \wedge$

$\forall j \in \mathbb{J}_z, u_z^{(j)} = u_z \wedge \mathsf{Acc.Vrfy}(ak, u_z, (j, c_{z,j}), w_{z,j}) = 1$ **then**

$\mathsf{msg}_z^{(i)} = \mathsf{EC.Dec}(\{(j, c_{z,j})\}_{j\in\mathbb{J}_z})$

else $\mathsf{msg}_z^{(i)} = \perp$

Step 3: // reach a consensus on all delivered messages

for $z \in \mathbb{S}$, **invoke** $\mathsf{BA}[\mathsf{sid}||z]$ with the input $\mathsf{msg}_z^{(i)}$, and obtain the output msg_z

output $\{\mathsf{msg}_z\}_{z\in\mathbb{S}}$

FIGURE 4.12: The Subset Byzantine Broadcast protocol.

LEMMA 18 (SBC). *With a setup for an adaptively secure unique threshold signature scheme, the SBC protocol in Fig.4.12 is an adaptively secure* (n, t, ℓ)-SBC *protocol (cf. Definition 8) for any* $t < \frac{n}{2}$. *In addition, its expected communication cost is* $O(|\mathbb{S}|n\ell + \lambda|\mathbb{S}|n^2 \log n)$, *and its expected round complexity is* $O(\log |\mathbb{S}|)$. *Moreover, there is a predetermined bound* $T_{\mathsf{SBC}} = O(\kappa)$, *such that SBC can terminate within* T_{SBC} *rounds, with an overwhelming probability* $1 - \mathsf{negl}(\kappa)$.

The Analysis. We provide a proof of Lemma 18.

PROOF. Regarding security, *agreement* follows that of BA, and *termination* follows the synchrony message delivery bound Δ, and the termination of BA. For *validity*, let's consider a party $P_i \in \mathbb{H} \cap \mathbb{S}$ and has an input msg_i. Since party P_i has not been corrupted, even by an adaptive adversary, at the end of the propose phase, it must have correctly generated the n coded fragments from its input msg_i, calculated corresponding witnesses that prove the validity of fragments, and sent them to every $P_j \in \mathcal{P}$. We emphasize that this information is successfully delivered to all honest parties before the deliver phase begins, regardless of whether an adaptive adversary corrupts P_i after the end of the propose phase.

Consequently, all parties can collect fragments and witness from all parties in $\mathbb{H}$ at the beginning of the deliver phase. Then, they pick the fragments and witnesses received from $\mathbb{S}$, and multicast them. For any $t < \frac{n}{2}$, there must be at least $t+1$ valid fragments of msg_i available to every honest party by the end of deliver phase step 1. This ensures any honest party P_j can reconstruct the same $\mathsf{msg}_i^{(j)} = \mathsf{msg}_i$, and use it to invoke $\mathsf{BA}[\mathsf{sid}||i]$. By the validity of BA, participants should output the original input, hence $\mathsf{BA}[\mathsf{sid}||i]$ outputs $\mathsf{msg}_i^{(k)} = \mathsf{msg}_i$ for all honest party P_k.

Next, we analyze the communication complexity of SBC. In the $\mathsf{SBC.propose}$ phase, each party P_i sends out $(\mathsf{sid}, \mathsf{propose}, u_i, c_{i,j}, w_{i,j})$ whose size is $O(\lambda + \mathsf{w} + \ell/n)$ to every $P_j \in \mathcal{P}$. The communication cost of this procedure is $O(n^2(\lambda + \mathsf{w}) + n\ell)$. In the $\mathsf{SBC.deliver}$ phase step 1, each party P_i multicast $(\mathsf{sid}, \mathsf{deliver}, \{u_z, (i, c_{z,i}, w_{z,i})\}_{z\in\mathbb{S}})$ whose size is $O(|S| \cdot (\lambda + \mathsf{w} + \ell/n))$. In step 3, each party invokes $|S|$ BA instances where each BA has an input $\mathsf{msg}_z^{(i)}$ of size $O(\ell)$. Adding them together, the total communication cost is $O(n^2|S|(\lambda + \mathsf{w}) + n|S|\ell + |S|\mathsf{BA}_n(\ell))$. Consider using the efficient BA instantiation and a Merkle tree instantiation for witness, the expected total communication cost reduces to $O(|\mathbb{S}|n\ell + \lambda|\mathbb{S}|n^2 \log n)$, i.e. the communication complexity stated in Lemma 18.

Finally, we investigate the round complexity in expectation or with an overwhelming probability. Since the underlying BA protocol (Abraham et al., 2019) is a randomized protocol that terminates when a randomly elected leader is honest, the number of rounds to terminate is a geometric random variable X_i with probability $p = \frac{f+1}{2f+1}$. Therefore a single BA instance is expected to terminate in $E(X_i) = 1/p = O(1)$ iteration (round). However, in SBC, $|S|$ numbers of BA instances are invoked concurrently. These $|S|$ parallel BA instances are expected to terminate in $E(\max_{i\in[|S|]}\{X_i\})$, the expected value of the maximum of $|S|$ IID geometric random variables with mean $1/p$, which approximates to $O(\log |S|)$ iterations (rounds). On the other hand, by Lemma 2, with an overwhelming probability $1 - 2^{-\kappa}$, there is an integer $T' = O(\kappa)$, such that among any $N' \leq 2^\kappa$ BA instances, every BA instance can terminate within T' rounds. In $N = \mathsf{poly}(\kappa)$ SBC instances, the number of BA instances is still polynomial in κ, so every BA instance can terminate within T' rounds with an overwhelming probability. Note that besides the $\mathsf{poly}(\kappa)$ parallel BA instances, the other parts of our SBC protocol are deterministic and only incur constant rounds. Therefore, there is an integer T_{SBC}, such that every SBC instance among N SBC instances can terminate within T_{SBC} rounds with an overwhelming probability. This proves the round complexity stated in Lemma 18. □

4.7.2 The Field-element DKG Protocol

Building Blocks. Our field-element DKG uses the following components.

SCRAPE'S LOW DEGREE TEST. Following (Shrestha et al., 2023; Das et al., 2023c), we use a variant of Pedersen's polynomial commitment. Particularly, to commit a t-degree polynomial $f \in \mathbb{F}[X]$, we randomly sample another t-degree polynomial $f' \in \mathbb{F}[X]$, and compute the commitment as $(g^{f(1)}h^{f'(1)}, \ldots, g^{f(n)}h^{f'(n)})$, where $n > t+1$ is an integer, and g and h are uniformly sampled two distinct generators of $\mathbb{G}$. Such a polynomial commitment admits the efficient low-degree test due to Scrape (Cascudo and David, 2017). For ease of use, we formulate the next two algorithms for verifying if a vector of n group elements $(C_1, \ldots, C_n)$ commits a t-degree polynomial.

- $\mathsf{PC.VkGen}(n, t, \mathbb{F})$. Uniformly sample $q(X) \in \mathbb{F}[X]$ whose degree is at most $(n-t-2)$, and return $\mathsf{pvk} = (pvk_j)_{j\in[n]}$ where $pvk_j = \frac{q(j)}{\prod_{i=1, i\neq j}^{n}(j-i)}$.
- $\mathsf{PC.Vrfy}(\mathsf{pvk}, (C_1, \ldots, C_n))$. Return 1, if $\prod_{j=1}^{n}(C_j)^{\mathsf{pvk}_j} = \mathbf{1}_{\mathbb{G}}$; Otherwise, 0.

The effectiveness of the test is summarized by the following lemma from (Shrestha et al., 2023; Das et al., 2023c).

LEMMA 19 (borrowed from (Shrestha et al., 2023; Das et al., 2023c)). *Let g and h be two distinct generators for a group $\mathbb{G}$ with the scalar field $\mathbb{F}$. Let $pvk \leftarrow \mathsf{PC.VkGen}(n, t, \mathbb{F})$. Then, for any $(C_1, \ldots, C_n) \in \mathbb{G}^n$, if $\mathsf{PC.Vrfy}(pvk, C_1, \ldots, C_n) = 1$, then*

$$\Pr[\exists t\text{-degree } f, f' \in \mathbb{F}[X], \text{ s.t. } C_i = g^{f(i)}h^{f'(i)}, \forall i \in [n]] \geq 1 - \frac{1}{|\mathbb{F}|}.$$

NIZK. We use a non-interactive zero-knowledge proof scheme $\mathsf{NIZK} = \{\mathsf{NIZK.Prove}, \mathsf{NIZK.Vrfy}\}$ for the following relationship:

$$\{(\mathsf{cm}, pk, g, h) \in \mathbb{G}^4; (sk, r) \in \mathbb{F}^2 : \mathsf{cm} = g^{sk}h^r \wedge pk = g^{sk}\} \tag{4.1}$$

Here (cm, pk, g, h) corresponds to the public statement, and (sk, r) is the private knowledge. The NIZK scheme shall satisfy *completeness*, *simulation knowledge soundness*, and *zero knowledge*. In Appendix B.3.1, we give a simple instantiation of it in the random oracle model.

CONSENSUS AND LEADER ELECTION COMPONENTS. We use SBC (described in Fig.4.12) as the consensus component. To provide common coins used in SBC, our group-element DKG protocol RDKG

(cf. Fig. 4.9 in Section 4.6) is used to establish the setup of one-round coin-flipping protocol from (Bacho et al., 2024) (described in Fig.B.1). The leader election subroutine $\mathsf{Election}^{[n]^{2\kappa}}$, which randomly samples 2κ indexes in $[n]$, is realized from the same coin-flipping protocol (e.g., randomly permute $[n]$ using common coins and then pick up the first 2κ indexes). In addition, we assume a standard digital signature scheme $\mathsf{DS} = \{\mathsf{DS.KeyGen}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy}\}$ with existential unforgeability under chosen message attacks.

Protocol description. Following the high-level idea mentioned in Section 4.3.2 and given the above ingredients, we formally depict our field-element DKG protocol in Fig.4.13. Informally, it executes as follows.

Initially at **step 0**, all parties execute our group-element DKG protocol to establish an adaptively secure unique threshold signature, serving as the setup phase of SBC and election protocols.

From **step 1** to **step 3**, each node interacts with all other nodes and tries to form a publicly verifiable transcript of secret sharing. In particular, a node P_i commits to a secret polynomial f_i using Pedersen commitment and sends a piece of commitment $\mathsf{cm}_{i,j}$ along with its opening to each P_j (step 1). When receiving a valid opening for a commitment $\mathsf{cm}_{j,i}$ from P_j, P_i signs the commitment (step 2). Finally, a valid transcript consists of all commitments, signatures for the commitments, and valid openings without signatures. Such a transcript ensures that all nodes can receive consistent shares, since either the receiver has confirmed via its signature, or the share is contained in the transcript. Then, every node disperses the transcript to the whole network using SBC (step 3).

At **step 4**, 2κ indices are randomly sampled from $[n]$. Since all nodes have erased their secrets, an adaptive adversary cannot learn these secrets by corrupting the selected nodes. After the election, all nodes retrieve the transcripts dispersed by the selected nodes within the delivery phase of SBC, at **step 5**.

From **step 6** to **step 7**, each node verifies the received transcripts, and obtains its secret key share by aggregating the secret shares from the valid transcripts; then computes its public key share, and multicasts it. Finally, at **step 8**, all public key shares were already received or can be derived by interpolating the received valid public key shares.

Each $P_i \in \mathcal{P}$ performs the following steps:

Step 0: Inovke RDKG[sid|rdkg], and Coin.ComSetup[sid|coin] for coin flipping

Step 1: Sample t-degree polynomials: f_i, f'_i over $\mathbb{F}[X]$

Commit shares: $\mathsf{cm}_{i,j} = g^{f_i(j)} h^{f'_i(j)}$

send $(\mathsf{sid}, \textsc{Share}, \mathsf{cm}_{i,j}, f_i(j), f'_i(j))$ to P_j, for each $j \in [n]$

Step 2: // upon receving shares, return the signed commitments

receive $(\mathsf{sid}, \textsc{Share}, \mathsf{cm}_{j,i}, f_j(i), f'_j(i))$ from every $P_j \in \mathcal{P}$

for every $j \in [n]$, **if** $\mathsf{cm}_{j,i} = g^{f_j(i)} \cdot h^{f'_j(i)}$ **then**

$\mathsf{DS.Sign}(\mathsf{DS}.vk_i, \mathsf{DS}.sk_i, \mathsf{cm}_{j,i}) \to \sigma_{j,i}$; **send** $(sid, \textsc{Sign}, \sigma_{j,i})$ to P_j

Step 3: // prepare and propose the PVSS transcript

receive $(sid, \textsc{Sign}, \sigma_{i,j})$ from every $P_j \in \mathcal{P}$; Initialize a vector Trans_i

for every $j \in [n]$, **if** $\mathsf{DS.Vrfy}(\mathsf{DS}.vk_j, \mathsf{cm}_{i,j}, \sigma_{i,j}) = 1$ **then** $\mathsf{cert}_{i,j} = \sigma_{i,j}$

else $\mathsf{cert}_{i,j} = (f_i(j), f'_i(j))$

Invoke SBC.propose[sid|sbc] with the input $\mathsf{Trans}_i = ((\mathsf{cm}_{i,j}, \mathsf{cert}_{i,j}))_{j\in[n]}$

wait until it terminates. **erase** f_i, f'_i except $f_i(i), f'_i(i)$

Step 4: Invoke $\mathsf{Election}^{[n]^{2\kappa}}$[sid|ele] $\to \mathbb{S}$// elect a random subset of 2κ indexes

Step 5: // deliver the PVSS transcripts of the chosen subset through SBC.deliver

Inovke SBC.deliver[sid|sbc] with the subset $\mathbb{S}$, and obtain the output $\{\mathsf{Trans}_z\}_{z\in\mathbb{S}}$

Step 6: Initialize $\mathsf{Qual} = \emptyset$; $\mathsf{pvk} \leftarrow \mathsf{PC.VkGen}(n, t, p)$// verify delivered transcripts

for every $z \in \mathbb{S}$, parse $\mathsf{Trans}_z = ((\mathsf{cm}_{z,j}, \mathtt{cert}_{z,j}))_{j\in[n]}$

if $\mathsf{PC.Vrfy}(\mathsf{pvk}, (\mathsf{cm}_{z,j})_{j\in[n]}) = 1$ and **for** $j \in [n]$, $\mathtt{cert}_{z,j}$ is either a valid opening of $\mathsf{cm}_{z,j}$ or a valid signature for $\mathsf{cm}_{z,j}$, **then** $\mathsf{Qual} = \mathsf{Qual} \cup \{z\}$

Step 7: $sk_i \leftarrow \sum_{z\in\mathsf{Qual}} f_z(i), pk_i = g^{sk_i}, r_i \leftarrow \sum_{z\in\mathsf{Qual}} f'_z(i), \mathsf{cm}_i = \prod_{z\in\mathsf{Qual}} \mathsf{cm}_{z,i}$

$\pi_i \leftarrow \mathsf{NIZK.Prove}((\mathsf{cm}_i, pk_i), (sk_i, r_i))$// prove pk_i is honestly generated

multicast (pk_i, π_i) and wait for $(pk'_j, \pi_j)_{j\in[n]}$ until the next round starts

Step 8: Initialize $\mathsf{Valid} = \emptyset$// extract the public key

for $j \in [n], \mathsf{cm}_j = \prod_{z\in\mathsf{Qual}} \mathsf{cm}_{z,j}$, **if** $\mathsf{NIZK.Vrfy}((\mathsf{cm}_j, pk'_j), \pi_j) = 1$,

then $\mathsf{Valid} = \mathsf{Valid} \cup \{j\}$

Let $pk_j = pk'_j$ for all $j \in \mathsf{Valid}$. Let $\mathsf{Valid}' \subset \mathsf{Valid}$ include the first $t+1$ indexes in Valid.

// Let $\{\lambda_j(x)\}_{j\in\mathsf{Valid}'}$ be Lagrange basis over Valid'

Interpolate $\{pk_j\}_{j\in\mathsf{Valid}'}$: $pk = \prod_{j\in\mathsf{Valid}'} pk_j^{\lambda_j(0)}$, and

$pk_\tau = \prod_{j\in\mathsf{Valid}'} pk_j^{\lambda_j(\tau)}, \forall \tau \notin \mathsf{Valid}$; **output** $(pk, pk_1, \ldots, pk_n, sk_i)$

FIGURE 4.13: Field-element DKG with Full Secrecy. Boxed actions are for adaptive security.

4.7.3 The Analysis

Efficiency analysis. We analyze the communication cost step by step. According to Section 4.6, RDKG at step 0 incurs $O(\lambda\kappa n^2 \log n)$ bits communication and $O(\kappa \log n)$ rounds. For guaranteed termination, at step 3 and step 5, the SBC subroutine incurs $O(\lambda\kappa n^2 \log n)$ bits communication and $O(\kappa)$ rounds. For all other steps, they only incur $O(\lambda n^2)$ communication cost and $O(1)$ rounds. Therefore, the overall communication cost of our field-element DKG in Fig.4.13 is $O(\lambda\kappa n^2 \log n)$, and its round complexity is $O(\kappa \log n)$.

Security analysis. Regarding security, we establish the following result with proofs.

THEOREM 4. *The DKG protocol in Fig.4.13 satisfies robustness and oracle-aided simulatability against strongly adaptive adversaries who can corrupt up to $t < n/2$ nodes. In addition, it achieves full secrecy against static adversaries.*

Sketch Proof. First, with an overwhelming probability there exists an index $i^* \in \mathbb{S}$ s.t. P_i remains honest before step 4. Then, Trans_{i^*} must be valid, which means Qual is non-empty. On the other hand, ensured by the unforgeabililty of the underlying signature, and the binding property of Pedersen commitment, for every $z \in \mathbb{S}$, the honest nodes have valid shares from P_z. Therefore, all honest nodes can have consistent secret key shares and public key shares at step 7. Ensured by the completeness of NIZK, all public key shares from honest nodes will be computed with valid proofs and delivered to every node by the end of step 7. On the other hand, ensured by the knowledge soundness of NIZK and the binding property of Pedersen commitment, every valid public key share was honestly constructed. It follows that our DKG satisfies robustness.

The proof of full secrecy is similar to the proof in (Shrestha et al., 2023). In particular, the simulator will randomly select one honest node P_{i^*} and generate its PVSS transcript without knowing the committed polynomials, which can be done since the signatures of other honest nodes can be generated by the simulator. P_{i^*} will be elected at step 4, by programming the random oracle inside the leader election subroutine. Then, after the set Qual is decided, the simulator recovers the public key shares of all malicious nodes and then interpolates the public key shares of honest nodes based on the given public key, which ensures the generated public key is equal to the given public key. Regarding the oracle-aided algebraic simulatability, it largely follows the proof for GJKR(Gennaro et al., 2007) provided in (Bacho and Loss, 2022).

Detailed Proof. To facilitate our proof, we first restate the following fact about the low-degree testing that was previously discussed in (Shrestha et al., 2023; Das et al., 2023c).

LEMMA 20. *Let $(C_1, \ldots, C_n)$ be n group elements in $\mathbb{G}$, for which there there exist t-degree polynomials f and $f' \in \mathbb{F}[X]$ s.t. $C_i = g^{f(i)}h^{f'(i)}$ for $i \in [n]$. Then, for any subset $\mathbb{I} \subset [n]$ with $|\mathbb{I}| = t+1$ and $(a_i, b_i)_{i\in\mathbb{I}}$ s.t. $C_i = g^{a_i}h^{b_i}$ for $i \in \mathbb{I}$, interpolate two t-degree polynomials a and b such that $a(i) = a_i$ and $b(i) = b_i$. It follows that $C_i = g^{a(i)}h^{b(i)}$ for all $i \in [n]$.*

PROOF. $t+1$ evaluations uniquely identify a degree-t polynomial. Hence we have $f = a$, $f' = b$ and $g^{a(i)}h^{b(i)} = g^{f(i)}h^{f'(i)} = C_i$ for all $i \in [n]$. □

We prove the robustness in Lemma 21, the full secrecy in Lemma 23, and the oracle-aided algebraic simulatability in Lemma 22.

LEMMA 21. *The DKG protocol in Fig.4.13 satisfies robustness.*

PROOF. We first argue that all participants P_i's will output some public key and public key shares at the end of the protocol execution. By the end of **Step 7**, for every $j \in [n]$, every honest P_i either (i) receives the same valid (pk_j, π_j) from P_j, or (ii) receives invalid (pk_j, π_j) or nothing from P_j. Notably, the former case happens for every honest P_j, since the properties of SBC, $\mathsf{Election}$ and low-degree test ensure: by the end of **Step 6**, all honest nodes hold the common set of valid secret sharing transcripts $\{\mathsf{Trans}_z\}_{z\in\mathsf{Qual}}$ for non-empty $\mathsf{Qual} \in \mathbb{S}$, so all of them can multicast valid (pk_j, π_j) messages consistent to $\{\mathsf{Trans}_z\}_{z\in\mathsf{Qual}}$ during the **Step 7** for the completeness of NIZK generating π_j. Following that, when entering **Step 8**, all honest nodes have already received at least $n - t$ valid (pk_j, π_j) messages from distinct P_j, making them always interpolate some public key and public key shares.

Then, we argue that the output public key and public key shares are consistent and agreed, *i.e.*, there exists a t-degree polynomial $f \in \mathbb{Z}_p[X]$, s.t., all honest nodes output $pk = g^{f(0)}$ and $pk_i = g^{f(i)}$ for $i \in [n]$. Following **Step 1-3**, we know that honest nodes can always generate valid transcripts. Let $\mathbb{H}'$ be the set of all so-far-honest nodes before **Step 4**. Since there are κ nodes being randomly sampled in **Step 4**, the probability of $\mathbb{S} \cap \mathbb{H}' = \emptyset$ is smaller than $2^{-\kappa}$, which is negligible. By the validity of SBC, for every $z \in \mathbb{S} \cap \mathbb{SH}_3$, the transcript Trans_z delivered by SBC is exactly the transcript provided by P_z in **Step 3**. Thus, $\{\mathsf{Trans}_z\}_{z\in\mathbb{S}}$ contains honestly generated transcripts (with an overwhelming probability), which means Qual is not empty.

According to **Step 7**, an honest P_i calculates sk_i as $\sum_{z\in\mathsf{Qual}} f_z(i)$. Since every $\mathsf{cm}_{z,j}, z \in \mathsf{Qual}$ has a valid certificate $\mathsf{cert}_{z,j}$ (according to **Step 6**) which is either a valid signature under P_j's verification key or a valid opening. According to **Step 2**, an honest node will sign a commitment only when it has received the correct opening. Following the unforgeability of the underlying digital signature scheme, P_i has the correct openings for all $\{\mathsf{cm}_{z,i}\}_{z\in\mathsf{Qual}}$. Therefore, all honest P_i can calculate sk_i, pk_i, r_i, and π_i, as per **Step 7**. Then, for the completeness of NIZK for proving public key share, $(pk'_j, \pi_j)_{j\in[n]}$ received at the end of **Step 7** contain the public key shares of all honest nodes. Since honestly generated public key shares come with a valid proof, Valid contains at least $t+1$ indexes. According to **Step 8**, the other public key shares and the public key are obtained by interpolating $\{pk_j\}_{j\in\mathsf{Valid}'}$, it follows that pk, $\{pk_j\}_{j\notin\mathsf{Valid}}$, and $\{pk_j\}_{j\in\mathsf{Valid}'}$ are consistent.

It remains to show that all honest nodes terminate with the same public key and public key shares.

In our protocol, as the corrupted nodes may not send the same public key shares to all honest nodes, different honest nodes may use different subsets of valid public key shares as Valid' to interpolate the public key and other public key shares. In the following, we show that for any set of valid public key shares, $\{pk_j\}_{j\in\mathsf{Valid}}$, there exists a t-degree polynomial f, such that $pk_j = g^{f(j)}$. If such a polynomial f exists, then all honest nodes can obtain the same public key and public key shares, as they all receive the same $t+1$ public key shares from the forever honest nodes, which already determine the polynomial f.

It remains to show that all $\{pk_j\}_{j\in\mathsf{Valid}}$ (including those not in Valid') are consistent. Following the knowledge soundness of NIZK, each P_j for $j \in \mathsf{Valid}$ knows an opening (sk_j, r_j) for $\mathsf{cm}_j = \prod_{z\in\mathsf{Qual}} \mathsf{cm}_{z,j}$, and $pk_j = g^{sk_j}$. On the other hand, for every $\mathsf{Trans}_z = ((\mathsf{cm}_{z,j}, \mathsf{cert}_{z,j}))_{j\in[n]}, z \in \mathsf{Qual}$, as it satisfies $\mathsf{PC.Vrfy}(\mathsf{pvk}, \mathsf{cm}_{z,1}, \ldots, \mathsf{cm}_{z,n}) = 1$ (according to **Step 6**), it follows that, with an overwhelming probability, there exist t-degree polynomials $a_z, b_z \in \mathbb{Z}_p[X]$, s.t., $\mathsf{cm}_{z,j} = g^{a_z(j)}h^{b_z(j)}$ for all $j \in [n]$, which is ensured by Lemma 19. Furthermore, it is easy to argue that there exist t-degree polynomials $a, b \in \mathbb{Z}_p[X]$, s.t. $\mathsf{cm}_i = \prod_{z\in\mathsf{Qual}} \mathsf{cm}_{z,i} = g^{a(i)}h^{b(i)}$, for every $i \in [n]$. Then, following Lemma 20, based on $(sk_j, r_j)_{j\in\mathsf{Valid}'}$, we can obtain the opening $(sk'_i = \sum_{j\mathsf{Valid}'} \lambda_j(i)sk_j, r'_i = \sum_{j\mathsf{Valid}'} \lambda_j(i)r_j)$ for any other cm_i, $i \in [n] \setminus \mathsf{Valid}'$.

Ensured by the binding property of the underlying Pedersen commitment, we know $sk_i = sk'_i$ for all $i \in \mathsf{Valid}\backslash\mathsf{Valid}'$. In other words, all $\{(j, sk_j)\}_{j\in\mathsf{Valid}}$ are the evaluation of the same t-degree polynomial. Thus, we completed this proof. □

LEMMA 22. *The DKG protocol in Fig.4.13 satisfies the oracle-aided algebraic simulatability against adaptive adversaries.*

PROOF. We proceed this proof by constructing a PPT simulator $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG,ada}}$ which simulates an execution of the DKG involving the adversary $\mathcal{A}$. For clarity, we write down the pseudo-code of $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG,ada}}$ in Fig.4.14. In particular, the simulator is given $t+1$ group elements $D_0, \ldots, D_t \in \mathbb{G}$ and can access to the discrete logarithm oracle $\mathsf{DL}_g(\cdot)$, which on input a group element $A \in \mathbb{G}$ returns a such that $A = g^a$, for at most t times. Aligning with the definition, the simulator is algebraic about the queries to $\mathsf{DL}_g(\cdot)$ and the final pk.

We show that at the view of $\mathcal{A}$, the execution simulated by $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG,ada}}$ is indistinguishable with a real execution, through the following hybrid simualted executions.

Hybrid 0: The simulator Sim_0 acts on behalf of all so-far-honest participants by honestly following the protocol specification. When the adversary $\mathcal{A}$ corrupts a party, the simulator returns the current internal state of it to $\mathcal{A}$.

Hybrid 1: The simulator Sim_1 is almost identical to Sim_0, except that:

- At **step 7**, it generates the proof π_i for each $i \in \mathbb{H}$ by using the simulated prover algorithm $\mathsf{NIZK.SimProve}$.

Hybrid 2: The simulator Sim_2 is almost identical to Sim_1, except that:

- Before **step 4**, it uses secret key shares rsk_i (w.r.t. RDKG) of honest nodes to create the unique threshold signature $\mathsf{UTS}.\sigma$ for sid. Check whether $(\mathsf{sid}, \mathsf{UTS}.\sigma)$ has been queried by $\mathcal{A}$ to the random oracle. If it has been queried, **abort**. Otherwise, it uniformly samples $z_1, \ldots, z_\kappa \leftarrow\$ [n]$, and records $\mathsf{RO}(\mathsf{sid}, \mathsf{UTS}.\sigma) := (z_1, \ldots, z_\kappa)$.

Hybrid 3: The simulator Sim_3 is almost identical to Sim_2, except that:

- At **step 1**, it samples $i^* \leftarrow\$ \mathbb{H}$. At **step 4**, it programs the random oracle to ensure that i^* is included in the election outcome $\mathbb{S}$, as $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG,ada}}$ does in **step 4**.

$\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG},\mathsf{ada}}(D_0, D_1, D_2, \ldots, D_t)$

Prepare: Let $\mathbb{H} \subset [n]$ be the set of all so-far-honest nodes.

Let $\mathbb{C} \subset [n]$be the set of all corrupted nodes.

Step 0: honestly execute this step on the behalf of all $\{P_i\}_{i\in\mathbb{H}}$

If $\mathcal{A}$ corrupts P_j, return all internal states of P_j,

and update $\mathbb{C}$ and $\mathbb{H}$ accordingly

Step 1: sample $i^* \leftarrow\$ \mathbb{H}$, and honestly execute this step on the behalf of all

$\{P_i\}_{i\in\mathbb{H}\setminus\{i^*\}}$; For P_{i^*}, do the following:

for $j \in [n]$, compute $A_j = \prod_{\tau\in[0,t]} D_\tau^{j^\tau}$; sample $b_j \leftarrow\$ \mathbb{Z}_p$; compute

$B_j = h^{b_j}$, and $C_j = A_j B_j$

for $j \in \mathbb{C}$, **query**$\mathsf{DL}_g(\cdot)$ with $A_j$along with its description

$(0, 0, (j^\tau)_{\tau\in[0,t]})$, and obtain a_j, s.t. $A_j = g^{a_j}$

send $(\mathsf{sid}, \textsc{Share}, C_j, a_j, b_j)$ to P_j, for each $j \in \mathbb{C}$

Handling corruption after step 0:

If $\mathcal{A}$ corrupts P_{i^*} before step 4, **abort and rewind**

Otherwise, if $\mathcal{A}$ corrupts P_j for $j \in \mathbb{H}$, update $\mathbb{H}$ and $\mathbb{C}$ accordingly.

query $\mathsf{DL}_g(\cdot)$ with A_j along with its description$(0, (j^\tau)_{\tau\in[0,t]})$ over

$(g, D_0, \ldots, D_t)$, and obtain a_j, s.t. $A_j = g^{a_j}$

return (a_j, b_j) along with other internal states of P_j to $\mathcal{A}$

Step 2: honestly execute this step on the behalf of all $\{P_i\}_{i\in\mathbb{H}}$

Step 3: honestly execute this step on the behalf of all $\{P_i\}_{i\in\mathbb{H}\setminus\{i^*\}}$;

For P_{i^*}, follow the protocol except that:

for every $j \in \mathbb{H}$, let $\mathsf{cert}_j = \mathsf{DS.Sign}(\mathsf{DS}.vk_i, \mathsf{DS}.sk_i, C_j) \to \sigma_{j,i^*}$

Step 4: Program the random oracle so that $i^* \in \mathbb{S}$:

Let $\mathsf{UTS}.\sigma$ be the unique threshold signature under rpk for sid.

Sample $s^* \leftarrow\$ [\kappa]$; Let $z_{s^*} = i^*$, and sample $z_1, \ldots, z_{s^*-1}, z_{s^*+1}, \ldots, z_\kappa \leftarrow\$ [n]$

Let $\mathsf{RO}(\mathsf{sid}, \mathsf{UTS}.\sigma) := (z_1, \ldots, z_\kappa)$

Step 5: honestly execute this step on the behalf of all $\{P_i\}_{i\in\mathbb{H}}$

Step 6: honestly execute this step on the behalf of all $\{P_i\}_{i\in\mathbb{H}}$, and obtain Qual

Step 7: **for** every $i \in \mathbb{H}$, compute $\bar{sk}_i = \sum_{z\in\mathsf{Qual}\setminus\{i^*\}} f_z(i)$, and $\bar{pk}_i = g^{\bar{sk}_i}$

for $i \in \mathbb{H}$, compute $pk_i = A_i\bar{pk}_i$; simulate the proof $\pi_i \leftarrow \mathsf{NIZK.SimProve}(\mathsf{cm}_i, pk_i)$

multicast (pk_i, π_i) on behalf of P_i

Step 8: honestly execute this step on the behalf of all $\{P_i\}_{i\in\mathbb{H}}$. In addition:

Compute $\bar{sk} = \sum_{i\in\mathbb{H}} (\bar{sk}_i)^{\bar{\lambda}_i(0)}$, where $\{\bar{\lambda}_i(0)\}_{i\in\mathbb{H}}$ is the Lagrange basis over $\mathbb{H}$

output $(\bar{sk}, 1, 0, \ldots, 0)$ as the description for pk over $(g, D_0, D_1, \ldots, D_t)$

FIGURE 4.14: The simulator for the oracle-aided algebraic simulatability.

The execution in **Hybrid 0** is identical to the real execution. It is also easy to see that, at the view of $\mathcal{A}$, the execution simulated by Sim_2 is identical to the the execution simulated by $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG},\mathsf{ada}}$ on input the uniformly sampled $(D_0, \ldots, D_t)$ and with the access to $\mathsf{DL}_g(\cdot)$.

Due to the zero knowledge property of NIZK, it follows that **Hybrid 0** and **Hybrid 1** are indistinguishable to the adversary $\mathcal{A}$. Regarding **Hybrid 1** and **Hybrid 2**, they are indentical, if Sim_2 does not terminate in **Hybrid 2**. On the other hand, due to the security of the underlying unique threshold signature scheme, the probability that $\mathcal{A}$ can query RO with $(\mathsf{sid}, \mathsf{UTS}.\sigma)$ is negligible. Thus, **Hybrid 1** and **Hybrid 2** are indistinguishable to $\mathcal{A}$. Regarding **Hybrid 2** and **Hybrid 3**, the only difference in $\mathcal{A}$'s view is the distribution of the output of $\mathsf{Election}$. In particular, the output distribution U_2 in **Hybrid 2** is the uniform distribution over $[n]\kappa$, while the distribution U_3 in **Hybrid 3** is the uniform distribution over $[n]^\kappa \setminus \mathbb{C}'^\kappa$, where $\mathbb{C}'$ is the set of corrupted nodes before **Step 4**. It is easy to see the statistic distance between U_2 and U_3 is merely $(\frac{|\mathbb{C}|}{n})^\kappa$, which is negligible. Therefore, the simulation execution provided by $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG},\mathsf{ada}}$ and the real execution are indistinguishable to $\mathcal{A}$.

Then, we argue that $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG},\mathsf{ada}}$ provides the correct expression of pk, *i.e.*, $pk = g^{\bar{sk}} D_0$. Recall that in our DKG, pk is obtained by interpolating the public key shares $\{pk_j\}_{\mathsf{Valid}'}$ with valid proofs obtained by the end of **Step 7**. For $j \in \mathsf{Valid}' \cap \mathbb{H}'$ where $\mathbb{H}'$ is the set of so-far-honest nodes before **Step 7**, pk_j is generated as $A_j g^{\bar{sk}_j}$; For $j \in \mathsf{Valid}' \setminus \mathbb{H}'$, according to the simulation knowledge soundness of NIZk, $\mathcal{A}$ knows the opening (sk'_j, r'_j) of $\mathsf{cm}_j = (\prod_{z \in \mathsf{Qual} \setminus \{i^*\}} \mathsf{cm}_{z,j}) A_j B_j$ and $pk_j = g^{sk'_j}$. As $A_j = g^{a_j}$ and $B_j = h^{b_j}$, we know that $(sk'_j - a_j, r'_j - b_j)$ is the opening for $\bar{\mathsf{cm}}_j = \prod_{z \in \mathsf{Qual} \setminus \{i^*\}} \mathsf{cm}_{z,j}$. On the other hand, for each $i \in \mathbb{H}$, we have an opening $(\bar{sk}_i, \bar{r}_i)$ for $\bar{cm}_i = \prod_{z \in \mathsf{Qual} \setminus \{i^*\}} \mathsf{cm}_{z,i}$. Following Lemma 20, for every $j \in \mathsf{Valid}' \setminus \mathbb{H}'$, we know that $\bar{sk}_j = \sum_{i \in \mathbb{H}} \bar{\lambda}_i(j) \bar{sk}_i$ and $\bar{r}_j = \sum_{i \in \mathbb{H}} \bar{\lambda}_i(j) \bar{r}_i$ is an opening for $\bar{cm}_j$. Then, due to the binding property of the underlying Pedersen commitment, we know $\bar{sk}_j = sk'_j - a_j$ excepet for a negligible probability. Thus, for $j \in \mathsf{Valid}' \setminus \mathbb{H}'$, $pk_j = g^{\bar{sk}_j} A_j$. It follows easily that $pk = \prod_{j \in \mathsf{Valid}} pk_j^{\lambda_j(0)} = g^{\bar{sk}} D_0$.

Finally, let us assume w.l.o.g. that at the end of the execution there are t corrupted party $\mathbb{C} = \{j_1, \ldots, j_t\}$, and then the simulation matrix can be represented as below:

$$L = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ 1 & j_1 & \ldots & j_1^t \\ 1 & \ldots & \ldots & \ldots \\ 1 & j_t & \ldots & j_t^t \end{pmatrix}.$$

It is easy to check L is invertible over $\mathbb{Z}_p$, and thus we complete this proof. □

LEMMA 23. *The DKG protocol in Fig.4.13 satisfies the full secrecy against static aderversaries.*

PROOF. We proceed this proof by constructing an PPT simulator $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG}}$ which simulates an execution of the DKG involving an adversary $\mathcal{A}$. For clarity, we write down the pseudo-code of $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG}}$ in Fig.4.15, which takes as input pk, which is generated by the default key generation algorithm, and the identities of corrupted parties denoted by $\mathbb{C} \subset [n]$. We assume w.l.o.g. that $|\mathbb{C}| = t$.

In the following, we argue that the execution simulated by $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG}}$ and the real execution are indistinguishable to $\mathcal{A}$.

Hybrid 0: The simulator Sim_0 acts on behalf of all honest participants by honestly following the protocol specification.

Hybrid 1: The simulator Sim_1 is almost identical to Sim_0, except that:

- At **step 7**, it generates the proof π_i for each $i \in \mathbb{H}$ by using the simulated prover algorithm $\mathsf{NIZK.SimProve}$.

Hybrid 2: The simulator Sim_2 is almost identical to Sim_1, except that:

- Before **step 4**, it uses secret key shares rsk_i (w.r.t. RDKG) of honest nodes to create the unique threshold signature $\mathsf{UTS}.\sigma$ for sid. Check whether $(\mathsf{sid}, \mathsf{UTS}.\sigma)$ has been queried by $\mathcal{A}$ to the random oracle. If it has been queried, **abort**. Otherwise, it uniformly samples $z_1, \ldots, z_\kappa \leftarrow\!\!\$\ [n]$, and records $\mathsf{RO}(\mathsf{sid}, \mathsf{UTS}.\sigma) := (z_1, \ldots, z_\kappa)$.

Hybrid 3: The simulator Sim_3 is almost identical to Sim_2, except that:

- At **step 1**, it samples $i^* \leftarrow\!\!\$\ \mathbb{H}$. At **step 4**, it programs the random oracle to ensure that i^* is included in the election outcome $\mathbb{S}$, as $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG},\mathsf{ada}}$ does in **step 4**.

Hybrid 4: The simulator Sim_4 is almost identical to Sim_3, except that:

- At **step 1**, after generating $C_j = \mathsf{cm}_{i^*,j}$ for all $j \in \mathbb{C}$, P_{i^*} computes $C_0 = g^{f_{i^*}(0)} h^{f'_{i^*}(0)}$, and $\mathsf{cm}_{i^*,i} = \prod_{j \in \mathbb{C} \cup \{0\}} C_j^{\lambda_j(i)}$, where $\{\lambda_j(x)\}_{j \in \mathbb{C} \cup \{0\}}$ is the Lagrange basis over $\mathbb{C} \cup \{0\}$. At **step 3**, C_i for $i \in \mathbb{H}$ is signed by using the signing key of P_i.

$\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG}}(pk, \mathbb{C})$

Prepare: define $\mathbb{H} := [n] \setminus \mathbb{C}$, sample $i^* \leftarrow\$ \mathbb{H}$

Step 0: honestly execute this step on the behalf of all $\{P_i\}_{i\in\mathbb{H}}$

Step 1: honestly execute this step on the behalf of all $\{P_i\}_{i\in\mathbb{H}\setminus\{i^*\}}$;

For P_{i^*}, do the following:

for $j \in \mathbb{C}$, sample $a_j, b_j \leftarrow\$ \mathbb{Z}_p$, and compute $C_j = g^{a_j} h^{b_j}$

Sample $C_0 \leftarrow\$ \mathbb{G}$, compute $C_j = \prod_{\tau\in\{0\}\cup\mathbb{C}} C_\tau^{\lambda_\tau(j)}$ for $j \in [n] \setminus \mathbb{C}$

send $(\mathsf{sid}, \textsc{Share}, C_j, a_j, b_j)$ to P_j, for each $j \in \mathbb{C}$

Step 2: honestly execute this step on the behalf of all $\{P_i\}_{i\in\mathbb{H}}$

Step 3: honestly execute this step on the behalf of all $\{P_i\}_{i\in\mathbb{H}\setminus\{i^*\}}$;

For P_{i^*}, follow the protocol except that:

for every $j \in \mathbb{H}$, let $\mathsf{cert}_j = \mathsf{DS.Sign}(\mathsf{DS}.vk_i, \mathsf{DS}.sk_i, C_j) \to \sigma_{j,i^*}$

Step 4: Program the random oracle so that $i^* \in \mathbb{S}$:

Let $\mathsf{UTS}.\sigma$ be the unique threshold signature under rpk for sid.

Sample $s^* \leftarrow\$ [\kappa]$; Let $z_{s^*} = i^*$

Sample $z_1, \ldots, z_{s^*-1}, z_{s^*+1}, \ldots, z_\kappa \leftarrow\$ [n]$

Let $\mathsf{RO}(\mathsf{sid}, \mathsf{UTS}.\sigma) := (z_1, \ldots, z_\kappa)$

Step 5: honestly execute this step on the behalf of all $\{P_i\}_{i\in\mathbb{H}}$

Step 6: honestly execute this step on the behalf of all $\{P_i\}_{i\in\mathbb{H}}$,

and obtain Qual

Step 7:

for every $j \in \mathbb{C}$, compute $pk_j^{(i^*)} = g^{a_j}$, where a_j

is sent on the behalf of P_{i^*} in Step 1

for every $i \in \mathbb{H}$, compute $\bar{sk}_i = \sum_{z\in\mathsf{Qual}\setminus\{i^*\}} f_z(i)$, and $\bar{pk}_i = g^{\bar{sk}_i}$.

Compute $\bar{pk} = \prod_{i\in\mathbb{H}} \bar{pk}_i^{\bar{\lambda}_i(0)}$, where $\{\bar{\lambda}_i(x)\}_{i\in\mathbb{H}}$

is the Lagrange basis over $\mathbb{H}$

Compute $pk_0^{(i^*)} = \dfrac{pk}{\bar{pk}}$

for $i \in \mathbb{H}$, compute $pk_i^{(i^*)} = \prod_{j\in\{0\}\cup\mathbb{C}} (pk_j^{(i^*)})^{\hat{\lambda}_j(i)}$,

where $\{\hat{\lambda}_j(x)\}_{j\in\{0\}\cup\mathbb{C}}$ is the Lagrange basis over $\{0\} \cup \mathbb{C}$

for $i \in \mathbb{H}$, compute $pk_i = pk_i^{(i^*)} \bar{pk}_i$

simulate the proof $\pi_i \leftarrow \mathsf{NIZK.SimProve}(\mathsf{cm}_i, pk_i)$

multicast (pk_i, π_i) on the behalf of P_i

Step 8: honestly execute this step on the behalf of all $\{P_i\}_{i\in\mathbb{H}}$

FIGURE 4.15: The simulator for the full secrecy.

- At **step 7**, for every $i \in \mathbb{H}$, pk_i is calculated as follows: First, calculate $\bar{pk}_i = g^{\sum_{z \in \mathsf{Qual} \setminus \{i^*\}} f_z(i)}$. Next, let A_0 be $g^{f_{i^*}(0)}$, and interpolate $pk_i^{(i^*)} = A_0^{\lambda_0(i)} \prod_{j \in \mathbb{C}} g^{a_j \lambda_j(i)}$, where a_j (along with some b_j) for $j \in \mathbb{C}$ is the opening for $\mathsf{cm}_{i^*,j}$ sent to P_j at **step 1**. Then, let $pk_i = \bar{pk}_i pk_i^{(i^*)}$.

Hybrid 5: The simulator Sim_5 is almost identical to Sim_4, except that:

- At **step 1**, P_{i^*} does not explicitly sample f_{i^*}. Instead, it samples $A_0 \leftarrow\!\!\$\ \mathbb{G}$, and $\{a_j\}_{j \in \mathbb{C}} \leftarrow\!\!\$\ \mathbb{Z}_p^t$, which implicitly decides f_{i^*}, s.t. $f_{i^*}(j) = a_j$, and $A_0 = g^{f_{i^*}(0)}$. It also does not sample f'_{i^*}. Instead, it samples $\{b_j\}_{j \in \{0\} \cup \mathbb{C}} \leftarrow\!\!\$\ \mathbb{Z}_p^{t+1}$, and computes $C_0 = A_0 h^{b_0}$.

Hybrid 6: The simulator Sim_6 is almost identical to Sim_5, except that:

- At **step 1**, P_{i^*} does not sample A_0. C_0 is uniformly sampled from $\mathbb{G}$.
- At **step 7**, A_0 is randomly chosen.

The execution in **Hybrid 0** is identical to a real execution. We also note that **Hybrid 6** is identical to the simulated execution provided by $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG}}$ in the view of $\mathcal{A}$, as when the input pk is uniformly distributed, $pk_0^{(i^*)} = \frac{pk}{\bar{pk}}$ is also uniformly distributed. Following the similar arguments in the proof for Lemma 22, **Hybrid 0** and **Hybrid 3** are indistinguishable. Through a sanity check, **Hybrid 3**, **Hybrid 4**, and **Hybrid 5** are identical in the view of $\mathcal{A}$. Moreover, **Hybrid 5** and **Hybrid 6** are identical due to the perfect hiding property of the underlying commitment scheme. Therefore, **Hybrid 0** and **Hybrid 6** are indistinguishable in the view of $\mathcal{A}$.

Finally, we show that in the simulated execution by $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG}}$, the output pk' is exactly the input public key pk of $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG}}$. Note that, for every $i \in \mathbb{H}$, $pk_i = pk_i^{(i^*)} \bar{pk}_i$. It follows that $pk = pk_0^{(i^*)} \bar{pk} = \prod_{i \in \mathbb{H}} (pk_i^{(i^*)})^{\lambda_i(0)} (\bar{pk}_i)^{\lambda_i(0)} = \prod_{i \in \mathbb{H}} (pk_i)^{\lambda_i(0)}$, where $\{\lambda_i(x)\}_{i \in \mathbb{H}}$ is the Lagrange basis over $\mathbb{H}$. In **step 8**, pk' is generated by interpolating the first $t+1$ public key shares with valid proofs, denoted by $\{pk_j\}_{j \in \mathsf{Valid}'}$. Due to the simulation knowledge soundness of NIZK, $\mathcal{A}$ knows an opening (sk'_j, r'_j) for $\mathsf{cm}_j = \prod_{z \in \mathsf{Qual}} \mathsf{cm}_{z,j}$ and $pk_j = g^{sk'_j}$. On the other hand, following Lemma 20, we know that $(sk_j = \sum_{i \in \mathbb{H}} \lambda_i(j) \bar{sk}_i + a_j, *)$ is also an opening for cm_j. Due to the binding property of Pedersen commitment, we have $sk_j = sk'_j$ and $pk_j = g^{\sum_{i \in \mathbb{H}} \lambda_i(j) \bar{sk}_i + a_j} = \prod_{i \in \mathbb{H}} (pk_i)^{\lambda_i(j)}$. In other workds, all valid pk_j's are on the t-degree polynomial defined by $\{pk_i\}_{i \in \mathbb{H}}$, thus it follows that $pk' = pk$. $\square$

CHAPTER 5

Conclusion and Future Works

Distributed key generation (DKG) is a fundamental cryptographic primitive that supports threshold cryptography and widespread applications in blockchain, such as cross-chain bridge, MEV protection and checkpointing into Bitcoin. Those new applications raise a new fundamental challenge of deploying distributed key generation on a very large scale.

In this thesis, we identify four main obstacles on the way towards large scale distributed key generation, namely, (1) classic DKGs suffer from high communication and computation complexity due to heavy computation and broadcast overhead in adversarial cases; (2) Recent efficient DKGs cannot achieve both adaptive security and generate field-element secrets at the same time; (3) the round complexity of DKG is linear to n due to underlying deterministic consensus protocols; and (4) DKG has weak support in weighted validator settings over PoS blockchains.

To address these challenges, we provide insights on both practical and theoretical landscape.

In Chapter 3, we present Any-Trust DKG, a practical, scalable and adaptively secure DKG protocol with field-element secrets and intend to solve challenge (1) and (2). By assuming a common coin and a broadcast channel, our Any-Trust DKG achieves (quasi-)linear computation and broadcast overhead per-node cost against weak adaptive adversaries. The key idea is to randomly sample an *Any-Trust* group, a small group of nodes among which there is at least one honest node with overwhelming probability, and delegate most heavy communication and computation tasks to this group. The population only trusts that at least one member in the group is honest, without knowing which one. Besides, in Section 3.8, we introduce a sub-ID framework that addresses challenge (3) to deploy our DKG in a weighted setting without compromising efficiency. Along with sub-ID method, our Any-Trust DKG leads to a fully practical instantiation of Filecoin's checkpointing mechanism, in which *all* validators of a Proof-of-Stake (PoS) blockchain periodically run DKG and threshold signing to create checkpoints on Bitcoin, to enhance the security of the PoS chain.

In Chapter 4, on the theoretical side, we purpose Circular Dragon, the first construction of adaptively secure DKG protocol to solve challenge (1), (2) and (3). Our Circular Dragon realizes optimal $n/2$ resilience in the synchronous network, and attains near-optimal asymptotic complexities, i.e., $\widetilde{O}(n^2)$ communication and $\widetilde{O}(1)$ rounds. In addition, the proposed DKG protocol also produces field-element secrets to support the standard discrete-logarithm based threshold cryptosystem. Last but not least, the practicality of Circular Dragon is yet to be assessed. Our solution indeed reduces the round complexity theoretically, however **concrete** improvements can only be observed in the asymptotic manner. This means super large network size is anticipated even for loose security parameters. Claiming real-world practicality of sub-linear round DKG still remains as an open problem.

Both our approaches assume a synchronous network, leading to a future challenge: designing a scalable, adaptively secure DKG for asynchronous settings that better reflect real Internet conditions and are truly deployable for production. Another interesting theoretical question is whether we can remove the $\log n$ term in communication complexity of Circular Dragon DKG, and have a DKG protocol with exact $O(n^2\lambda)$ communication complexity, matching the lower bound established in (Shrestha et al., 2023), or whether this lower bound is tight. Yet another broader question is how to adapt DKG and threshold protocols to other cryptographic systems, like lattice-based ones, while maintaining our performance and security metrics. Finally, another interesting question is that whether our techniques, including forming consortium via partitioning and recursively bootstrapping randomized consensus, can find broader applications beyond DKG; thus whether we may simplify our DKGs, for example, by identifying a new way of organizing consortiums, without comprising efficiency and security.

Bibliography

Ittai Abraham, Renas Bacho, Julian Loss, and Gilad Stern. 2025. Nearly quadratic asynchronous distributed key generation. Cryptology ePrint Archive, Paper 2025/006.

Ittai Abraham, T.-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. 2023a. Communication complexity of byzantine agreement, revisited. *Distributed Comput.*, 36(1):3–28.

Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. 2019. Synchronous byzantine agreement with expected O(1) rounds, expected o(n^2) communication, and optimal resilience. In *Financial Cryptography*, volume 11598 of *Lecture Notes in Computer Science*, pages 320–334. Springer.

Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. 2023b. Bingo: Adaptivity and asynchrony in verifiable secret sharing and distributed key generation. In *CRYPTO (1)*, volume 14081 of *LNCS*, pages 39–70. Springer.

Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. 2021a. Reaching consensus for asynchronous distributed key generation. In *PODC*, pages 363–373. ACM.

Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. 2020. Sync hotstuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 106–118. IEEE.

Ittai Abraham, Kartik Nayak, and Nibesh Shrestha. 2021b. Optimal good-case latency for rotating leader synchronous bft. *Cryptology ePrint Archive*.

Ittai Abraham, Kartik Nayak, and Nibesh Shrestha. 2023c. Communication and round efficient parallel broadcast protocols. *Cryptology ePrint Archive*.

Thomas Attema, Ronald Cramer, and Matthieu Rambaud. 2021. Compressed $\varsigma $-protocols for bilinear group arithmetic circuits and application to logarithmic transparent threshold signatures. In *ASIACRYPT (4)*, volume 13093 of *LNCS*, pages 526–556. Springer.

Sarah Azouvi and Marko Vukolic. 2022. Pikachu: Securing pos blockchains from long-range attacks by checkpointing into bitcoin pow using taproot. In *ConsensusDay@CCS*, pages 53–65. ACM.

Renas Bacho, Daniel Collins, Chen-Da Liu-Zhang, and Julian Loss. 2023a. Network-agnostic security comes (almost) for free in DKG and MPC. In *CRYPTO (1)*, volume 14081 of *Lecture Notes in Computer Science*, pages 71–106. Springer.

Renas Bacho, Christoph Lenzen, Julian Loss, Simon Ochsenreither, and Dimitrios Papachristoudis. 2023b. Grandline: Adaptively secure DKG and randomness beacon with (almost) quadratic communication complexity. *IACR Cryptol. ePrint Arch.*, page 1887.

Renas Bacho, Christoph Lenzen, Julian Loss, Simon Ochsenreither, and Dimitrios Papachristoudis. 2023c. GRandLine: Adaptively secure DKG and randomness beacon with (log-)quadratic communication complexity. Cryptology ePrint Archive, Paper 2023/1887. Version: September 3, 2024.

Renas Bacho, Christoph Lenzen, Julian Loss, Simon Ochsenreither, and Dimitrios Papachristoudis. 2024. Grandline: Adaptively secure DKG and randomness beacon with (almost) quadratic communication complexity. In *CCS*. ACM.

Renas Bacho and Julian Loss. 2022. On the adaptive security of the threshold BLS signature scheme. In *CCS*, pages 193–207. ACM.

Renas Bacho and Julian Loss. 2023a. Adaptively secure (aggregatable) PVSS and application to distributed randomness beacons. In *CCS*, pages 1791–1804. ACM.

Renas Bacho and Julian Loss. 2023b. Adaptively secure (aggregatable) pvss and application to distributed randomness beacons. In *CCS*. ACM.

Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. 2018. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *CCS*, pages 913–930. ACM.

Joonsang Baek and Yuliang Zheng. 2003. Simple and efficient threshold cryptosystem from the gap diffie-hellman group. In *GLOBECOM*, pages 1491–1495. IEEE.

Leemon Baird, Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. 2023. Threshold signatures in the multiverse. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1454–1470. IEEE.

Mihir Bellare, Alexandra Boldyreva, Kaoru Kurosawa, and Jessica Staddon. 2007. Multirecipient encryption schemes: How to save on bandwidth and computation without sacrificing security. *IEEE Trans. Inf. Theory*, 53(11):3927–3943.

Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. 2022. Better than advertised security for non-interactive threshold signatures. In *CRYPTO (4)*, volume 13510 of *LNCS*, pages 517–550. Springer.

Mihir Bellare and Phillip Rogaway. 1993. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, pages 62–73. ACM.

Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. 2020. Can a public blockchain keep a secret? In *TCC (1)*, volume 12550 of *Lecture Notes in Computer Science*, pages 260–290. Springer.

Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Alex Miao, and Tal Rabin. 2022. Threshold cryptography as a service (in the multiserver and YOSO models). In *CCS*, pages 323–336. ACM.

Adithya Bhat, Aniket Kate, Kartik Nayak, and Nibesh Shrestha. 2023. Optrand: Optimistically responsive distributed random beacons. In *NDSS*.

Adithya Bhat, Nibesh Shrestha, Zhongtang Luo, Aniket Kate, and Kartik Nayak. 2021. Randpiper - reconfiguration-friendly random beacons with quadratic communication. In *CCS*, pages 3502–3524. ACM.

Richard E Blahut. 1983. *Theory and practice of error control codes*. Addison-Wesley.

G. R. Blakley. 1979. Safeguarding cryptographic keys. In *1979 International Workshop on Managing Requirements Knowledge (MARK)*, pages 313–318.

Aptos Blockchain. 2021. Aptosan. `https://aptoscan.com`.

Alexandra Boldyreva. 2003. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer.

Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short signatures from the weil pairing. In *ASIACRYPT*, volume 2248 of *LNCS*, pages 514–532. Springer.

Gabriel Bracha. 1987. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143.

Carlo Brunetta, Hans Heum, and Martijn Stam. 2024. Sok: Public key encryption with openings. In *PKC (4)*, volume 14604 of *LNCS*, pages 35–68. Springer.

Christian Cachin, Klaus Kursawe, and Victor Shoup. 2000. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract). In *PODC*, pages 123–132. ACM.

Christian Cachin, Klaus Kursawe, and Victor Shoup. 2005. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *J. Cryptol.*, 18(3):219–246.

Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. 1996. Adaptively secure multi-party computation. In *STOC*, pages 639–648. ACM.

Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 1999. Adaptive security for threshold cryptosystems. In *CRYPTO*, volume 1666 of *LNCS*, pages 98–115. Springer.

Ignacio Cascudo and Bernardo David. 2017. SCRAPE: scalable randomness attested by public entities. In *ACNS*, volume 10355 of *LNCS*, pages 537–556. Springer.

Andrea Cerulli, Aisling Connolly, Gregory Neven, Franz-Stefan Preiss, and Victor Shoup. 2023. vetkeys: How a blockchain can keep many secrets. Cryptology ePrint Archive, Paper 2023/616. `https://eprint.iacr.org/2023/616`.

Melissa Chase and Anna Lysyanskaya. 2006. On signatures of knowledge. In *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 78–96. Springer.

David Chaum and Torben P. Pedersen. 1992. Wallet databases with observers. In *CRYPTO*, volume 740 of *LNCS*, pages 89–105. Springer.

Jing Chen and Silvio Micali. 2019. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183.

Kevin Choi, Aathira Manoj, and Joseph Bonneau. 2023. Sok: Distributed randomness beacons. In *SP*, pages 75–92. IEEE.

Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. 1985. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 383–395. ISSN: 0272-5428.

Pierre Civit, Muhammad Ayaz Dzulfikar, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. 2024. DARE to agree: Byzantine agreement with optimal resilience and adaptive communication. In *PODC*, pages 145–156. ACM.

Ran Cohen, Jack Doerner, Eysa Lee, Anna Lysyanskaya, and Lawrence Roy. 2024. An unstoppable ideal functionality for signatures and a modular analysis of the dolev-strong broadcast. Cryptology ePrint Archive, Paper 2024/1807.

Cosmos. 2016. `https://cosmos.network`.

Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. 2001. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 280–299. Springer.

Elizabeth C. Crites, Chelsea Komlo, and Mary Maller. 2023. Fully adaptive schnorr threshold signatures. In *CRYPTO (1)*, volume 14081 of *Lecture Notes in Computer Science*, pages 678–709. Springer.

Sourav Das, Philippe Camacho, Zhuolun Xiang, Javier Nieto, Benedikt Bünz, and Ling Ren. 2023a. Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 356–370.

Sourav Das and Ling Ren. 2024. Adaptively secure BLS threshold signatures from DDH and co-cdh. In *CRYPTO (7)*, volume 14926 of *Lecture Notes in Computer Science*, pages 251–284. Springer.

Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. 2023b. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. In *USENIX Security Symposium*, pages 5359–5376. USENIX Association.

Sourav Das, Zhuolun Xiang, Alin Tomescu, Alexander Spiegelman, Benny Pinkas, and Ling Ren. 2023c. Verifiable secret sharing simplified. Cryptology ePrint Archive, Paper 2023/1196.

Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. 2022. Practical asynchronous distributed key generation. In *SP*, pages 2518–2534. IEEE.

Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. 2018. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT (2)*, volume 10821 of *LNCS*, pages 66–98. Springer.

Luciano Freitas de Souza and Andrei Tonkikh. 2023. Swiper and dora: efficient solutions to weighted distributed problems. *CoRR*, abs/2307.15561.

Danny Dolev and Rüdiger Reischuk. 1982. Bounds on information exchange for byzantine agreement. In *PODC*, pages 132–140. ACM.

Danny Dolev and H. Raymond Strong. 1983. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666.

DRand. 2023. A distributed randomness beacon daemon - go implementation. `https://github.com/drand/drand`.

Fatima Elsheimy, Giorgos Tsimos, and Charalampos Papamanthou. 2024. Deterministic byzantine agreement with adaptive $O(n \cdot f)$ communication. In *SODA*, pages 1120–1146. SIAM.

Paul Feldman. 1987. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, pages 427–437. IEEE Computer Society.

Paul Feldman and Silvio Micali. 1988. Optimal algorithms for byzantine agreement. In *STOC*, pages 148–161. ACM.

Hanwen Feng, Yingzi Gao, Yuan Lu, Qiang Tang, and Jing Xu. 2025. Practical asynchronous distributed key reconfiguration and its applications. *Cryptology ePrint Archive*.

Hanwen Feng, Zhenliang Lu, and Qiang Tang. 2024a. Dragon: Decentralization at the cost of representation after arbitrary grouping and its applications to sub-cubic DKG and interactive consistency. In *PODC*, pages 469–479. ACM.

Hanwen Feng, Tiancheng Mai, and Qiang Tang. 2024b. Scalable and adaptively secure any-trust distributed key generation and all-hands checkpointing. In *ACM CCS*.

Hanwen Feng and Qiang Tang. 2024. Asymptotically optimal adaptive asynchronous common coin and DKG with silent setup. Cryptology ePrint Archive, Paper 2024/2098.

Amos Fiat and Adi Shamir. 1986. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer.

Filecoin. 2017. `https://filecoin.io/`.

Michael J. Fischer and Nancy A. Lynch. 1982. A lower bound for the time to assure interactive consistency. *Inf. Process. Lett.*, 14(4):183–186.

Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. 1985. Easy impossibility proofs for distributed consensus problems. In *PODC*, pages 59–70. ACM.

Pierre-Alain Fouque and Jacques Stern. 2001. One round threshold discrete-log key generation without private channels. In *Public Key Cryptography*, volume 1992 of *LNCS*, pages 300–316. Springer.

Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Efficient asynchronous byzantine agreement without private setups. In *ICDCS*, pages 246–257. IEEE.

Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT (2)*, volume 9057 of *LNCS*, pages 281–310. Springer.

Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. 2024. hints: Threshold signatures with silent setup. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3034–3052. IEEE.

Rosario Gennaro and Steven Goldfeder. 2018. Fast multiparty threshold ECDSA with fast trustless setup. In *CCS*, pages 1179–1194. ACM.

Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 2007. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptol.*, 20(1):51–83.

Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. 2021a. YOSO: you only speak once - secure MPC with stateless ephemeral roles. In *CRYPTO (2)*, volume 12826 of *LNCS*, pages 64–93. Springer.

Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. 2022. Practical non-interactive publicly verifiable secret sharing with thousands of parties. In *EUROCRYPT (1)*, volume 13275 of *LNCS*, pages 458–487. Springer.

Craig Gentry, Shai Halevi, Bernardo Magri, Jesper Buus Nielsen, and Sophia Yakoubov. 2021b. Random-index PIR and applications. In *TCC (3)*, volume 13044 of *LNCS*, pages 32–61. Springer.

Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP*, pages 51–68. ACM.

Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, and Leonid Reyzin. 2016. NSEC5 from elliptic curves: Provably preventing DNSSEC zone enumeration with shorter responses. *IACR Cryptol. ePrint Arch.*, page 83.

Jens Groth. 2021. Non-interactive distributed key generation and key resharing. *IACR Cryptol. ePrint Arch.*, page 339.

Jens Groth and Victor Shoup. 2023. Fast batched asynchronous distributed key generation. Technical report, Cryptology ePrint Archive. `https://eprint.iacr.org/2023/1175`.

Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2020. Dumbo: Faster asynchronous BFT protocols. In *CCS*, pages 803–818. ACM.

Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. 2021. Aggregatable distributed key generation. In *EUROCRYPT (1)*, volume 12696 of *LNCS*, pages 147–176. Springer.

Gene Itkis and Leonid Reyzin. 2001. Forward-secure signatures with optimal signing and verifying. In *CRYPTO*, volume 2139 of *LNCS*, pages 332–354. Springer.

Aniket Kate, Easwar Vivek Mangipudi, Pratyay Mukherjee, Hamza Saleem, and Sri Aravinda Krishnan Thyagarajan. 2023. Non-interactive VSS using class groups and application to DKG. Cryptology ePrint Archive, Paper 2023/451. `https://eprint.iacr.org/2023/451`.

Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer.

Jonathan Katz. 2024. Round-optimal, fully secure distributed key generation. In *CRYPTO (7)*, volume 14926 of *Lecture Notes in Computer Science*, pages 285–316. Springer.

Jonathan Katz and Chiu-Yuen Koo. 2006. On expected constant-round protocols for byzantine agreement. In *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 445–462. Springer.

Dafna Kidron and Yehuda Lindell. 2011. Impossibility results for universal composability in public-key models and with fixed inputs. *J. Cryptol.*, 24(3):517–544.

Eleftherios Kokoris-Kogias, Dahlia Malkhi, and Alexander Spiegelman. 2020. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In *CCS*, pages 1751–1767. ACM.

Chelsea Komlo and Ian Goldberg. 2020. FROST: flexible round-optimized schnorr threshold signatures. In *SAC*, volume 12804 of *LNCS*, pages 34–65. Springer.

Sung-Shine Lee, Alexandr Murashkin, Martin Derka, and Jan Gorzny. 2023. Sok: Not quite water under the bridge: Review of cross-chain bridge hacks. In *ICBC*, pages 1–14. IEEE.

Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. 2002. On the composition of authenticated byzantine agreement. In *STOC*, pages 514–523. ACM.

Yehuda Lindell and Ariel Nof. 2018. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *CCS*, pages 1837–1854. ACM.

Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. 2020. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *PODC*, pages 129–138. ACM.

Dahlia Malkhi and Pawel Szalachowski. 2022. Maximal extractable value (MEV) protection on a DAG. In *Tokenomics*, volume 110 of *OASIcs*, pages 6:1–6:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. 1999. Verifiable random functions. In *FOCS*, pages 120–130. IEEE Computer Society.

Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *CCS*, pages 31–42. ACM.

Atsuki Momose and Ling Ren. 2021. Optimal communication complexity of authenticated byzantine agreement. In *DISC*, volume 209 of *LIPIcs*, pages 32:1–32:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Moni Naor and Moti Yung. 1990. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC*, pages 427–437. ACM.

Kartik Nayak, Ling Ren, Elaine Shi, Nitin H. Vaidya, and Zhuolun Xiang. 2020. Improved extension protocols for byzantine broadcast and agreement. In *DISC*, volume 179 of *LIPIcs*, pages 28:1–28:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Jesper Buus Nielsen. 2002. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *CRYPTO*, volume 2442 of *LNCS*, pages 111–126. Springer.

Torben P. Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer.

Polkadot. 2016. `https://www.polkadot.network/`.

Tian Qiu and Qiang Tang. 2023. Predicate aggregate signatures and applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 279–312. Springer.

Irving S Reed and Gustave Solomon. 1960. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304.

Ruby.Exchange. 2021. How skale solves the front-running problem. `https://blog.ruby.exchange/how-skale-solves-the-front-running-problem/?ref=blog.pantherprotocol.io`.

Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. 2020. Hydrand: Efficient continuous distributed randomness. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 73–89. IEEE.

Berry Schoenmakers. 1999. A simple publicly verifiable secret sharing scheme and its application to electronic. In *CRYPTO*, volume 1666 of *LNCS*, pages 148–164. Springer.

Adi Shamir. 1979. How to share a secret. *Commun. ACM*, 22(11):612–613.

Victor Shoup. 2023. The many faces of schnorr. Cryptology ePrint Archive, Paper 2023/1019. `https://eprint.iacr.org/2023/1019`.

Nibesh Shrestha, Adithya Bhat, Aniket Kate, and Kartik Nayak. 2023. Synchronous distributed key generation without broadcasts. Cryptology ePrint Archive, Paper 2021/1635. `https://eprint.iacr.org/2021/1635`.

Selma Steinhoff, Chrysoula Stathakopoulou, Matej Pavlovic, and Marko Vukolic. 2021. BMS: secure decentralized reconfiguration for blockchain and BFT systems. *CoRR*, abs/2109.03913.

Ertem Nusret Tas, David Tse, Fangyu Gai, Sreeram Kannan, Mohammad Ali Maddah-Ali, and Fisher Yu. 2023. Bitcoin-enhanced proof-of-stake security: Possibilities and impossibilities. In *SP*, pages 126–145. IEEE.

Tezos. 2018. `https://tezos.com`.

Sri Aravinda Krishnan Thyagarajan, Guilhem Castagnos, Fabien Laguillaumie, and Giulio Malavolta. 2021. Efficient CCA timed commitments in class groups. In *CCS*, pages 2663–2684. ACM.

Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan-Gueta, and Srinivas Devadas. 2020. Towards scalable threshold cryptosystems. In *IEEE Symposium on Security and Privacy*, pages 877–893. IEEE.

Andrei Tonkikh and Luciano Freitas de Souza. 2024. Swiper: a new paradigm for efficient weighted distributed protocols. In Ran Gelles, Dennis Olivetti, and Petr Kuznetsov, editors, *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC 2024, Nantes, France, June 17-21, 2024*, pages 283–294. ACM.

Total-blockchain. 2022. Osmosis will soon be frontrunning mev free. `https://medium.com/@totalblockchainemail/osmosis-will-soon-be-frontrunning-mev-free-b7da89f04ce9`.

Dennis Trautwein, Aravindh Raman, Gareth Tyson, Ignacio Castro, Will Scott, Moritz Schubotz, Bela Gipp, and Yiannis Psaras. 2022. Design and evaluation of IPFS: a storage layer for the decentralized web. In *SIGCOMM*, pages 739–752. ACM.

Gang Wang and Mark Nixon. 2020. Randchain: Practical scalable decentralized randomness attested by blockchain. In *Blockchain*, pages 442–449. IEEE.

David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. 2012. Scalable anonymous group communication in the anytrust model. In *European Workshop on System Security (EuroSec)*, volume 4.

Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew Miller. 2022. hbacss: How to robustly share many secrets. In *NDSS*. The Internet Society.

Jiaheng Zhang, Tiancheng Xie, Thang Hoang, Elaine Shi, and Yupeng Zhang. 2022. Polynomial commitment with a one-to-many prover and applications. In *USENIX Security Symposium*, pages 2965–2982. USENIX Association.

APPENDIX A

Supplementary Materials for Any-Trust DKG

A.1 Supplementary Materials for Bitcoin Checkpointing

A.1.1 The Blueprint of Pikachu

Long-range attacks against PoS blockchain. Unlike proof-of-work chains, block creation in PoS systems is both costless (in terms of physical resources like energy) and timeless (unconstrained by time limits), which enables adversaries to easily fork a chain. Existing PoS chains prevent malicious forking by punishing misbehavior validators. However, an attacker can choose to present the fork chain after all its stakes have been withdrawn, thus free of being slashed, What is worse, a late coming client may not be able to decide the canonical chain among the forks.

Securing PoS with Bitcoin checkpointing. A few works (Azouvi and Vukolic, 2022; Tas et al., 2023) have shown that long-range attacks can be effectively mitigated by creating checkpoints of the PoS chain on a PoW chain, such that a late coming client can distinguish the canonical chain among forks. Pikachu illustrates a threshold signature-based checkpointing mechanism. At a high level, the lifetime of the PoS system is divided into multiple epochs, and checkpoints are supposed to be created per epoch. At every epoch i, a configuration $C_i = \{(\mathcal{V}_{i,j}, w_{i,j})\}_{j\in[n_i]}$ for some integer n_i, which is the set of all validators $\{\mathcal{V}_{i,j}\}_{j\in[n_i]}$ with their weights $\{w_{i,j}\}_{j\in[n_i]}$, is associated with a public key Q_i (w.r.t. Schnorr signature scheme) which can serve as a Bitcoin address, while the secret key of Q_i is secretly shared among C_i. At epoch $i+1$, validators in C_i will jointly create a Bitcoin transaction which transfers all assets on Q_i to Q_{i+1}, the address belongs to the current configuration C_{i+1}; This transaction is the checkpoint. We elucidate their design with the following three algorithms/protocols[1].

[1]Slightly different from their original description where the PoS digest is embedded into the Bitcoin address, we choose to put it in OP_RETURN for simplicity.

- $\mathsf{AllocateSubID}(C) \rightarrow \{d_j\}_{j\in[n]}$.The sub-identity allocation algorithm takes input as a configuration

$$C = \{(\mathcal{V}_j, w_j)\}_{j\in[n]}$$

and determines the number of sub-identities d_j for each $\mathcal{V}_j$ according to their weight w_j.

- $\mathsf{DKG}(\{(\mathcal{V}_j, d_j)\}_{j\in[n]})$. The validators in C run a DKG protocol, while each sub-identity is viewed as an independent participant. Therefore, each validator $\mathcal{V}_j$ obtains d_j pairs of $(pk_{j,z}, sk_{j,z})_{z\in[d_j]}$, and all validators obtain the same public key $Q = pk$ and the list of public key shares $\vec{pk} = (pk_{j,z})_{j\in[n],z\in[d_j]}$.
- $\mathsf{CreateCKP}(C_i, \mathsf{ckp}, \mathsf{PreAdd}, Q_{i+1}) \rightarrow \mathsf{TX}$. At the epoch $i+1$, assume that validators in C_{i+1} have generated the public key Q_{i+1}, the digest of PoS block to be checkpointed is ckp, and the address of the last checkpointing Bitcoin transaction is PreAdd. Then, the validators in C_i invoke a Threshold Schnorr protocol to sign a Bitcoin transaction TX with the following information.

$$\{\mathsf{Input} : \mathsf{PreAdd}; \mathsf{Output} : Q_{i+1}; \mathsf{OP\_Return} : \mathsf{ckp}\}.$$

Once the transaction has been properly signed, every validator should disseminate it to the Bitcoin network.

With checkpoints on Bitcoin, it is rather straightforward for a late-coming user to decide which fork is the canonical chain when the user is provided with a block tree of finalized PoS blocks. Specifically, the user first synchronizes with the Bitcoin blockchain. Then, it finds the initial checkpoint transaction and builds a chain of transactions following the initial transaction. Next, it obtains the digest ckp from the latest checkpoint transaction and decides the fork with the block whose digest is ckp as the canonical chain. Moreover, while other approaches like key-evolving forward-secure signatures (Chen and Micali, 2019; David et al., 2018) may also mitigate long-range attacks, the checkpointing mechanism enjoys the unique advantage of ensuring malicious validators are always slashable. We defer a detailed discussion to Section A.1.2.

A.1.2 Security of Checkpointing

This paradigm has been thoroughly analyzed in (Azouvi and Vukolic, 2022). It considers an efficient adversary $\mathcal{A}$, which at each epoch i can corrupt all validators in previous configurations $\{C_j\}_{j<i-L}$ and a fraction of validators up to f in "recent" configurations $\{C_j\}_{i-L<j\leq i}$, for some parameter L such that

the checkpoint transaction for epoch i_0 will be confirmed in Bitcoin by epoch $i_0 + L$. Such an adversary can mount long-range attacks by using the previous secret keys to forge another validate-looking chain (called a long-range attack chain). However, since the Bitcoin blockchain has recorded transactions that transferred all assets from previous addresses $\{Q_j\}_{j<i-L}$, $\mathcal{A}$ cannot create valid checkpoints using secret keys of $\{Q_j\}_{j<i-L}$. Therefore, a bootstrapping client can decide the canonical chain with Bitcoin checkpoints. We summarize their results in the following theorem.

THEOREM 5 ((Azouvi and Vukolic, 2022)). *Assume both the Bitcoin blockchain and the PoS chain satisfy consistency, chain growth, and chain quality (as defined in (Garay et al., 2015)). Assume the Threshold Schnorr signature satisfies unforgeability and robustness under the DKG protocol against $\mathcal{A}$ corrupting up to t sub-identities, and* AllocateSubID *allocates at most t sub-IDs to $\mathcal{A}$. Then, the checkpointing mechanism satisfies the following properties.*

- *Safety. $\mathcal{A}$ cannot produce any valid checkpointing transactions for long-range attack chains.*
- *Liveness. $\mathcal{A}$ cannot stop the checkpoints from happening.*

On Slashable Safety. Babylon claims the slashable safety. Specifically, for a PoS system with $3t + 1$ units of stake, validators with at least t units should become slashable in the view of all honest validators whenever there is a safety violation. Many PoS systems offer slashable safety against *short-range* attacks by locking validators' stakes for a period and slashing one's stake once proof of security violation is presented. However, long-range attackers can evade being slashed by publishing the attack chain after withdrawing their stakes from the canonical chain.

It has been proved in (Tas et al., 2023) that slashable safety against long-range attacks is impossible without external trust. With this result, (Tas et al., 2023) also shows that other approaches for mitigating long-range attacks, such as key-evolving signatures (Badertscher et al., 2018; Chen and Micali, 2019) cannot provide slashable safety. Nonetheless, leveraging the Bitcoin blockchain as an external trust can certainly bypass this impossibility. Assuming that checkpoints for the canonical PoS chain have been properly posted on the Bitcoin blockchain, the attacker cannot present an attack chain that diverges from the canonical chain before the latest checkpoint. In this case, the attacker must not have withdrawn its stakes and thus is slashable.

In light of the above, both ours/Pikachu and Babylon can guarantee slashable safety once the checkpoints have been properly created. Now, we turn to examine the case in which checkpoints may not

be generated correctly. The adversary has the following options: (1) not make a checkpoint; (2) make a checkpoint for an ill-formed block; (3) make more than one checkpoint for different well-formed blocks at the same height and hide the block whose checkpoint appears earlier; (4) make a checkpoint for a well-formed block but exclude some valid transactions (for censorship). Babylon introduces an *emergency break* to prevent from (2) and (3). The client can notice these attacks happening and then no longer process this chain. In case the adversary refuses to participate in the checkpoint creation, Babylon considered the punishment of inactivity, which enables the removal of the inactive validators. Regarding censorship resistance (4), Babylon proposed a roll-up technique that is orthogonal to the checkpointing mechanism.

In our system, as all checkpoints are in the chain of transactions, the adversary cannot mount the attack of (3). For (2) and (4), we can follow the exact same approach as Babylon does. For the attack of (1), it may be hard to identify who makes the DKG/threshold signing fail. Instead, we require a checkpoint to be made by a certain height of the Bitcoin blockchain, and then the client can switch to *emergency* break when it does not find a valid checkpoint by the designated position. In summary, our checkpointing mechanism provides slashable safety as long as honest clients do not switch to emergency breaks.

APPENDIX B

Supplementary Materials for Circular Dragon DKG

B.1 Supplementary Material to Coin Flipping from Our New Recursive DKG for Group-Element Secrets (§4.6)

B.1.1 Aggregatable PVSS

Recall the syntax of APVSS in Section 2.2. We present an aggregatable PVSS scheme in this section, identical to the scheme in (Gurkan et al., 2021) with a slightly different syntax solely for notational convenience. In particular, the two schemes have a similar transcript, where ours is a superset of theirs, and includes CIDs and commitments $\{A_0^{(j)}\}$ for all CIDs in the transcript. In terms of functionalities, the following algorithm in ours $\mathsf{Init}, \mathsf{Deal}, \mathsf{Agg}, \mathsf{PubVrfy}, \mathsf{Dec}, \mathsf{Rec}$ are equivalent to theirs $\mathsf{Keys}, \mathsf{Dist}, \mathsf{Agg}, \mathsf{Ver}, \mathsf{Dec}, \mathsf{Rec}$ respectively. For the getCID and $\mathsf{PubDriv}$ algorithms, they are helper functions that extract information from a transcript directly and compute.

Building blocks. We first introduce a few needed cryptographic building blocks.

Pairing. There is an efficient deterministic algorithm $\mathsf{GroupGen}$ which outputs the description of the pairing groups, including $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$, where $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are groups of order p, g is the generator of $\mathbb{G}_1$, h is the generator of $\mathbb{G}_2$, and $e : \mathbb{G}_1 \times \mathbb{G}_2$ is a bilinear map.

Signature of Knowledge. A signature of knowledge (SoK) scheme SoK for an NP language L in the random oracle model consists of two algorithms. $\mathsf{Sign}(x, w, m)$ on inputs a statement x, a witness w, and a message m, produces a signature σ on m. $\mathsf{Vrfy}(x, m, \sigma)$ verifies the signature. A SoK scheme should satisfy the Sim-Ext security (Chase and Lysyanskaya, 2006). The SoK in the APVSS scheme can be instantiated by using Schnorr signature.

- $\mathsf{Init}(1^\lambda, n)$. (1) $\mathsf{GroupGen} \to (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$; Define $\mathtt{crs} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$. (2) For uncorrupted $i \in [n]$, $dk_i \leftarrow\$ \mathbb{Z}_p$, and $ek_i = h^{dk_i}$;
- $\mathsf{Deal}((ek_i)_{i\in[n]}, \mathtt{cid})$. Sample $(a_0, a_1, \ldots, a_t) \leftarrow\$ \mathbb{Z}_p^{t+1}$, define $f(X) = \sum_{i=0}^{t} a_i X^i$, $(A_i = g^{f(i)})_{i\in[0,n]}$, and $(\hat{Y}_i = ek_i^{f(i)})_{i\in[1,n]}$.

  Sign $\mathtt{cid}$ with the knowledge of $f(0)$ w.r.t. A_0: $\mathsf{SoK.Sign}(A_0, f(0), \mathtt{cid}) \to \varsigma$. Let $sk = h^{f(0)}$ and $\mathtt{Trans} = ((A_i)_{i\in[n]}, (\hat{Y}_i)_{i\in[n]}, \{A_0\}, \{\sigma\}, \{\mathtt{cid}\})$ and return $(\mathtt{Trans}, sk)$.
- $\mathsf{Agg}(\{\mathtt{Trans}_j\}_{j\in[m]}, (ek_i)_{i\in[n]})$. Parse $\mathtt{Trans}_j = ((A_i^{(j)})_{i\in[n]}, (\hat{Y}_i^{(j)})_{i\in[n]}, \mathsf{A}_0^{(j)}$ $\gamma^{(j)}, \mathsf{CIDs}^{(j)})$. Compute $(A_i = \prod_{j\in[m]} A_i^{(j)})_{i\in[n]}$, $(\hat{Y}_i = \prod_{j\in[m]} \hat{Y}_i^{(j)})_{i\in[n]}$, $\mathsf{A}_0 = \cup_{j\in[m]} \mathsf{A}_0^{(j)}$, $\gamma = \cup_{j\in[m]} \gamma^{(j)}$, and $\mathsf{CIDs} = \cup_{j\in[m]} \mathsf{CIDs}^{(j)}$.

  Return $\mathtt{Trans} = ((A_i)_{i\in[n]}, (\hat{Y}_i)_{i\in[n]}, \mathsf{A}_0, \gamma, \mathsf{CIDs})$.
- $\mathsf{PubVrfy}((ek_i)_{i\in[n]}, \mathtt{Trans})$. Parse $\mathtt{Trans} = ((A_i)_{i\in[n]}, (\hat{Y}_i)_{i\in[n]}, \{A_0^{(j)}\}_{j\in[m]}$ $\{\varsigma^{(j)}\}_{j\in[m]}, \{\mathtt{cid}^{(j)}\}_{j\in[m]})$. It first checks $\mathsf{SoK.Vrfy}(A_0^{(j)}, \mathtt{cid}^{(j)}, \varsigma^{(j)})$ for $j \in [m]$. Next, it randomly samples a $(n-t)$-degree polynomial $q(x) \in \mathbb{Z}_p[x]$, and computes the dual code

$$(\mathsf{code}_0^\perp, \ldots, \mathsf{code}_n^\perp), \text{ where } \mathsf{code}_i^\perp = \frac{q(i)}{\prod_{j=0, j\neq i}^{n}(i-j)}.$$

  The dual code is used to check the validity of Scrape's polynomial commitment. See (Cascudo and David, 2017). In our case, it computes $A_0 = \prod_{j\in[m]} A_0^{(j)}$ and checks whether

$$\prod_{\tau=0}^{n} A_\tau^{\mathsf{code}_\tau^\perp} = 1. \tag{B.1}$$

  If the above check passes, it confirms that the exponents of $(A_i)_{i\in[0,n]}$ are from a t-degree polynomial $f(X)$, and $A_i = g^{f(i)}$. Then, it checks if $e(g, \hat{Y}_i) = e(A_i, ek_i)$ for $i \in [n]$. It returns 1 if all checks pass; otherwise, it returns 0.
- $\mathsf{getCID}(\mathtt{Trans})$. It returns $\mathtt{Trans}[4] = \{\mathtt{cid}^{(j)}\}_{j\in[m]}$.
- $\mathsf{PubDriv}(\mathtt{Trans})$. It returns $pk = \prod_{j\in[m]} A_0^{(j)}$ and $(pk_i = A_i)_{i\in[n]}$.
- $\mathsf{Dec}(ek_i, dk_i, \mathtt{Trans})$. It returns $sk_i = \hat{Y}_i^{\frac{1}{dk_i}}$.
- $\mathsf{Rec}(\{(i, sk_i)\}_{i\in\mathbb{I}})$. It first computes the Lagrange coefficients $\{\lambda_i\}_{i\in\mathbb{I}}$ based on $\mathbb{I}$, and then $sk = \prod sk_i^{\lambda_i}$.

B.1.2 Bacho et al.'s One-round Coin Flipping (Bacho et al., 2024)

For completeness, we include a description of Bacho et al.' coin flipping (Bacho et al., 2024) in the following. Here we only specify the key structure required by the scheme, without giving all details of

the setup and distributed key generation parts, as they will be fulfilled by our new group-element DKG protocol in §4.6.

Building Block. The scheme requires pairing friendly groups $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$, which are the same groups used by the APVSS scheme. It also uses a NIZK proof system, dubbed Dleq, for showing two group elements have the same discrete logarithm w.r.t. two different bases. Formally, Dleq works for the following relation:

$$R_{eq} := \{(g_1, g', g_2, g'') \in \mathbb{G}^4; \alpha : g' = g_1^{\alpha} \wedge g'' = g_2^{\alpha}\}.$$

Note that Dleq has an efficient instantiation in the random oracle model (Chaum and Pedersen, 1992). In addition, it assumes a random oracle $H_1 : \{0,1\}^* \to \mathbb{G}_1$ and a random oracle $H_2 : \mathbb{G}_T \to V$.

Description. The description of the protocol is available in Fig.B.1. It is easy to see the communication complexity for each coin is $O(\lambda n^2)$.

Setup. The setup phase consists of a DKG setup and a one-round *commitment* setup.

- *DKG:* Each participant P_i obtains $(vk, vk_1, \ldots, vk_n) \in \mathbb{G}_1^{n+1}$ and its secret key $sk_i \in \mathbb{G}_2$. The output satisfies that $e(pk_i, h) = e(g, sk_i)$, and there exists a t-degree polynomial f, s.t. $pk_j = g^{f(j)}$ for $j \in [n]$ and $pk = g^{f(0)}$.
- $\mathsf{ComSetup}$: In addition to the DKG setup, each honest participant P_i performs the following tasks in one round:
  - Initialize a local set $\mathcal{G} = \emptyset$.
  - Sample $\alpha_i \leftarrow\!\!\$\, \mathbb{Z}_p^*$, and multicast $\mathsf{cm}_i := (g^{\alpha_i}, h^{-\alpha_i} sk_i)$. Store α_i.
  - Upon receiving $\mathsf{cm}_j = (\mathsf{cm}_{j,1}, \mathsf{cm}_{j,2})$ from P_j, check if $e(pk_j, h) = e(\mathsf{cm}_{j,1}, h) \cdot e(g, \mathsf{cm}_{j,2})$. Only if the check verifies, update $\mathcal{G} \leftarrow \mathcal{G} \cup \{P_j\}$. Store all $\{\mathsf{cm}_j\}_{j \in \mathcal{G}}$.

$\mathsf{Coin}^V[\mathsf{sid}]$**.** Each honest participant P_i performs the following tasks to obtain a the coin value for the instance with sid:

- Compute $\sigma_i \leftarrow (H_1(\mathsf{sid})^{\alpha_i}, e(H_1(\mathsf{sid}), sk_i))$ along with a proof $\pi_i = \mathsf{Dleq}(g, g^{\alpha_i}, H_1(\mathsf{sid}), H_1(\mathsf{sid})^{\alpha_i})$. Multicast (σ_i, π_i).
- Upon receiving (σ_j, π_j) from P_j, check if π_j verifies $\sigma_{j,1}$ and $\mathsf{cm}_{j,1}$. Further, check if $\sigma_{j,2} = e(H_1(\mathsf{sid}), \mathsf{cm}_{j,2}) \cdot e(\sigma_{j,1}, h)$. Upon recieving $t+1$ valid tuples $\{(\sigma_j, \pi_j)\}_{j \in S}$ from distinct nodes in $\mathcal{G}$, compute $\sigma = \prod_{j \in S} \sigma_{j,2}^{\lambda_j}$, where $\lambda_j = \prod_{k \in S} \frac{-k}{j-k}$. Return $r_{\mathsf{sid}} = H_2(\sigma) \in V$.

FIGURE B.1: The coin-flipping protocol in (Bacho et al., 2024).

The security of the coin flipping protocol is based on the co-one-more discrete logarithm (co-OMDL) assumption (Bacho and Loss, 2023b), which we recall in the following definition.

DKG[sid]. Every $P_i \in \mathcal{D}^{(k)} \subset \mathcal{P}$ performs the following steps:

Step 1: // Each consortium runs an inner-group DKG

activate an inner-group DKG protocol $\mathsf{DKG}[\mathsf{sid}|k]\langle\mathcal{D}^{(k)}\rangle$

Step 2: // Each node broadcasts a PVSS script to its consortium group

$\mathsf{PVSS.Deal}((ek_j)_{j\in[n]}, P_i) \to T_i$, which shares a random secret s_i

$\mathsf{BB}[\mathsf{sid}|k]\langle\mathcal{D}^{(k)}, P_i(T_i)\rangle$, which broadcasts T_i across the group $\mathcal{D}^{(k)}$

wait for receiving all $\mathbb{T} := \{T_j\}_{j\in\mathcal{D}^{(k)}}$ from BBs until next step starts

Step 3: // Every consortium broadcasts its aggregated PVSS to all nodes

$\mathsf{PVSS.Agg}(\mathbb{T}) \to T^{(k)}$//This step skips empty and invalid PVSS transcripts

$\mathsf{PCSBB}[\mathsf{sid}]\langle\{\mathcal{D}^{(1)}, \ldots, \mathcal{D}^{(m)}\}, \mathcal{D}^{(k)}(T^{(k)})\rangle \to \mathcal{V}$

$\mathsf{PVSS.Agg}(\mathcal{V}) \to T$, $\mathsf{PVSS.PubDrive}(T) \to pk, (pk_j)_{j\in[n]}$, $\mathsf{PVSS.Dec}(dk_i, T) \to sk_i$

output $pk, (pk_i)_{j\in[n]}, sk_i$

//**if** need to flip coins, multicast sk_i and reconstruct sk, **output** $\mathsf{Hash}(sk)$

//or use the DKG output to run the coin-flipping protocol from (Bacho et al., 2024) (Fig. B.1)

FIGURE B.2: A simple sub-linear round DKG protocol for coin flipping.

DEFINITION 9 (co-OMDL Assumption). *Let $(\mathbb{G}_1, \mathbb{G}_2, p, g, h)$ be cyclic groups of prime order p. For a positive integer $k \in \mathbb{N}$, the k-co-OMDL assumption states that, for any PPT adversary $\mathcal{A}$,*

$$\Pr\left[\begin{array}{l}(z_1, \ldots, z_k) \leftarrow\$ \mathbb{Z}_p^k, \xi_i \leftarrow (g^{z_i}, h^{z_i}) \in \mathbb{G}_1 \times \mathbb{G}_2, \forall i \in [k] \\ (z'_1, \ldots, z'_k) \leftarrow \mathcal{A}^{\mathsf{DL}_g(\cdot)}(\mathbb{G}_1, \mathbb{G}_2, p, g, h, (\xi_i)_{i\in[k]}) : \\ z'_i = z_i, \forall i \in [k]\end{array}\right] \leq \mathsf{negl}(\lambda),$$

where the oracle DL_g on input a group element X in $\mathbb{G}_1$ returns x s.t. $X = g^x$, and $\mathcal{A}$ can access DL_g at most $k-1$ times.

B.2 A Simple Group-Element DKG with Sub-linear Round

Figure B.2 illustrates our first very simple construction of group-element DKG for Bacho et al.'s coin-flipping protocol (Bacho et al., 2024) with sub-linear rounds from PCSBB and APVSS, which divides the whole network into a partition $\mathbb{D} := \left\{\mathcal{D}^{(k)}\right\}_{k\in[m]}$ with m consortium groups, to take advantage of PCSBB. Every $\mathcal{D}^{(i)}$ in the partition $\mathbb{D}$ contains the (almost) same number η of nodes, and $\mathbb{D}$ does not rely on a trusted setup, as it is some arbitrarily pre-specified parameters that can be hard-coded by the protocol in any deterministic way. In greater detail, the protocol proceeds as follows:

- **Step 1**. For each consortium group $\mathcal{D}^{(k)}$, its member nodes collectively execute an inner-group DKG protocol, which distributedly initializes a threshold cryptosystem across the nodes $\{P_1^{(k)}, \ldots P_\eta^{(k)}\}$ in the group. Note that although these inner-group DKG protocols only set up threshold cryptosystems for each consortium group, it suffices to use them to instantiate a PCSBB protocol, thus enabling each honest-majority group to broadcast messages across the whole network efficiently.
- **Step 2**. Then, each node P_i in the group $\mathcal{D}^{(k)}$ samples a random secret s_i and computes a (t, n) PVSS transcript T_i encrypting n secret shares of s_i. Though T_i is a PVSS transcript sharing s_i by encrypting s_i's n secret shares with the public keys of the whole network, P_i only attempts to send T_i to its associated consortium group $\mathcal{D}^{(k)}$. For this purpose, P_i invokes an inner-group BB protocol using T_i as input, and then waits for receiving all $\mathbb{T} := \{T_j\}_{j \in \mathcal{D}^{(k)}}$ from all BBs in its consortium group.
- **Step 3**. Subsequently, every node P_i, as a member of its consortium group $\mathcal{D}^{(k)}$, aggregates $\mathbb{T}$ into an aggregate PVSS transcript $T^{(k)}$, and executes a PCSBB protocol running among the whole network, using $T^{(k)}$ as input. After PCSBB outputs $\mathcal{V}$, P_i aggregates all valid PVSS scripts carried by $\mathcal{V}$ into T. All honest nodes would obtain the same aggregate PVSS transcript T, so they can derive all required public keys and locally decrypt their exclusive secret key shares, thus completing the DKG across the whole network.

Efficiency Analysis. Before analyzing the complexities of our construction, let us first recall the complexities of its building blocks. If using the threshold signature from MTSS with $\lambda \log n$-bit size, our PCSBB construction attains an expected communication of $O(nm\ell + \lambda mn^2 \log n)$ bits and it terminates in $O(m)$ epochs, i.e., $O(m\kappa)$ rounds (as we can safely choose each PCSBB epoch to be $O(\kappa)$ rounds, according to Lemma 2). Also, an inner-group DKG protocol costs $O(\eta)$ rounds and $O(\lambda\eta^3)$ communicated bits, and an inner-group BB can attain $O(\eta)$ rounds and $O(\ell\eta + \lambda\eta^2)$ communicated bits, even if instantiating them by classical (adaptively secure) results. Note that the input length ℓ of BB and PCSBB is λn, as it corresponds to the bit length of a PVSS transcript.

So our DKG protocol in Fig. B.2 realizes an expected communication complexity of $O(m \cdot \mathsf{Coin}_\eta + n \cdot \mathsf{BB}_\eta(\lambda n) + \mathsf{PCSBB}(\lambda n)) = m \cdot O(\lambda\eta^3) + n \cdot O(\lambda n\eta) + O(nm \cdot \lambda n + \lambda mn^2 \log n) = O(\lambda\eta n^2 + \lambda mn^2 \log n)$ bits and a round complexity of $O(\eta) + O(\eta) + O(m\kappa) = O(\eta + m\kappa)$. When $m := n^{0.5}$ and $\eta := n^{0.5}$, the communication becomes expected $O(\lambda n^{2.5} \log n)$, and more importantly, its round complexity is $O(\kappa n^{0.5})$ which is sub-linear.

Security Analysis. The security of our simple sub-linear round DKG protocol described in Fig. B.2 can be summarized by the following Lemma.

LEMMA 24. *If instantiating our sub-linear DKG protocol in Fig.B.2 using the APVSS scheme from (Bacho and Loss, 2023b), it can securely realize the key generation part of the adaptively-secure unique threshold signature scheme from (Bacho et al., 2024), in the presence of a PPT adversary that can adaptively corrupt up to $n/2$ nodes.*

SKETCH. Despite the adversary, in the partition $\mathbb{D} := \left\{\mathcal{D}^{(k)}\right\}_{k\in[m]}$, there must exist some consortium group $\mathcal{D}^{(h)} \in \mathbb{D}$, such that $\mathcal{D}^{(h)}$ has honest majority, otherwise, the basic honest-majority assumption across the whole network is violated. That said, we can state: (i) all honest nodes in $\mathcal{D}^{(h)}$ must receive from the inner-group BBs the same set $\mathbb{T}$ of PVSS transcripts including those from all honest nodes in $\mathcal{D}^{(h)}$, otherwise, the security of inner-group BB is breached; (ii) PCSBB must deliver the same list $\mathcal{V}$ of results to the whole network, and more importantly, the list $\mathcal{V}$ includes a PVSS transcript that is aggregated from $\mathbb{T}$, otherwise, PCSBB is breached. Thus, all honest nodes must aggregate $\mathcal{V}$ to obtain the same PVSS transcript T, which at least aggregates the PVSS transcripts generated by all honest nodes in $\mathcal{D}^{(h)}$. Once we prove the above "consensus" part and demonstrate that all honest nodes must obtain an aggregate PVSS transcript (which aggregates at least some honest node's PVSS transcript) despite $n/2$ adaptive corruptions, the remaining analysis would largely mirror that of the linear-round DKG in (Bacho et al., 2024) and repeat our previous analysis in Section 4.6.3. We refrain from repeating such existing analysis and refer interested readers to the earlier section for details.

□

B.3 Supplementary Material to Field-Element DKG (§4.7)

B.3.1 NIZK Construction

We introduce a NIZK construction in the random oracle model for the following relation.

$$\{(\mathsf{cm}, pk, g, h) \in \mathbb{G}^4; (sk, r) \in \mathbb{F}^2 : \mathsf{cm} = g^{sk}h^r \wedge pk = g^{sk}\} \tag{B.2}$$

We prove the relation by proving the knowledge of sk w.r.t. pk and then r w.r.t. cm/pk. In the detail the protocol works as follows.

- $\mathsf{NIZK.Prove}((\mathsf{cm}, pk, g, h); (sk, r))$. First, sample $a \leftarrow\!\$\ \mathbb{F}$ and $b \leftarrow\!\$\ \mathbb{F}$, and compute $A = g^a$, $B = h^b$. Then, compute $c = \mathsf{Hash}((\mathsf{cm}, pk, g, h), (A, B)) \in \mathbb{F}$, where Hash is a hash function modeled as a random oracle. Next, compute $z_1 = a + c \cdot sk$ and $z_2 = b + c\dot{r}$. The proof π is (z_1, z_2, c).
- $\mathsf{NIZK.Vrfy}((\mathsf{cm}, pk, g, h), \pi)$. Compute $H = \mathsf{cm}/pk$, and then check if

  $$\mathsf{Hash}((\mathsf{cm}, pk, g, h), (g^{z_1}/pk^c, h^{z_2}/H^c)).$$

  If passing the check, return 1. Otherwise, return 0.

It is easy to check the underlying Σ protocol satisfies honest-verifier zero-knowledge, 2-special soundness, and completeness. Then, the zero-knowledge, simulation knowledge soundness, and completeness trivially follow the security results about the Fiat-Shamir transformation(Fiat and Shamir, 1986; Bellare and Rogaway, 1993).