DESIGN AND ANALYSIS OF COMPACT, PAIRING-FREE DISTRIBUTED VERIFIABLE RANDOM FUNCTIONS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ARDA BUĞRA ÖZER

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
CRYPTOGRAPHY

JULY 2026

Approval of the thesis:

DESIGN AND ANALYSIS OF COMPACT, PAIRING-FREE DISTRIBUTED VERIFIABLE RANDOM FUNCTIONS

submitted by **ARDA BUĞRA ÖZER** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Cryptography Department, Middle East Technical University** by,

Prof. Dr. A. Sevtap Kestel
Dean, Graduate School of **Applied Mathematics** _____

Assoc. Prof. Dr. Oğuz Yayla
Head of Department, **Cryptography** _____

Assoc. Prof. Dr. Oğuz Yayla
Supervisor, **Cryptography, METU** _____

Prof. Dr. Zülfükar Saygı
Co-supervisor, **Mathematics, TOBB ETU** _____

Examining Committee Members:

Prof. Dr. Ali Aydın Selçuk
Computer Engineering, TOBB ETU _____

Assoc. Prof. Dr. Oğuz Yayla
Cryptography, METU _____

Assoc. Prof. Dr. Ayşe Nurdan Saran
Computer Engineering, Çankaya University _____

Assist. Prof. Dr. Buket Özkaya
Cryptography, METU _____

Assist. Prof. Dr. Abdullah Talayhan
Computer Engineering, Bilkent University _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ARDA BUĞRA ÖZER

Signature :

ABSTRACT

DESIGN AND ANALYSIS OF COMPACT, PAIRING-FREE
DISTRIBUTED VERIFIABLE RANDOM FUNCTIONS

Özer, Arda Buğra
Ph.D., Department of Cryptography
Supervisor : Assoc. Prof. Dr. Oğuz Yayla
Co-Supervisor : Prof. Dr. Zülfükar Saygı

July 2026, 110 pages

Verifiable Random Functions (VRFs) are cryptographic primitives that generate unpredictable randomness together with a publicly verifiable proof of correct generation following the protocol, a critical requirement for decentralized applications in blockchain infrastructure, decentralized finance, and online gaming. While distributed VRFs (DVRFs) eliminate dependence on a single trusted authority, existing constructions face a fundamental dilemma: linear proof sizes in the threshold parameter (DDH-DVRF) or reliance on computationally expensive bilinear pairings (GLOW-DVRF, FlexiRand). This thesis resolves both facets of this dilemma with three contributions despite the downside of adding another interaction round amongst the parties involved in the generation.

We first introduce DVRFwCP, a distributed VRF with constant-size, pairing-free proofs achieved by layering a threshold structure over a Chaum-Pedersen NIZK system, where we use an augmented secure distributed key generation to produce the required nonce. We then introduce Icy-DVRF, which eliminates the quadratic interaction bottleneck of DVRFwCP by integrating the FROST-style preprocessed nonces, reducing total per-evaluation communication from $O(n^2t)$ to $O(t)$ while preserving constant-size proofs. Finally, we introduce IcyVeil, the first pairing-free output-private DVRF, which extends Icy-DVRF with a Schnorr-based blinding mechanism that conceals the VRF output until revealed by the user. We evaluate all three constructions theoretically and empirically via Solidity

smart-contract implementation on the Ethereum Sepolia testnet. The measurements show a 43.02% reduction in on-chain verification gas cost compared to GLOW-DVRF.

Keywords: Cryptography, Distributed Verifiable Random Function, Output Privacy, Blockchain, Smart Contracts

ÖZ

KOMPAKT VE EŞLEŞMESİZ DAĞITIK DOĞRULANABİLİR RASTGELE FONKSİYONLARIN TASARIMI VE ANALİZİ

Özer, Arda Buğra
Doktora, Kriptografi Bölümü
Tez Yöneticisi : Doç. Dr. Oğuz Yayla
Ortak Tez Yöneticisi : Prof. Dr. Zülfükar Saygı

Temmuz 2026, 110 sayfa

Doğrulanabilir Rastgele Fonksiyonlar (VRF'ler), blokzinciri altyapısındaki, merkeziyetsiz finanstaki ve çevrimiçi oyunlardaki merkeziyetsiz uygulamalar için kritik bir gereksinim olan, protokole uygun doğru üretimin herkese açık olarak doğrulanabilir bir kanıtı ile birlikte öngörülemeyen rastgelelik üreten kriptografik ilkellerdir. Dağıtık VRF'ler (DVRF'ler) tek bir güvenilir otoriteye olan bağımlılığı ortadan kaldırırken, mevcut yapılar temel bir ikilemle karşı karşıyadır: eşik parametresinde doğrusal kanıt boyutları (DDH-DVRF), veya hesaplama açısından maliyetli bilineer eşleştirmelere dayanma (GLOW-DVRF, FlexiRand). Bu tez, üretime katılan taraflar arasına fazladan bir etkileşim turu ekleme dezavantajına rağmen, sunduğu üç katkı ile bu ikilemin her iki yönünü de çözmektedir.

İlk olarak, gerekli tek seferlik rastgele değer üretmek için bir zenginleştirilmiş güvenli dağıtık anahtar üretimi kullandığımız, bir Chaum-Pedersen etkileşimsiz sıfır bilgi sistemi üzerine bir eşik yapısı katarak elde edilen, sabit boyutlu ve eşleştirmesiz kanıtlara sahip dağıtık bir VRF olan DVRFwCP'yi sunuyoruz. Daha sonra, FROST tarzı önceden işlenmiş tek seferlik rastgele değerler entegre ederek DVRFwCP'nin karesel etkileşim darboğazını ortadan kaldıran, kanıtlardaki kompakt sabit boyutu korurken değerlendirme başına toplam iletişimi $O(n^2t)$ seviyesinden $O(t)$ seviyesine düşüren Icy-DVRF'yi sunuyoruz. Son olarak, kullanıcı tarafından açıklanana kadar VRF çıktısını gizleyen Schnorr tabanlı bir körleme mekanizmasıyla Icy-DVRF'yi genişleten, eşleştirmesiz ve çıktı gizliliğine sahip ilk DVRF

olan IcyVeil'i sunuyoruz. Her üç yapıyı da teorik olarak ve Ethereum Sepolia test ağındaki Solidity akıllı sözleşme uygulaması aracılığıyla deneysel olarak değerlendiriyor, GLOW-DVRF'ye kıyasla zincir üstü doğrulama gas maliyetinde %43,02'lik bir düşüş olduğunu gösteriyoruz.

Anahtar Kelimeler: Kriptografi, Dağıtık Doğrulanabilir Rastgele Fonksiyon, Çıktı Gizliliği, Blokzincir, Akıllı Sözleşme

“This is a ballad dat I wrote. It’s all, it’s about peace and love and good happiness stuff.”
Blue Powder - Steve Vai

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my supervisor, Assoc. Prof. Dr. Oğuz Yayla, who guided and directed me throughout my studies.

I am indebted to my co-supervisor, Prof. Dr. Zülfükar Saygı, without whom none of this would have started. With his consistent guidance and support, I found my way in this field.

My co-author Dr. Ahmet Ramazan Ağırtaş deserves more than words can express. He gave his time and effort without ever being asked twice. The work in this thesis is as much a product of his generosity as it is of mine.

Special thanks go to Nejla Erdoğdu and the METU Institute of Applied Mathematics family for all the help and for keeping the wonderful atmosphere.

I thank our extended family and friends, especially those to whom we had to say early goodbyes. They may have wandered from our sight, but never from our hearts and memories.

Gülnihal Bozkurt, my mother-in-law, has supported me and welcomed me as her own, and I could not be more grateful.

My brother Derya, his wife Sevilay, and their bundles of joy Eda Sıla and Umut Bora have filled every step of this journey with warmth.

I am blessed to have had my parents, Şeniz and Mehmet Naili, by my side through everything. They have been there from the very beginning and I know they will be there 'till the end, providing me with everything I needed before I even knew to ask. My journey, with every twist and turn, has blossomed from their endless support and love.

And to İpek, my better half, in whose soul I found that the greatest thing I'll ever learn is just to love and be loved in return.

TABLE OF CONTENTS

ABSTRACT vii

ÖZ ix

ACKNOWLEDGMENTS xiii

TABLE OF CONTENTS xv

LIST OF TABLES xxi

LIST OF FIGURES xxiii

LIST OF ABBREVIATIONS xxv

1 INTRODUCTION 1

1.1 Motivation 1

1.2 The Trilemma 2

1.3 Contributions 2

1.4 Publications 3

1.5 Organization of the Thesis 3

CHAPTERS

2 BACKGROUND AND PRELIMINARIES 5

2.1 A Survey of the VRF Landscape 5

2.2 Notation and Group Setup 6
2.3 Hardness Assumptions 7
2.4 Lagrangian Interpolation 8
2.5 Verifiable Secret Sharing 8
2.6 Distributed Key Generation 10
2.7 Non-Interactive Zero-Knowledge Proof Systems 12
2.7.1 Σ-Protocols and Non-Interactive Zero-Knowledge 12
2.7.2 Schnorr Proof of Knowledge of Exponent 14
2.7.3 Chaum-Pedersen DLEQ Proof 14
2.8 FROST: Flexible Round-Optimized Schnorr Threshold Signatures . 15
2.9 Two Prior Baseline DVRF Constructions 16
2.9.1 DDH-DVRF 16
2.9.2 GLOW-DVRF 17
3 DVRFWCP: COMPACT PROOFS WITHOUT PAIRINGS 19
3.1 The Two-Sided Problem 19
3.2 The Augmented Secure-DKG 20
3.3 Protocol Definition 21
3.4 Protocol Execution 23
3.5 Security 25
3.6 Comparison 25
4 ICY-DVRF: ELIMINATING THE INTERACTION BOTTLENECK 29
4.1 FROST as the Key Insight 29

4.2 Protocol Definition and Setup 30
4.2.1 Domain Separation of Hash Functions 31
4.2.2 Notation and Distributed Key Generation 32
4.3 The Preprocess and Binding Mechanisms 33
4.4 Protocol Execution 34
4.5 Formal Security Analysis 36
4.5.1 Correctness and Consistency 37
4.5.2 Uniqueness and Pseudorandomness 37
4.5.3 Knowledge-Soundness of T-DLEQ-Int 39
4.5.4 Knowledge-Soundness of T-DLEQ 45
4.5.4.1 Overview of Proof 45
4.5.4.2 Setting, Assumptions, and the Bi-DLEQ Assumption . . 46
4.5.4.3 Knowledge-Soundness, Single-Honest-Party 51
4.5.5 Security Tightness and Parameter Recommendations 60
4.6 Comparison 61
4.6.1 Mathematical Comparison 62
4.6.2 Communication Cost 62
4.6.3 Theoretical Gas Cost 64
5 ICYVEIL: OUTPUT PRIVACY VIA PAIRING-FREE BLINDING 65
5.1 Motivation and Positioning 65
5.2 The Blinding Mechanism 66
5.3 The Smart Contract Framework 67

5.4 Protocol Definition 69

5.5 Security 72

5.5.1 The Output-Privacy Game 73

5.5.2 One-More Unforgeability 74

5.5.3 Proof Sketch of Output-Privacy 76

5.5.4 Proof Sketch of One-More Unforgeability 77

5.6 Comparison 77

6 EVALUATION AND COMPARISON 81

6.1 Theoretical Analysis 81

6.1.1 DVRFwCP and Icy-DVRF Verification Cost 81

6.1.2 Icy-DVRF Communication Improvement over DVRFwCP . . . 82

6.1.3 IcyVeil Overhead Extrapolation 82

6.1.4 Scalability Under Committee Growth 83

6.2 Empirical Results 83

6.2.1 On-Chain Evaluation and Benchmarks 83

Verification gas costs. 84

Calldata and precompile analysis. 85

Storage and state management. 85

6.2.2 Off-Chain Evaluation and Benchmarks 86

Runtime comparison. 86

Communication cost at concrete committee sizes. 87

Setup cost and preprocess amortization. 87

Per-phase breakdown. 88

Measurement library. 88

Throughput under slot budgets. 89

Batching. 90

6.3 Summary 91

7 ROBUSTNESS, ACCOUNTABILITY, AND THE RANDOMNESS-BEACON LANDSCAPE 93

7.1 The Robustness/Efficiency Tradeoff 93

7.2 Cryptographic Identifiability and Economic Disincentives 94

7.3 Resilience against Malicious Combiners 94

7.4 Consensus on Node Exclusion 95

7.5 Operational Fault-Tolerance Model 95

7.6 Positioning within the Randomness-Beacon Landscape 96

7.7 Conclusion 97

8 CONCLUSION AND FUTURE WORK 99

8.1 Summary of Contributions 99

8.2 Directions for Future Work 100

REFERENCES 103

CURRICULUM VITAE 109

LIST OF TABLES

Table 3.1 Comparison of Operations of DDH-DVRF, GLOW-DVRF and DVRFwCP. 26

Table 3.2 Gas Cost Estimation Comparison for the BLS12-381 Curve. 27

Table 4.1 Mathematical Operations of DVRFwCP and Icy-DVRF Schemes. 63

Table 4.2 Per Participant Communication Cost of DVRFwCP and Icy-DVRF. 63

Table 5.1 Theoretical Gas Cost Estimations of Verification Costs on BLS12-381. . . . 78

Table 6.1 Consolidated Theoretical Gas Estimates for On-Chain Verify on BLS12-381. 82

Table 6.2 Measured Full Verification Gas on the Sepolia Testnet. 84

Table 6.3 Calldata Comparison for Icy-DVRF and GLOW-DVRF Verify. 85

Table 6.4 Comparison of Precompile-Only Costs Excluding Verify Framing Overhead. 86

Table 6.5 Off-Chain Per-Request Cost: DVRFwCP vs. Icy-DVRF. 87

Table 6.6 Per-Node Post-Setup Broadcast Instantiating the Formulas of Table 4.2. . . 87

Table 6.7 One-Time Setup Cost for DVRFwCP and Icy-DVRF. 88

Table 6.8 Measured Per-Phase Computation Time (ms) at $(n, t) = (10, 7)$. 89

Table 6.9 Off-Chain Throughput Under Ethereum Slot Budgets. 90

Table 6.10 Batched Throughput (requests/s) as a Function of Batch Size B. 91

LIST OF FIGURES

Figure 5.1 Message Flow for IcyVeil via Smart Contract. 68

LIST OF ABBREVIATIONS

AGM	Algebraic Group Model
BLS	Barreto-Lynn-Scott (curve) / Boneh-Lynn-Shacham (signatures)
CDH	Computational Diffie-Hellman
DDH	Decisional Diffie-Hellman
DKG	Distributed Key Generation
DL	Discrete Logarithm
DLEQ	Discrete Logarithm Equality
DRB	Distributed Randomness Beacon
DVRF	Distributed Verifiable Random Function
EC	Elliptic Curve
EIP	Ethereum Improvement Proposal
FROST	Flexible Round-Optimized Schnorr Threshold Signatures
NIZK	Non-Interactive Zero-Knowledge
OMDL	One-More Discrete Logarithm
OM-CDH	One-More Computational Diffie-Hellman
PKI	Public Key Infrastructure
PoS	Proof of Stake
VSS	Verifiable Secret Sharing
VRF	Verifiable Random Function

CHAPTER 1

INTRODUCTION

1.1 Motivation

Generating randomness that is unpredictable, unbiased, and publicly verifiable is a foundational requirement of modern decentralized systems. Blockchain consensus protocols rely on randomness to select block proposers and validation committees, as in Algorand [35], Cardano [23], and Dfinity [16]; decentralized finance and on-chain gaming distribute rewards and outcomes accordingly. More broadly, these Web3 applications all depend on a source of unpredictability that no single participant can foresee, influence, or manipulate. The same guarantee extends well beyond the blockchain setting: the DNS security extension NSEC5 uses a verifiable random function to authenticate denial-of-existence responses without exposing zone contents [45]. Verifiable randomness is equally important in machine learning, where it is central to training regularization, generative model sampling, and fair client selection in federated learning [54]. In each case, when the randomness is not itself verifiable, no external party can confirm that the claimed procedure was the one actually followed, leaving room for hidden bias or manipulation. In a system of mutually distrusting parties, the randomness must not only be unpredictable and free of manipulation but also publicly verifiable: any party must be able to check and confirm that it was generated honestly.

The cryptographic primitive that provides this guarantee is a Verifiable Random Function (VRF), introduced by Micali, Rabin, and Vadhan [44]. A single-party VRF, however, merely transfers the trust problem: the holder of the secret key can predict every output in advance. Distributed Verifiable Random Functions (DVRFs) address this by sharing the secret key among a committee of n parties using threshold cryptography, so that any t of them acting as a union can produce a verifiable output, while no coalition of fewer than t parties can predict, bias, or withhold it. The focus of this thesis is making such a construction efficient enough for on-chain verification, so that the gas overhead charged by the network is reduced.

1.2 The Trilemma

The two base DVRF constructions that preceded this work, both due to Galindo et al. [33], are non-interactive after the distributed key generation phase, but are positioned on two different sides of a spectrum: linear-size proofs and linear-size verification cost, or constant proof size but using bilinear pairings in verification.

The two foundational DVRF constructions by Galindo et al. [33] present a trade-off: linear proof sizes and verification costs versus constant proof sizes that rely on bilinear pairings. We introduce a third option to this paradigm by achieving constant-size, pairing-free proofs at the expense of an additional interaction round. We refer to these dire straits as "The Trilemma of DVRFs":

- **Linear proof size.** DDH-DVRF is free of bilinear pairings, but its proof consists of t group elements, one per participating party, hence both proof size and verification cost grow linearly in the threshold t, becoming unviable as the committee size increases.
- **Bilinear pairing cost.** GLOW-DVRF compresses the proof to a single group element using bilinear pairings, but its verification carries a large fixed cost dominated by the pairing operations.
- **Interaction overhead.** DVRFwCP and Icy-DVRF constructions presented in this thesis, achieve a constant-size, pairing-free proof, but requires a second interactive round among the proving parties incurring overhead costs that are off-chain.

A fourth limitation orthogonal to these three dimensions, affecting all of the previous options, is that the VRF output becomes public as soon as the proof is verified. For applications such as on-chain gaming, the requester must be able to learn and reuse the randomness before the rest of the network observes it; the smart-contract ecosystem provides no mechanism to guarantee this privacy on-chain. Output privacy had previously been achieved by FlexiRand [40], but built on top of the pairing-based GLOW-DVRF, it inherits the high cost of verification via pairings.

1.3 Contributions

In this thesis we develop three distributed verifiable random functions that have compact proofs via pairing-free operations, and evaluate them both theoretically and empirically.

1. **DVRFwCP** [2] (Chapter 3) addresses the linear-proof and pairing facets simultaneously. It integrates a t-out-of-n threshold structure over a Chaum-Pedersen DLEQ proof, using an Augmented Secure-DKG (Section 3.2) that shares the ephemeral commitment in two bases. The result is a constant-size proof of

one group element and two scalars, verified in constant time and without pairings. The shortcoming of this construction is the two round interaction of Augmented Secure-DKG with a communication cost of $O(n^2t)$ group elements.

2. **Icy-DVRF** [3] (Chapter 4) removes the interaction overhead introduced by DVRFwCP. It replaces the Augmented Secure-DKG with the preprocessed, binding nonce logic of FROST (Section 2.8), reducing total per-evaluation communication to $O(t)$ and decreasing the interaction rounds to one, while preserving the compact pairing-free proof. Its knowledge-soundness is reduced to the one-more discrete logarithm problem through a self-contained Generalized Forking Lemma argument.

3. **IcyVeil** [4] (Chapter 5) adds output privacy to Icy-DVRF by adapting the blinding mechanism of FlexiRand using only discrete-logarithm machinery. It allows a requester to blind the input, have the committee evaluate the blinded value, and unblind the result locally, so that an on-chain observer is left with a Computational Diffie-Hellman instance. To the best of our knowledge, it is the first pairing-free output-private DVRF.

4. **Empirical validation** (Chapter 6). A Solidity smart-contract implementation deployed on the Ethereum Sepolia testnet confirms a 43.02% reduction in on-chain verification gas cost relative to GLOW-DVRF, a saving attributable to the elimination of a single bilinear pairing.

1.4 Publications

The three constructions presented in this thesis are based on the following publications, co-authored with Ahmet Ramazan Ağırtaş, Zülfükar Saygı, and Oğuz Yayla.

- A. R. Ağırtaş, A. B. Özer, Z. Saygı, and O. Yayla. *Distributed Verifiable Random Function with Compact Proof*. In Cyber Security, Cryptology, and Machine Learning (CSCML 2025), Springer, 2025, pp. 119–134. DOI: 10.1007/978-3-031-76934-4_8

- A. R. Ağırtaş, A. B. Özer, Z. Saygı, and O. Yayla. *Icy-DVRF: A Distributed Verifiable Random Function Based on FROST Signatures*. IEEE Access, vol. 14, 2026, pp. 105882–105909. DOI: 10.1109/ACCESS.2026.3712805

- A. R. Ağırtaş, A. B. Özer, Z. Saygı, and O. Yayla. *IcyVeil: An Output-Private DVRF Based on Icy-DVRF*. To appear in BCAA 2026. https://eprint.iacr.org/2026/1138

1.5 Organization of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 surveys the verifiable random function literature and presents the cryptographic preliminaries common to all three

constructions: the hardness assumptions, secret sharing and distributed key generation, the Schnorr and Chaum-Pedersen proof systems, FROST, and the prior DVRF schemes used as baselines. Chapters 3, 4, and 5 present DVRFwCP, Icy-DVRF, and IcyVeil respectively, together with their security analyses. Chapter 6 consolidates the theoretical cost estimates and reports the empirical measurements obtained on the Sepolia testnet. Chapter 7 examines the robustness/efficiency tradeoff inherent in the abort-and-identify model, together with the accountability mechanisms that make it viable in a real world deployment. Chapter 8 concludes the thesis and identifies directions for future work.

CHAPTER 2

BACKGROUND AND PRELIMINARIES

2.1 A Survey of the VRF Landscape

True randomness is a crucial element of computing, and its applications range from generating cryptographic keys to digital signing, online gaming, and gambling. Recently, with the introduction and rapid growth of blockchain technologies and Web3-based applications, especially in decentralized finance and the online gaming industry, the demand for trustworthy sources of randomization has increased substantially. In most of these applications involving multiple participants, ensuring that the randomness used is neither predictable nor biased toward any one party is essential to the integrity of the application. Many platforms that rely on blockchain technology have consensus protocols that involve authorizing the creation of blocks to a block producer, whose selection mechanism frequently involves a means for collective sampling of random values. A typical solution to avoid dependence on a trusted party is to utilize a process that permits the distributed generation of verifiable randomness.

Micali, Rabin, and Vadhan [44] introduced verifiable random functions (VRF) that offer such capabilities. A VRF may be viewed as the public-key equivalent of a keyed cryptographic hash. For a secret key chosen uniformly at random, a VRF, on the input of a plaintext α, outputs γ and a proof π, which can publicly be used to verify the correct evaluation of γ. They realized that VRFs and unique signatures share similarities and constructed an RSA signature-based VRF. Dodis and Yampolskiy [26] constructed a more optimal VRF using bilinear pairings and collision-resistant hash functions. A more efficient construction using elliptic curves was introduced by Papadopoulos et al. [45]. This was later brought into IETF standardization by Goldberg et al. [36]. In [15], Buser et al. considered post-quantum VRFs based on symmetric primitives. Later, Esgin et al. [29] studied few-time verifiable Lattice-Based post-quantum secure VRFs, and a few other post-quantum secure VRFs have appeared since.

Many blockchains utilize VRF through smart contracts, including Algorand [35] and Polkadot [41]. Chainlink [14] offers one of the most popular VRF services, but their construction is not decentralized; hence, when a specific node is compromised, the secret key is known to the attacker, and in turn, the output of the VRF is completely predictable. Therefore, constructing

decentralized VRF services is necessary.

A distributed VRF (DVRF) is a VRF that collectively provides a verifiably random and unpredictable source of data without relying on a single, trusted authority. This is crucial for ensuring fairness, transparency, and security in various applications. The construction of a DVRF poses many challenges, such as communication and computational complexity. To our knowledge, Dodis [25] proposed the first DVRFs under the requirement of a trusted dealer. Using unique aggregate signatures, Kuchta and Manulis [43] proposed a generic construction of DVRFs. Hanke et al. [16] introduced a BLS-pairing DVRF. Galindo et al. [33] provided the first systematic analysis and definitions for (Non-Interactive) Fully Distributed Verifiable Random Functions (DVRFs), where they introduced DDH-DVRF (a Decisional Diffie-Hellman DVRF) and GLOW-DVRF (a BLS-pairing DVRF) and provided comparisons to [16], which is now known as Dfinity-DVRF. The major drawback of these constructions is that the proof size of DDH-DVRF is $O(t)$, where t is the threshold number of participants. Therefore, the number of group actions needed for verification is in the same order. On the other hand, the construction of GLOW-DVRF is similar; although it has compact proofs, verification is done using costly procedures such as bilinear pairings. Kate et al. [40] introduced the notion of Output-Private DVRF, which is based on the GLOW-DVRF mechanism. Recently, Patel et al. [46] introduced RB-F-DVRF, improving the setup phase and reconstruction time for the output of GLOW-DVRF in large-scale applications using their novel ramp-based verifiable secret sharing.

The DVRF constructions above expose a structural tension in the design space: DDH-DVRF does not require pairings but has size $O(t)$ proof whose on-chain verification becomes increasingly expensive as the committee grows; whereas, GLOW-DVRF achieves a constant-size proof using pairings, whose verification cost is large and fixed. Our first construction, DVRFwCP [2], resolves both of these drawbacks simultaneously, but does so by adding interactions among the t participants per evaluation. This "trilemma", namely linear proofs, pairing cost, or interaction overhead, is the problem we try to address in this thesis. We propose Icy-DVRF to reduce this interaction cost, and build IcyVeil on top of it to achieve output privacy. The remainder of this chapter establishes the cryptographic foundations common to all three constructions.

2.2 Notation and Group Setup

Throughout this thesis, we fix the following notation and group-theoretic setup. We use the notation $s \xleftarrow{\$} S$ to mean *the element s is chosen with uniform probability from the set S*. Let $\mathbb{Z}_q$ be the residue ring of prime order q. From hereon, let $\mathbb{G} = \langle g \rangle$ be a group of prime order q. An element $y \in \mathbb{G}$ can be written as $y = g^x$ for $x \in \mathbb{Z}_q$; the integer x is called *the discrete logarithm of y with respect to g*, denoted $\log_g y$. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ denote the set of n participating parties, also referred to as the VRF committee or VRF clan, and let t denote the reconstruction threshold throughout.

2.3 Hardness Assumptions

The security of the constructions in this thesis rests on the following hardness assumptions in the group $\mathbb{G}$, each capturing a different aspect of the difficulty of the discrete logarithm problem.

Definition 2.1 (DDH Assumption [12])**.** Let $\mathbb{G} = \langle g \rangle$ be a (cyclic) group of prime order q, and let $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$. Let $\mathcal{A}$ be a PPT adversary. The Decisional Diffie-Hellman assumption holds if for $\gamma \xleftarrow{\$} \mathbb{Z}_q$ the value

$$\mathsf{Adv}^{\mathrm{ddh}}_{\mathbb{G}}(\mathcal{A}) = \left|\Pr\left[\mathcal{A}\left(\mathbb{G}, q, g, g^{\alpha}, g^{\beta}, g^{\alpha\beta}\right) = 1\right] - \Pr\left[\mathcal{A}\left(\mathbb{G}, q, g, g^{\alpha}, g^{\beta}, g^{\gamma}\right) = 1\right]\right|$$

is negligible.

Definition 2.2 (CDH Assumption [24])**.** Let $\mathbb{G} = \langle g \rangle$ be a (cyclic) group of prime order q, and let $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$. Let $\mathcal{A}$ be a PPT adversary. The Computational Diffie-Hellman assumption holds if the value

$$\mathsf{Adv}^{\mathrm{cdh}}_{\mathbb{G}}(\mathcal{A}) = \Pr\left[\mathcal{A}(\mathbb{G}, q, g, g^{\alpha}, g^{\beta}) = g^{\alpha\beta}\right]$$

is negligible.

Definition 2.3 (OMDL Assumption [9])**.** Let $\mathbb{G} = \langle g \rangle$ be a (cyclic) group of prime order q. Let $\mathcal{A}$ be a PPT adversary with access to a challenge oracle Chal and a discrete logarithm oracle Dlog. The Chal oracle, on each call i, samples $x_i \xleftarrow{\$} \mathbb{Z}_q$ and returns g^{x_i}. The Dlog oracle, on input $X \in \mathbb{G}$, returns $\log_g X$. Let $c_{\mathrm{id}} \geq 1$ be the number of calls to Chal and ℓ be the number of calls to Dlog. The One-More Discrete Logarithm assumption holds if the value

$$\mathsf{Adv}^{\mathrm{omdl}}_{\mathbb{G}}(\mathcal{A}) = \Pr\left[\mathcal{A}^{\mathsf{Chal},\mathsf{Dlog}}(\mathbb{G}, q, g) = (x_1, \ldots, x_{c_{\mathrm{id}}}) \wedge \ell < c_{\mathrm{id}}\right]$$

is negligible.

Definition 2.4 (Gap OM-CDH Assumption [38])**.** Let $\mathbb{G} = \langle g \rangle$ be a (cyclic) group of prime order q, and let $s \xleftarrow{\$} \mathbb{Z}_q$. Let $\mathcal{A}$ be a PPT adversary with access to a target-exponentiation oracle $\mathcal{O}_s \colon a \mapsto a^s$ and a decision oracle DDH_g, which on input $(a, b) \in \mathbb{G}^2$ returns 1 if and only if $b = a^s$. The adversary is given $(g, Y = g^s)$ together with N challenges $g_1, \ldots, g_N \xleftarrow{\$} \mathbb{G}$, and makes at most Q queries to $\mathcal{O}_s$. The Gap One-More Computational Diffie-Hellman assumption holds if the value

$$\mathsf{Adv}^{\text{gap-om-cdh}}_{\mathbb{G}}(\mathcal{A}) = \Pr\left[\mathcal{A}^{\mathcal{O}_s,\mathsf{DDH}_g}(g, Y, g_1, \ldots, g_N) \text{ outputs } g_j^s \text{ for } Q+1 \text{ distinct } j\right]$$

is negligible.

Definition 2.5 (Gap Threshold OM-CDH Assumption [38])**.** Fix $f \leq t - 1 < n$ and integers N, Q, and let $s \xleftarrow{\$} \mathbb{Z}_q$. Let $\mathcal{A}$ be a PPT adversary that declares a corrupt set $\mathcal{C} \subseteq \{1, \ldots, n\}$ with $|\mathcal{C}| = f$, together with values $\{s_i\}_{i \in \mathcal{C}} \subset \mathbb{Z}_q$. A polynomial $F \in \mathbb{Z}_q[X]$ of degree $t - 1$ is sampled subject to $F(0) = s$ and $F(i) = s_i$ for $i \in \mathcal{C}$. The adversary is given $\left(g, Y = g^s, \{g^{F(i)}\}_{i \in \{1,\ldots,n\}}\right)$ together with N challenges $g_1, \ldots, g_N \xleftarrow{\$} \mathbb{G}$, and has access

to a per-party oracle $\mathcal{O}_F(i, a) = a^{F(i)}$ for $i \notin \mathcal{C}$ only, together with the decision oracle DDH_g of Definition 2.4. Let q_i denote the number of queries made to slot i, so that $q_i = 0$ for $i \in \mathcal{C}$, and write $\vec{q} = (q_1, \ldots, q_n)$. For a weight $w \geq 1$, define

$$C_w(\vec{q}) = \max \left\{ k \geq 0 \ : \ \sum_{i=1}^{n} \min(q_i, k) \geq k\, w \right\},$$

the number of weight-w evaluations for which the profile $\vec{q}$ pays. With f shares known, completing one target costs $t - f$ fresh honest partial evaluations, so $C_{t-f}(\vec{q})$ is the number of full evaluations that the queries of $\mathcal{A}$ afford, and the realized profile must satisfy $C_{t-f}(\vec{q}) \leq Q$. The Gap Threshold One-More Computational Diffie-Hellman assumption holds if the value

$$\mathsf{Adv}_{\mathbb{G}}^{\text{gap-t-om-cdh}}(\mathcal{A}) = \Pr\left[\mathcal{A} \text{ outputs } g_j^s \text{ for } Q+1 \text{ distinct } j\right]$$

is negligible.

2.4 Lagrangian Interpolation

For a reconstruction set $\mathcal{S}$, the Lagrange basis polynomials are $\lambda_{j,\mathcal{S}}(X) = \prod_{k \in \mathcal{S} \setminus \{j\}} \frac{X-k}{j-k} \in \mathbb{Z}_q[X]$ with the Lagrange coefficients $\lambda_{j,\mathcal{S}} = \lambda_{j,\mathcal{S}}(0) \in \mathbb{Z}_q^*$. For any polynomial $f \in \mathbb{Z}_q[X]$ of degree at most $|\mathcal{S}| - 1$ this produces a way to reconstruct $f(X) = \sum_{i \in \mathcal{S}} \lambda_{i,\mathcal{S}}\ f(i)$. The process of reconstructing $f(0)$ is called *Lagrangian Interpolation*. We shall drop the subscript $\mathcal{S}$ and simplify the notation to write $\lambda_j = \lambda_{j,\mathcal{S}}$ for these coefficients whenever $\mathcal{S}$ is clear from the context.

2.5 Verifiable Secret Sharing

A (t, n)-threshold scheme is a cryptographic protocol designed for sharing a secret among a group of n participants so that any t participants working together can reconstruct the secret, while any subset of fewer than t participants gains no usable information. The three VSS schemes below form the building blocks of the distributed key generation protocols used throughout this thesis.

In [51], Shamir defined a way to share a secret $\sigma \in \mathbb{Z}_q$ among n parties using a polynomial of degree $t - 1$ so that any t parties or more could recover the secret, using the following protocol.

Shamir's Secret Sharing Protocol

1. The dealer chooses a random polynomial $f(X)$ over $\mathbb{Z}_q$ of degree $t - 1$, such that $f(0) = \sigma$, the secret being shared.
2. The dealer secretly transmits a share $s_i = f(i) \bmod q$ to the party P_i.

It is evident that $t-1$ or fewer parties cannot gather any information about the secret σ, whereas t or more parties can reconstruct it using Lagrange polynomial interpolation as described in Section 2.4. However, Shamir's protocol could fail as a dealer can share values that do not lie on a polynomial of degree $t-1$, and dishonest parties can contribute incorrect shares to disrupt reconstruction. These limitations motivated the development of *Verifiable Secret Sharing* (VSS) protocols.

In [31], Feldman extended Shamir's secret-sharing scheme, enabling share recipients to verify that the shares they receive from the dealer are consistent with the secret being shared (i.e., any subset of t shares yields the same unique secret) and to rule out the erroneous shares that dishonest parties submit during reconstruction. In Feldman's approach, which we call Feldman-VSS, the secret is merely *computationally secure*, i.e., the verification values are leaked; nevertheless, it can be shown that an adversary that holds $t-1$ or fewer shares cannot obtain any information on the secret except what can be learned from the public values.

Feldman's Verifiable Secret Sharing Protocol

1. The dealer chooses a random polynomial $f(X) = \sum_{k=0}^{t-1} a_k X^k$ over $\mathbb{Z}_q$ of degree $t-1$, such that $f(0) = \sigma$, the secret being shared.
2. The dealer broadcasts verification values $A_k = g^{a_k}$ for $k = 0, \ldots, t-1$.
3. The dealer secretly transmits a share $s_i = f(i) \bmod q$ to the party P_i.
4. The parties can verify that the shares s_i define a secret by checking
$$g^{s_i} = \prod_{k=0}^{t-1} (A_k)^{i^k} \quad . \tag{2.1}$$
5. (a) Each party P_i verifies (2.1) for their share s_i.
   (b) If verification fails, P_i broadcasts a *complaint* against the dealer.
   (c) For each complaining party P_i, the dealer reveals the share s_i matching Equation (2.1).
   (d) If any of the revealed shares fails (2.1), the dealer is disqualified.
   (e) At the time of reconstruction of the secret σ, the same equation is used to verify the shares submitted by the parties so that dishonest parties may be identified. If at least t honest parties exist, the secret may be reconstructed using Lagrangian interpolation.

Pedersen's Verifiable Secret Sharing (Pedersen-VSS) [48], in comparison to Feldman-VSS, offers perfect (information-theoretic) security for the shared secret, which means that the adversary can merely view values unrelated to the secret being shared under the assumption

that any adversary, especially the dealer, cannot solve the discrete logarithm problem. In addition to the generator g of the cyclic group $\mathbb{G} = \langle g \rangle$ of prime order q used in Feldman-VSS, this scheme requires an additional element $\bar{g} \in \mathbb{G}$ where it is assumed that $\log_g(\bar{g})$ is hard to compute. To share a secret σ among a group of participants $\mathcal{P} = \{P_1, \ldots, P_n\}$, Pedersen-VSS works as follows.

Pedersen's Verifiable Secret Sharing Protocol

1. The dealer generates two random polynomials $f(X) = \sum_{k=0}^{t-1} a_k X^k$ and $\bar{f}(X) = \sum_{k=0}^{t-1} b_k X^k$ over $\mathbb{Z}_q$, both of degree $t-1$, such that $f(0) = \sigma$.
2. The dealer publishes $C_k = g^{a_k} \bar{g}^{b_k}$ for $k = 0, \ldots, t-1$.
3. The dealer transmits $(s_i, \bar{s}_i) = \big(f(i), \bar{f}(i)\big)$ to P_i, for $i = 1, \ldots, n$.
4. (a) Each party $P_i \in \mathcal{P}$ checks

$$g^{s_i} \bar{g}^{\bar{s}_i} = \prod_{k=0}^{t-1} (C_k)^{i^k} \quad . \tag{2.2}$$

   (b) If the check fails, P_i complains against the dealer.

   (c) For each complaining party P_i, the dealer reveals $(s_i, \bar{s}_i)$ matching (2.2).

   (d) If any of the revealed shares fails (2.2), the dealer is disqualified.

In [48], it was shown that if the dealer is not disqualified, all honest parties can interpolate a unique polynomial of degree $t-1$, and any t of these honest parties can obtain the secret efficiently, with each share verifiable publicly.

2.6 Distributed Key Generation

Based on the Feldman-VSS protocol, in [47], Pedersen proposed the first distributed key generation protocol (DKG), which executes n parallel Feldman's VSS protocols for the parties $P_1, \ldots, P_n$. Gennaro et al. [34] showed that during a run of Pedersen's DKG, an attacker could affect the distribution of the secret key and public key pair, resulting in a non-uniform distribution. They proposed the following protocol, which is utilized in most distributed systems, including the constructions developed in this thesis. Similar to Pedersen's VSS, the Secure-DKG protocol contains an element $\bar{g} \in \mathbb{G}$ where computing $\log_g(\bar{g})$ is infeasible.

Secure-DKG Protocol

Generating x :

1. Each party $P_i \in \mathcal{P}$ performs a Pedersen-VSS of a random value z_i as a dealer:

   (a) P_i chooses two random secret polynomials $f_i(X), \bar{f}_i(X)$ over $\mathbb{Z}_q$ of degree $t-1$:

   $$\begin{aligned} f_i(X) &= a_{i,0} + a_{i,1}X + \cdots + a_{i,t-1}X^{t-1}, \\ \bar{f}_i(X) &= b_{i,0} + b_{i,1}X + \cdots + b_{i,t-1}X^{t-1}, \end{aligned}$$

   where $a_{i,0} = z_i = f_i(0)$.

   (b) P_i broadcasts $C_{i,k} = g^{a_{i,k}}\ \bar{g}^{b_{i,k}}$ for $k = 0, \ldots, t-1$.

   (c) P_i computes the shares $s_{i,j} = f_i(j)$ and $\bar{s}_{i,j} = \bar{f}_i(j) \bmod q$, and sends $(s_{i,j}, \bar{s}_{i,j})$ to each party $P_j \in \mathcal{P}$.

   (d) Each party $P_j \in \mathcal{P}$ verifies the shares they received from the other parties, i.e., for each $P_i \in \mathcal{P}$, P_j checks whether

   $$g^{s_{i,j}}\ \bar{g}^{\bar{s}_{i,j}} \stackrel{?}{=} \prod_{k=0}^{t-1} (C_{i,k})^{j^k} \tag{2.3}$$

   If the check fails for any index i, P_j broadcasts a *complaint* against P_i.

   (e) Each party P_i, acting as a dealer, who received a complaint from a party P_j, broadcasts the values $(s_{ij}, \bar{s}_{ij})$ that satisfy Equation (2.3).

   (f) Each party P_j marks P_i as *disqualified* if either

   - P_i received more than $t-1$ complaints in Step 1(d), or
   - P_i answered a complaint in Step 1(e) with values that do not satisfy Equation (2.3).

2. Each party builds the set of non-disqualified parties $\mathcal{Q}$, where it was shown in [34] that all honest parties build the same $\mathcal{Q}$.

3. The distributed secret value x is (usually) not explicitly computed by any party, but it is

   $$x = \sum_{i \in \mathcal{Q}} z_i \bmod q\ . \tag{2.4}$$

   Each party P_i sets their share of the secret as

   $$x_i = \sum_{j \in \mathcal{Q}} s_{j,i} \bmod q \qquad \text{and} \qquad \bar{x}_i = \sum_{j \in \mathcal{Q}} \bar{s}_{j,i} \bmod q\ .$$

Extracting $y = g^x$:

1. Each party $P_i \in \mathcal{Q}$ publishes $y_i = g^{z_i}$ via Feldman-VSS as follows.

(a) P_i broadcasts $A_{i,k} = g^{a_{i,k}}$ for $k = 0, \ldots, t-1$.

(b) Each party $P_j \in \mathcal{Q}$ verifies the values publicized by the other parties in $\mathcal{Q}$, i.e., for each $P_i \in \mathcal{Q}$, P_j checks if

$$g^{s_{i,j}} \stackrel{?}{=} \prod_{k=0}^{t-1} (A_{i,k})^{j^k} \quad . \tag{2.5}$$

If the check fails for a party member P_i, P_j broadcasts the values $(s_{ij}, \bar{s}_{ij})$ that satisfy Equation (2.3) but do not satisfy Equation (2.5), thereby broadcasts a *complaint* against P_i.

(c) For parties P_i who receive at least one valid complaint, i.e., values which satisfy Equation (2.3) and not Equation (2.5), the remaining parties run the reconstruction phase of Pedersen-VSS to compute z_i, $f_i(X)$, and $A_{i,k}$ for $k = 0, \ldots, t-1$. For all parties in $\mathcal{Q}$, let $y_i = A_{i,0} = g^{z_i}$ and compute y as

$$y = \prod_{i \in \mathcal{Q}} y_i \ .$$

2.7 Non-Interactive Zero-Knowledge Proof Systems

In this thesis, we will be using "proof systems" that are built from Σ-protocols compiled into non-interactive form. We introduce these notions in order: the interactive Σ-protocol, the Fiat-Shamir transform that removes interaction, and the resulting non-interactive zero-knowledge (NIZK) proof system. The Schnorr proof establishes knowledge of a discrete logarithm, while the Chaum-Pedersen proof attests to the equality of two discrete logarithms with respect to different bases.

2.7.1 Σ-Protocols and Non-Interactive Zero-Knowledge

A Σ-protocol is a three-move interactive proof in which a prover convinces a verifier that it knows a witness for a given statement, without revealing the witness itself. First, the prover sends a commitment; then the verifier responds with a random challenge; and lastly, the prover answers in a way that it could only produce with knowledge of the witness. This structure underlies a large family of efficient proofs of knowledge in the discrete-logarithm setting.

Definition 2.6 (Σ-Protocol [20, 22])**.** Let $\mathcal{R}$ be a relation. A Σ-*protocol* for $\mathcal{R}$ is a three-move public-coin protocol for a statement x with witness w, in which the prover sends a commitment a, the verifier replies with a uniformly random challenge c, and the prover responds with z; the verifier then accepts or rejects based on (x, a, c, z). It satisfies the following three properties.

- *Completeness*: If $(x, w) \in \mathcal{R}$ and both parties follow the protocol, the verifier accepts.

- *Special Soundness*: There exists an efficient "extractor" that, given two accepting transcripts (a, c, z) and (a, c', z') for the same commitment a with $c \neq c'$, recovers a valid witness w.

- *Special Honest-Verifier Zero-Knowledge*: There exists a PPT simulator that, on input x and a challenge c, outputs a transcript (a, c, z) with the same probability distribution as a real transcript between an honest prover and an honest verifier, without knowledge of the witness w.

There are different notions of soundness, and which one we use matters for our constructions. Plain *soundness* requires only that a prover cannot produce an accepting proof for a false statement. For the proofs we use in this thesis, this notion falls short: the statement "x has a discrete logarithm base g" is true for every $x \in \mathbb{G}$, since $\mathbb{G}$ is generated by the single element g of prime order q, so plain soundness is vacuous. What we actually want to capture is that the prover knows that discrete logarithm. This is the role of *knowledge-soundness* [8], which guarantees that a prover producing an accepting proof must know the witness, in the sense that the witness can be efficiently extracted from such a prover. This is the property that defines a *proof of knowledge*.

For Σ-protocols, *special soundness* is the mechanism that provides knowledge-soundness [20]. Instead of reasoning about extraction directly, it is enough to give an extractor that recovers the witness from any two accepting transcripts with the same commitment a and different challenges $c \neq c'$. Such a pair is obtained by running the prover, rewinding it to the point just after it sends a, and supplying a fresh challenge. This reduces knowledge-soundness to a simple algebraic condition, and it is the property that we need in our later security proofs to extract secret keys from a successful adversary.

A Σ-protocol is interactive: it requires the verifier to supply a fresh random challenge during the exchange. In many settings, and in particular for on-chain verification where no interactive verifier is available, this interaction is undesirable. The Fiat-Shamir heuristic removes it by replacing the verifier's random challenge with the output of a hash function applied to the statement and the prover's commitment, that is, $c \leftarrow H(x, a)$. Since a hash function modeled as a random oracle returns an unpredictable value, the prover cannot anticipate the challenge before committing, which preserves the soundness of the interactive protocol while collapsing it into a single non-interactive message.

Definition 2.7 (Fiat-Shamir Transform [32])**.** Let Π be a Σ-protocol for a relation $\mathcal{R}$, and let H be a hash function modeled as a random oracle. The *Fiat-Shamir transform* compiles Π into a non-interactive proof system $(\mathrm{Prove}, \mathrm{Verify})$ as follows.

- $\mathrm{Prove}(x, w)$ runs the prover to obtain a commitment a, computes the challenge $c \leftarrow H(x, a)$ in place of the verifier, runs the prover to obtain the response z, and outputs the proof $\pi = (c, z)$.

- $\mathrm{Verify}(x, \pi)$ parses $\pi = (c, z)$, recovers the commitment a from (x, c, z) as the verifier's check prescribes, and accepts if and only if $c = H(x, a)$.

In the random oracle model, the resulting system is a non-interactive zero-knowledge proof system for $\mathcal{R}$, defined as follows.

Definition 2.8 (Non-Interactive Zero-Knowledge Proof System [37])**.** Let $\mathcal{R}$ be an NP relation with associated language $L_{\mathcal{R}} = \{x : \exists w, (x, w) \in \mathcal{R}\}$. A non-interactive zero-knowledge (NIZK) proof system for $\mathcal{R}$ is a pair of algorithms $(\mathrm{Prove}, \mathrm{Verify})$ satisfying the following properties.

- *Completeness*: For every $(x, w) \in \mathcal{R}$, $\mathrm{Verify}(x, \mathrm{Prove}(x, w)) = 1$.
- *Soundness*: For every $x \notin L_{\mathcal{R}}$ and every computationally bounded prover $\mathcal{A}$, the probability that $\mathrm{Verify}(x, \mathcal{A}(x)) = 1$ is negligible.
- *Zero-Knowledge*: There exists a PPT simulator $\mathcal{S}$ such that for every $(x, w) \in \mathcal{R}$, the distribution of $\mathcal{S}(x)$ is computationally indistinguishable from that of $\mathrm{Prove}(x, w)$.

Together, these notions explain how one converts an interactive proof of knowledge into a non-interactive one. Special soundness carries over to knowledge-soundness via rewinding, and special honest-verifier zero-knowledge becomes full zero-knowledge through a simulator that programs the random oracle. The Schnorr and Chaum-Pedersen proof systems we introduce next are two instances of this construction.

2.7.2 Schnorr Proof of Knowledge of Exponent

The non-interactive zero-knowledge (NIZK) proof for the knowledge of the exponent, introduced by Schnorr [50], is a pair of algorithms $(\mathrm{ProveExp}_H, \mathrm{VerifyExp}_H)$ to prove the statement $x = g^k$. Consider a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$ modeled as a random oracle.

- $\mathrm{ProveExp}_H(g, x; k)$ chooses $r \xleftarrow{\$} \mathbb{Z}_q$, computes $u \leftarrow g^r$ and sets $c \leftarrow H(g, x, u)$. The output is a non-interactive proof $\pi = (c, s)$, where $s = r - k \cdot c$.
- $\mathrm{VerifyExp}_H(g, x, \pi)$ parses $\pi = (c, s)$, computes $\bar{u} \leftarrow g^s x^c$, and outputs 1 if $c = H(g, x, \bar{u})$; otherwise outputs 0.

2.7.3 Chaum-Pedersen DLEQ Proof

The Discrete Logarithm Equality (DLEQ) proof system by Chaum and Pedersen [17] is a pair of algorithms $(\mathrm{ProveEq}_H, \mathrm{VerifyEq}_H)$ to create a Non-Interactive Zero-Knowledge (NIZK) proof of $k = \log_g x = \log_h y$, as detailed below. Consider a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$ modeled as a random oracle.

- $\text{ProveEq}_H(g, h, x, y; k)$ chooses $r \xleftarrow{\$} \mathbb{Z}_q$, computes $u \leftarrow g^r$, $v \leftarrow h^r$ and sets $c \leftarrow H(g, h, x, y, u, v)$. The output is a non-interactive proof $\pi = (c, s)$, where $s = r - k \cdot c$.
- $\text{VerifyEq}_H(g, h, x, y, \pi)$ parses $\pi = (c, s)$, computes $\bar{u} \leftarrow g^s x^c$ and $\bar{v} \leftarrow h^s y^c$, and outputs 1 if $c = H(g, h, x, y, \bar{u}, \bar{v})$; otherwise outputs 0.

2.8 FROST: Flexible Round-Optimized Schnorr Threshold Signatures

The Flexible Round-Optimized Schnorr Threshold Signatures (FROST) protocol, developed by Komlo and Goldberg [42], represents a significant advancement in threshold cryptography. It effectively tackles the major obstacles that previously limited Schnorr-based threshold schemes by offering improved round efficiency (one round with preprocessing or two rounds), which is crucial for performance. FROST introduced a novel *nonce binding* technique that ensures provable security against concurrent execution attacks, such as those by Drijvers et al. [27] and the attack via ROS Solver by Benhamouda et al. [11], which are significant vulnerabilities in earlier protocols.

FROST uses a Signature Aggregator (SA) for efficiency, which can be one of the signers, collecting, verifying, and combining the partial signatures provided by the participating signers during the final signing stage. The SA, which can be any participant or external party with information of the public key shares, is responsible for collecting signatures, reporting misbehaving participants, and publishing the final group signature. While trusted for these tasks, a malicious SA cannot compromise the private key or cause improper messages to be signed. They are limited to performing denial-of-service attacks or falsely reporting misbehavior. The FROST protocol remains secure against adaptive attacks, even if the SA is malicious, because its view is not more privileged than that of a standard adversary.

FROST consciously trades robustness for efficiency. Unlike protocols that guarantee a valid signature output even if up to $t - 1$ participants misbehave (often requiring additional communication rounds or complexity), FROST adopts an *abort and identify* model. If any participant submits an invalid partial signature during the signing process, the SA detects this during the mandatory verification step. The protocol instance is then aborted for that session, and the misbehaving participant is identified. Honest participants could subsequently retry the signing operation, potentially excluding the identified faulty party. This is a pragmatic choice suitable for practical deployments where malicious behavior or failures are considered exceptional events, allowing for optimized performance in the common, optimistic case where all participants follow the protocol correctly.

We adopt and adapt the nonce binding technique and abort-and-identify model of FROST in Icy-DVRF (Chapter 4) and IcyVeil (Chapter 5) to eliminate the interactive round of DVRFwCP while maintaining compact, pairing-free proofs.

2.9 Two Prior Baseline DVRF Constructions

We recall the two baseline DVRF constructions against which the protocols in this thesis are benchmarked. Both protocols share the same group setup: $(\mathbb{G}, g, q)$ is a multiplicative group of prime order q where the DDH assumption holds, and $H_1 : \{0,1\}^* \to \mathbb{G}$, $H_2 : \{0,1\}^* \to \mathbb{Z}_q$ are hash functions modeled as random oracles. Before presenting the constructions, we recall the standard security properties a DVRF must satisfy.

- *Correctness*: This requires that a correctly generated VRF value β always passes verification.
- *Consistency*: This means that, for a given input α, successful execution of the protocol should produce the same VRF output β, regardless of the collection of correctly formed shares used.
- *Public verifiability*: Anyone should be able to verify the proof for the correctness of the pseudorandom value β, removing the need to trust the parties involved in the VRF value or proof generation.
- *Uniqueness*: For a given input α, the VRF output value β should be deterministic. This is particularly vital in decentralized applications, as it is crucial for selecting block proposers and validators, preventing forks, and ensuring consensus.
- *Pseudorandomness*: This ensures that the output VRF value β is unpredictable before its creation and that its distribution is statistically indistinguishable from random. This property also implies that no one can bias the output in their favor.

2.9.1 DDH-DVRF

Proposed by Galindo et al. [33] in 2020, DDH-DVRF constructs a distributed verifiable random function without relying on bilinear pairings, secured by the DDH assumption. It consists of four algorithms $\mathcal{V}^{\text{DDH-DVRF}} = (\text{DistKeyGen}, \text{PartialEval}, \text{Combine}, \text{Verify})$, defined as follows.

$\text{DistKeyGen}\left(1^\lambda, t, n\right)$ is run by n participating nodes $\mathcal{P} = \{P_1, \ldots, P_n\}$. Each node P_i chooses a random polynomial $f_i(X) = a_{i,0} + a_{i,1}X + \cdots + a_{i,t-1}X^{t-1}$. The protocol outputs a set of qualified nodes $\text{QUAL} \subseteq \mathcal{P}$, a secret key $\text{sk}_i = \sum_{j \in \text{QUAL}} f_j(i) \in \mathbb{Z}_q$ and a verification key $\text{vk}_i = g^{\text{sk}_i} \in \mathbb{G}$ for each $i \in \text{QUAL}$, an implicit distributed secret value $\text{sk} = \sum_{i \in \text{QUAL}} a_{i,0} \in \mathbb{Z}_q$, a global public key $\text{pk} = \prod_{i \in \text{QUAL}} g^{a_{i,0}}$, and the set of verification keys $\mathcal{VK} = (\text{vk}_1, \ldots, \text{vk}_n)$.

$\text{PartialEval}\left(\alpha, \text{sk}_i, \text{vk}_i\right)$ outputs $s_\alpha^i = (i, \gamma_i, \pi_i)$ for a plaintext α, where $\gamma_i \leftarrow H_1(\alpha)^{\text{sk}_i}$ and $\pi_i \leftarrow \text{ProveEq}_{H_2}\left(g, H_1(\alpha), \text{vk}_i, \gamma_i; \text{sk}_i\right)$.

$\mathrm{Combine}(\mathrm{pk}, \mathcal{VK}, \alpha, \mathcal{S})$ parses list $\mathcal{S} = \left\{s_\alpha^{j_1}, \ldots, s_\alpha^{j_{|\mathcal{S}|}}\right\}$ of $|\mathcal{S}| \geq t$ partial evaluation candidates and:

1. Identifies an index subset $I = \{i_1, \ldots, i_t\}$ such that $\mathrm{VerifyEq}_{H_2}(g, H_1(\alpha), \mathrm{vk}_i, \gamma_i, \pi_i) = 1$ holds for every $i \in I$, where $(i, \gamma_i, \pi_i) \leftarrow s_\alpha^i$. If no such subset exists, aborts.
2. Sets $\gamma \leftarrow \prod_{i \in I} \gamma_i^{\lambda_{0,i,I}}$ and $\pi \leftarrow \left\{s_\alpha^i\right\}_{i \in I}$.
3. Outputs (γ, π).

$\mathrm{Verify}(\mathrm{pk}, \mathcal{VK}, \alpha, \gamma, \pi)$ parses $\pi = \left\{s_\alpha^i\right\}_{i \in I}$ such that $|I| = t$ and $I \subseteq \mathrm{QUAL}$:

1. Parses $s_\alpha^i = (i, \gamma_i, \pi_i)$ for $i \in I$.
2. Checks if $\mathrm{VerifyEq}_{H_2}(g, H_1(\alpha), \mathrm{vk}_i, \gamma_i, \pi_i) \stackrel{?}{=} 1$ for every $i \in I$; if some of the checks fail, outputs 0.
3. Checks if $\gamma \stackrel{?}{=} \prod_{i \in I} \gamma_i^{\lambda_{0,i,I}}$; if so outputs 1; otherwise outputs 0.

The drawback of this construction is that the combined proof size grows linearly with the threshold number of participants t, and the verification algorithm must verify each public partial evaluation separately, which makes verification slow and costly, and places a burden on the public verifiers.

2.9.2 GLOW-DVRF

The GLOW-DVRF algorithm is a pairing-based DVRF, similar to DDH-DVRF, that uses bilinear pairings and BLS signatures [13] to build a distributed verifiable random function. Let $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, h_1, h_2)$ be a bilinear pairing where g_1, h_1 and g_2, h_2 are generators of $\mathbb{G}_1, \mathbb{G}_2$, respectively. Let $H_1 : \{0,1\}^* \to \mathbb{G}_1$, $H_2 : \{0,1\}^* \to \mathbb{Z}_q$ and $H_3 : \mathbb{G}_1 \to \{0,1\}^\Lambda$ be hash functions, where Λ is the desired output length. It consists of four algorithms $\mathcal{V}^{\text{GLOW-DVRF}} = (\mathrm{DistKeyGen}, \mathrm{PartialEval}, \mathrm{Combine}, \mathrm{Verify})$, defined as follows.

$\mathrm{DistKeyGen}\left(1^\lambda, t, n\right)$ proceeds as in DDH-DVRF, with the difference that the verification keys are placed in $\mathbb{G}_1$ while the global public key is placed in $\mathbb{G}_2$. Each node P_i with $i \in \mathrm{QUAL}$ also exposes $B_{i,0} = g_2^{a_{i,0}}$ via Feldman-VSS, and each node P_j checks $e\left(A_{i,0}, g_2\right) = e\left(g_1, B_{i,0}\right)$ together with the usual share verification. The protocol outputs a set of qualified nodes $\mathrm{QUAL} \subseteq \mathcal{P}$, a secret key $\mathrm{sk}_i = \sum_{j \in \mathrm{QUAL}} f_j(i) \in \mathbb{Z}_q$ and a verification key $\mathrm{vk}_i = g_1^{\mathrm{sk}_i} \in \mathbb{G}_1$ for each $i \in \mathrm{QUAL}$, an implicit distributed secret value $\mathrm{sk} = \sum_{i \in \mathrm{QUAL}} a_{i,0} \in \mathbb{Z}_q$, a global public key $\mathrm{pk} = \prod_{i \in \mathrm{QUAL}} g_2^{a_{i,0}} \in \mathbb{G}_2$, and the set of verification keys $\mathcal{VK} = (\mathrm{vk}_1, \ldots, \mathrm{vk}_n)$.

$\mathrm{PartialEval}\,(\alpha, \mathrm{sk}_i, \mathrm{vk}_i)$ outputs $s_\alpha^i = (i, \gamma_i, \pi_i)$ for a plaintext α, where $\gamma_i \leftarrow H_1(\alpha)^{\mathrm{sk}_i}$ and $\pi_i \leftarrow \mathrm{ProveEq}_{H_2}\,(g_1, H_1(\alpha), \mathrm{vk}_i, \gamma_i; \mathrm{sk}_i)$.

$\mathrm{Combine}(\mathrm{pk}, \mathcal{VK}, \alpha, \mathcal{S})$ parses list $\mathcal{S} = \left\{s_\alpha^{j_1}, \ldots, s_\alpha^{j_{|\mathcal{S}|}}\right\}$ of $|\mathcal{S}| \geq t$ partial evaluation candidates and:

1. Identifies an index subset $I = \{i_1, \ldots, i_t\}$ such that $\mathrm{VerifyEq}_{H_2}\,(g_1, H_1(\alpha), \mathrm{vk}_i, \gamma_i, \pi_i) = 1$ holds for every $i \in I$, where $(i, \gamma_i, \pi_i) \leftarrow s_\alpha^i$. If no such subset exists, aborts.
2. Sets $\pi \leftarrow \prod_{i \in I} \gamma_i^{\lambda_{0,i,I}}$ and $\gamma \leftarrow H_3(\pi)$.
3. Outputs (γ, π).

$\mathrm{Verify}(\mathrm{pk}, \alpha, \gamma, \pi)$ checks the pairing relation over the single combined element rather than each share:

1. Checks if $e\,(\pi, g_2) \stackrel{?}{=} e\,(H_1(\alpha), \mathrm{pk})$; if the check fails, outputs 0.
2. Checks if $\gamma \stackrel{?}{=} H_3(\pi)$; if so outputs 1; otherwise outputs 0.

Unlike DDH-DVRF, the combined proof of GLOW-DVRF is a single element of $\mathbb{G}_1$ whose size does not grow with the threshold t, and the Verify algorithm settles correctness with one pairing equation instead of checking each partial evaluation on its own. The cost of this compactness is the pairing operation itself, which is expensive to compute and, on blockchains, translates into high and fixed gas fees. Placing the verification keys in $\mathbb{G}_1$ keeps the checking of the partial evaluations pairing-free, since each share is validated with a proof of discrete logarithm equality inside a single group rather than with a pairing equation, and the public key is placed in $\mathbb{G}_2$ so that the combined element can be settled with the single pairing check above. This placement is also what the strong pseudorandomness proof of [33] depends on, since it prevents an adversary from checking a partial evaluation on its own.

CHAPTER 3

DVRFWCP: COMPACT PROOFS WITHOUT PAIRINGS

3.1 The Two-Sided Problem

The two systematic constructions of Galindo et al. [33], DDH-DVRF and GLOW-DVRF, occupy opposite corners of the design space, and neither is fully optimal for on-chain use.

DDH-DVRF is simple and free of bilinear pairings, but pays for this simplicity in the size of its proofs and the verification cost that grows accordingly. DDH-DVRF uses Chaum-Pedersen DLEQ Proofs to construct the DVRF share and each indiviual proof, then concatenates the proofs. The major drawback of this constructions is that the proof size of DDH-DVRF is $O(t)$, where t is the threshold number of participants. Therefore, the number of group actions needed for verification is in the same order.

On the other hand, the construction of GLOW-DVRF is similar: it uses Chaum-Pedersen DLEQ Proofs to construct the DVRF share and each indiviual proof, but then provides a compact aggregate proof. Although it has compact proofs, verification is done using costly procedures such as bilinear pairings.

Neither option is the most efficient for a smart-contract verifier. A proof whose size and verification cost grow linearly in t becomes prohibitively expensive on-chain as the committee grows, whereas a proof that depends on bilinear pairings incurs a large fixed verification cost regardless of committee size. This raises a natural question: whether a distributed VRF can achieve constant-size proofs and avoid pairings at the same time.

In this chapter, we answer this question in the affirmative by proposing a distributed verifiable random function with compact proofs (DVRFwCP) that does not require bilinear pairings and has constant proof size, independent of the threshold number of participants. Our approach alters and uses the Chaum-Pedersen NIZK proof system (see Section 2.7.3) by adding a t-out-of-n threshold structure and needs at least t participants to generate verifiable randomness, whereas requiring these parties to generate the NIZK proof jointly.

Unfortunately, this gain comes at a cost: the drawback of our construction is that an extra interaction of two rounds amongst the n qualified parties is required to create a shared

challenge for the NIZK. This makes the proof generation and combination more costly for the participants but makes the size of the proof constant and final verification less costly. In the blockchain setting, the entirety of the distributed key generation and the mentioned interaction can be performed off-chain; hence, both the sizes of the partial proofs and the time to obtain a joint proof could be, in this sense, ignored. This interactive round is the dominant cost of DVRFwCP. Reducing the number of rounds and interaction cost without sacrificing the compact, pairing-free proof is the problem addressed in Chapter 4.

3.2 The Augmented Secure-DKG

The joint generation of a single Chaum-Pedersen proof requires the participating nodes to share an ephemeral secret $\mathbf{k}$ and to hold its commitments in *both* bases g and h at once, since the proof asserts equality of two discrete logarithms taken with respect to these two generators. The Secure-DKG protocol of Gennaro et al. [34], recalled in Section 2.6, produces a shared secret together with a commitment in a single base only. We therefore introduce a modest extension of it and present a new "augmented" version of the Secure-DKG protocol, which will be used in our construction.

Protocol Augmented Secure-DKG.

This protocol extends the Secure-DKG protocol above to share commitments to the shared secret x relative to an auxiliary element $h \in \mathbb{G}$, where it is assumed that $\log_g(h)$ is hard to compute. This is done to reduce the number of interactions in our proposed algorithm. The generation phase of the shared secret x is the same, whereas the modifications to the extraction phase are emphasized in the color blue.

Extracting $y = g^x$:

1. Each party $P_i \in \mathcal{Q}$ publishes $y_i = g^{z_i}$ via Feldman-VSS as follows.
   (a) P_i broadcasts $A_{i,k} = g^{a_{i,k}}$ and $B_{i,k} = h^{a_{i,k}}$ for $k = 0, \ldots, t-1$.
   (b) Each party $P_j \in \mathcal{Q}$ verifies the values publicised by the other parties in $\mathcal{Q}$, i.e, for each $P_i \in \mathcal{Q}$, P_j checks if

$$g^{s_{i,j}} \stackrel{?}{=} \prod_{k=0}^{t-1} (A_{i,k})^{j^k} \quad . \tag{3.1}$$

$$h^{s_{i,j}} \stackrel{?}{=} \prod_{k=0}^{t-1} (B_{i,k})^{j^k} \quad . \tag{3.2}$$

   If the checks fail for a party member P_i, P_j broadcasts the values $(s_{ij}, \bar{s}_{ij})$ that satisfy Equation (2.3) but do not satisfy Equation (2.5), thereby broadcasts a *complaint* against P_i.

(c) For parties P_i who receive at least one valid complaint, i.e., values which satisfy Equation (2.3) and not Equation (2.5), the remaining parties run the reconstruction phase of Pedersen-VSS to compute z_i, $f_i(X)$, and $A_{i,k}$ for $k = 0, \ldots, t-1$. For all parties in $\mathcal{Q}$, compute

$$g^x = \prod_{i \in \mathcal{Q}} A_{i,0} \quad \text{and} \quad h^x = \prod_{i \in \mathcal{Q}} B_{i,0} \ .$$

The additional commitments $B_{i,k} = h^{a_{i,k}}$ are computed using the group element h, and the $a_{i,k}$ are kept private. The published values are $A_{i,k} = g^{a_{i,k}}$ and $B_{i,k} = h^{a_{i,k}}$, and extracting $a_{i,k}$ from either of them is the discrete logarithm problem in $\mathbb{G}$, whose hardness is implied by the CDH assumption (Definition 2.2). Publishing the pair reveals no more than the single commitment of Secure-DKG: given a challenge $A = g^a$ and a self-chosen $h = g^\eta$ with known η, one forms $B = A^\eta = h^a$, so any adversary recovering $a_{i,k}$ from $(A_{i,k}, B_{i,k})$ recovers it from $A_{i,k}$ alone. The hardness of $\log_g(h)$ assumed above is therefore not what conceals the coefficients. Its role lies elsewhere: since $h = H_1(\alpha)$ is a random oracle output, no party holds a representation $h = g^\eta$, so $(g, g^{\mathbf{x}}, h)$ is a properly distributed CDH instance. The evaluation $\gamma = h^{\mathbf{x}}$ is then not computable from the public key $g^{\mathbf{x}}$ under the CDH assumption (Definition 2.2), and not distinguishable from a uniform group element under DDH (Definition 2.1). If η were known, we would have $\gamma = (g^{\mathbf{x}})^\eta$ and both instances would degenerate.

The verification in Equation (3.2) succeeds for honest shares because $\prod_{k=0}^{t-1} (B_{i,k})^{j^k} = \prod_{k=0}^{t-1} h^{a_{i,k} j^k} = h^{\sum_k a_{i,k} j^k} = h^{f_i(j)} = h^{s_{i,j}}$, exactly mirroring the original Secure-DKG check in base g. Therefore, the augmentation thus inherits the verifiability and security guarantees of Secure-DKG at the cost of one additional group element broadcast per coefficient.

3.3 Protocol Definition

We now give the formal syntax of DVRFwCP together with the security properties it must satisfy. The concrete construction realizing this syntax is presented in Section 3.4. Since we are working in a distributed environment, we assume that the participants communicate with each other in a secure and authenticated channel.

Definition 3.1 (Distributed Verifiable Random Function with Compact Proof)**.** A t-out-of-n Distributed Verifiable Random Function with Compact Proof (DVRFwCP) consists of the six algorithms, i.e., $\mathcal{V}^{\text{DVRFwCP}} = (\text{DistKeyGen}, \text{PartialEval}, \text{EvalCombine}, \text{PartialProofGen}, \text{ProofCombine}, \text{Verify})$, described below.

- $\text{DistKeyGen}\left(1^\lambda, t, n\right)$: On input of a security parameter λ, the number of participating nodes n, and the threshold parameter t as inputs, and outputs a public key pk, a list of

partial secret keys $\mathcal{SK} = (x_1, \ldots, x_n)$, and a list of partial verification keys $\mathcal{VK} = (y_1, \ldots, y_n)$.

- $\mathrm{PartialEval}\,(\alpha, x_i, y_i)$: On input of a plaintext α, a partial secret and verification keys (x_i, y_i) as inputs, and outputs a partial evaluation γ_i and a partial evaluation NIZK proof π_i.

- $\mathrm{EvalCombine}(\mathcal{VK}, \alpha, \mathcal{S})$: On input of the list of verification keys $\mathcal{VK}$, the plaintext α and a list $\mathcal{S} = \left\{s_{i_1}, \ldots, s_{i_{|\mathcal{S}|}}\right\}$ of partial evaluation proofs from at least t nodes, outputs either a combined evaluation γ or aborts.

- $\mathrm{PartialProofGen}\,(\alpha, x_i, y_i)$: On input of a plaintext α, a partial secret and verification key (x_i, y_i) as inputs and outputs a partial NIZK proof $\Pi_i = (c, s_i)$.

- $\mathrm{ProofCombine}(\mathcal{VK}, \alpha, \mathcal{S})$: On input of the list of verification keys $\mathcal{VK}$, the plaintext α and a list $\mathcal{S} = \left\{\Pi_{i_1}, \ldots, \Pi_{i_{|\mathcal{S}|}}\right\}$ of partial proofs from at least t nodes, outputs either a combined NIZK proof $\Pi = (\gamma, c, \mathbf{s})$ or aborts.

- $\mathrm{Verify}(pk, \alpha, \beta, \Pi)$: On input of the public key pk, the plaintext α, the VRF value β, a NIZK proof Π as inputs, it outputs $0/1$.

A DVRFwCP needs to satisfy the following requirements.

- **Correctness:** This requires that for a correctly generated VRF value γ, Verify always outputs 1.

- **Uniqueness:** This requires that for every plaintext α a unique value γ passes the verification test. It is infeasible for an adversary to calculate two different output values γ_1 and γ_2 where both values pass the verification test with respect to α, even if the secret keys of the honest parties are compromised.

- **Consistency:** This means that, for a given plaintext α, successful execution of the protocol should produce the same VRF output γ, regardless of the collection of correctly formed shares used.

- **Robustness:** This guarantees the availability of computing the random function value on any plaintext in the presence of an active adversary, i.e., if $\mathrm{Combine}$ does not abort, its output passes the verification, even if the adversary inputs malformed values to the combination.

- **Strong pseudorandomness:** This guarantees that even if at most $t - 1$ participants are malicious, on input of the plaintext α, they are not able to distinguish between an honest output y of the DVRF protocol from a uniformly random value in the output range; this holds even when the malicious parties are involved in multiple executions on inputs $\alpha_1, \alpha_2, \ldots$ that are not α. During these executions, the malicious parties observe not only the DVRF outputs y_i on inputs of α_i but also the partial evaluations.

3.4 Protocol Execution

Our construction aims to aggregate the individual NIZK proofs used in DDH-DVRF (see Section 2.9.1) [33] in a way that it can be verified at once. To that end, we modified the NIZK proof system (see Section 2.7.3) by adding a t-out-of-n threshold structure over it. In other words, our approach needs at least t participants to generate a verifiable random value, and the same parties must be involved in generating the Discrete Logarithm Equality (DLEq) proof jointly.

We build a DVRF with compact proofs using the notation and setup of DDH-DVRF (Section 2.9.1), and also assume that we have a set $\mathcal{P}$ of n participants, i.e., $\mathcal{P} = \{P_1, \ldots, P_n\}$. Additionally, assume that $H_3 : \{0,1\}^* \to \{0,1\}^\Lambda$ is another hash function, where Λ denotes the desired output length. Let $\mathcal{V}^{\mathrm{DVRFwCP}}$ be defined via the following six algorithms.

$\mathrm{DistKeyGen}$: On input of the public parameters pp, the participants $P_1, \ldots, P_n$ jointly run the protocol $\mathrm{Secure\text{-}DKG}$ as in Section 2.6 to obtain their partial secret keys $x_1, \ldots, x_n \in \mathbb{Z}_q$ of the shared secret $\mathbf{x} \in \mathbb{Z}_q$, and the group public key

$$y = \prod_{i \in \mathcal{S}} (g^{x_i})^{\lambda_i} = g^{\sum\limits_{i \in \mathcal{S}} x_i \lambda_i} = g^{\mathbf{x}}$$

where $\mathcal{Q}$ is the set of n qualified participants, $\mathcal{S} \subset \mathcal{Q}$ is a subset of at least t qualified participants, and λ_i's are the appropriate Lagrange coefficients.

$\mathrm{PartialEval}$: Each $P_i \in \mathcal{Q}$ performs the following actions.

1. Compute $h \leftarrow H_1(\alpha) \in \mathbb{G} - \{1\}$.
2. Compute partial evaluation $\gamma_i = h^{x_i} \in \mathbb{G}$.
3. Compute partial evaluation proof $\pi_i \leftarrow \mathrm{ProveEq}_{H_2}\left(g, h, \mathrm{g}^{x_i}, \gamma_i; x_i\right)$.
4. Broadcast (γ_i, π_i).

$\mathrm{EvalCombine}$: Each $P_i \in \mathcal{Q}$ performs the following actions.

1. Upon receiving (γ_j, π_j) from P_j, check $\mathrm{VerifyEq}_{H_2}\left(g, h, \mathrm{g}^{x_j}, \gamma_j, \pi_j\right) \stackrel{?}{=} 1$.
2. If the checks do not pass for at least t partial proofs, abort. Otherwise, define $\mathcal{I}$ as the set of indices of the participants whose partial evaluation proofs are valid.
3. Perform Lagrange interpolation to reconstruct $\gamma = h^{\mathbf{x}} = \prod\limits_{j \in \mathcal{I}} \gamma_j^{\lambda_j}$, where λ_j is the appropriate Lagrange coefficient with respect to P_j and $\mathcal{I}$ is the set of indices of at least t received valid γ_j's.

$\mathrm{PartialProofGen}$: Each $P_i \in \mathcal{Q}$ performs the following actions.

1. Participate in the Augmented Secure-DKG protocol from Section 3.2 amongst the members of $\mathcal{Q}$ using the elements $g, h \in \mathbb{G}$ to generate a shared secret nonce $\mathbf{k}$. At the end of this step, each P_i obtains a partial ephemeral secret k_i, public values $g^{\mathbf{k}}$ and $h^{\mathbf{k}}$.
2. Compute the joint challenge, i.e., $c \leftarrow H_2(g, h, y, \gamma, g^{\mathbf{k}}, h^{\mathbf{k}})$.
3. After obtaining the joint challenge c, compute $s_i \leftarrow k_i - cx_i \mod q$.
4. Broadcast the partial proof $\Pi_i = (c, s_i)$.

$\mathrm{ProofCombine}$:

1. Upon receiving partial proof $\Pi_j = (c, s_j)$ from P_j, P_i verifies them by checking whether $g^{k_j} \stackrel{?}{=} y_j^c \cdot g^{s_j}$ and $h^{k_j} \stackrel{?}{=} \gamma_j^c \cdot h^{s_j}$.
2. If the checks do not pass for at least t partial proofs, abort. Otherwise, define $\mathcal{S}$ as the set of indices of the participants whose partial proofs are valid.
3. Perform Lagrange interpolation to obtain the aggregated proof, i.e., $\mathbf{s} = \sum_{i \in \mathcal{S}} s_i \lambda_i \pmod q$, where λ_i's are the appropriate Lagrange coefficients.
4. Output the VRF value $\beta \leftarrow H_3(\gamma)$ and the proof as $\Pi = (\gamma, c, \mathbf{s})$.

Verify: Given the group public key y, the proof $\Pi = (\gamma, c, \mathbf{s})$, the VRF value β and the plaintext α, perform verification as follows:

1. Compute $u = y^c \cdot g^{\mathbf{s}}$.
2. Given α, obtain $h = H_1(\alpha)$, compute $v = \gamma^c \cdot h^{\mathbf{s}}$.
3. Check if $c \stackrel{?}{=} H_2(g, h, y, \gamma, u, v)$ and $\beta \stackrel{?}{=} H_3(\gamma)$. If both are satisfied, output 1; otherwise, output 0.

Remark 3.2. In the first step of the $\mathrm{ProofCombine}$ algorithm, each P_i verifies the partial proof s_j to ensure that partial proofs to be aggregated are valid. Notice that the values g^{k_j}, h^{k_j} and γ_j's are known to P_i from the first step of the $\mathrm{PartialProofGen}$ algorithm.

Remark 3.3. The Secure-DKG in our construction can be replaced with any black box distributed key generation protocol provided there's an augmented version that would share auxiliary commitments as in Augmented Secure-DKG. This offers more versatility to our construction and leaves room for improvement in the interaction phase.

The threshold structure is layered over the DLEQ proof system in two stages. In $\mathrm{EvalCombine}$, Lagrange interpolation in the exponent reconstructs the single group element $\gamma = h^{\mathbf{x}}$ from the partial evaluations $\gamma_j = h^{x_j}$. In $\mathrm{ProofCombine}$, the same interpolation applied to the partial responses $s_j = k_j - c \cdot x_j$ gathers what would have been t separate DLEQ proofs into the single aggregated response $\mathbf{s} = \sum_{i \in \mathcal{S}} s_i \lambda_i$. The result is a proof $\Pi = (\gamma, c, \mathbf{s})$ of constant size, verified by a single DLEQ check rather than t of them.

3.5 Security

DVRFwCP satisfies the correctness property because of the following.

$$\begin{aligned} u = y^c \cdot g^{\mathbf{s}} = (g^{\mathbf{x}})^c \cdot g^{\sum\limits_{i \in \mathcal{S}} s_i \lambda_i} = g^{c\mathbf{x}} \cdot g^{\sum\limits_{i \in \mathcal{S}} (k_i - c x_i) \lambda_i} \\ = g^{c\mathbf{x}} \cdot g^{\sum\limits_{i \in \mathcal{S}} k_i \lambda_i - c \sum\limits_{i \in \mathcal{S}} x_i \lambda_i} = g^{c\mathbf{x}} \cdot g^{\mathbf{k} - c\mathbf{x}} = g^{\mathbf{k}} \end{aligned} \tag{3.3}$$

and

$$\begin{aligned} v = \gamma^c \cdot h^{\mathbf{s}} = (h^{\mathbf{x}})^c \cdot h^{\sum\limits_{i \in \mathcal{S}} s_i \lambda_i} = h^{c\mathbf{x}} \cdot h^{\sum\limits_{i \in \mathcal{S}} (k_i - c x_i) \lambda_i} \\ = h^{c\mathbf{x}} \cdot h^{\sum\limits_{i \in \mathcal{S}} k_i \lambda_i - c \sum\limits_{i \in \mathcal{S}} x_i \lambda_i} = h^{c\mathbf{x}} \cdot h^{\mathbf{k} - c\mathbf{x}} = h^{\mathbf{k}} \end{aligned} \tag{3.4}$$

and from Equation (3.3) and Equation (3.4), we have

$$c = H_2\left(g, h, y, \gamma, u, v\right) = H_2\left(g, h, y, \gamma, g^{\mathbf{k}}, h^{\mathbf{k}}\right) .$$

The properties of Secure-DKG [34] immediately imply consistency. Uniqueness follows from the fact that the output is consistent and the hash functions are based on random oracles. We argue that the DVRFwCP satisfies robustness and strong pseudorandomness, as the proposed algorithm is a modified version of the DDH-DVRF proposed in [33]; therefore, the security is reduced to the security of DDH-DVRF if the underlying distributed key generation algorithm is secure.

3.6 Comparison

In Table 3.1, we compare our construction with the DVRF schemes proposed in [33]. This comparison includes the number of operations required in each phase of the schemes, the number of rounds of interaction, and the proof size. In Table 3.2, we give a rough gas cost estimation for the verification and the proof size of the schemes compared in Table 3.1, assuming the EIP-2537: Precompile for BLS12-381 curve operations [53].

Table 3.1: Comparison of Operations of DDH-DVRF, GLOW-DVRF and DVRFwCP.

	DDH-DVRF	GLOW-DVRF	DVRFwCP
DistKeyGen	2 poly. in $\mathbb{Z}_q$ $4t+3$ exp. in $\mathbb{G}$ $n+3t-2$ mul. in $\mathbb{G}$ $n-1$ add. in $\mathbb{Z}_q$	2 poly. in $\mathbb{Z}_q$ $4t+3$ exp. in $\mathbb{G}$ $n+3t-2$ mul. in $\mathbb{G}$ $n-1$ add. in $\mathbb{Z}_q$ $2n$ pairings	2 poly. in $\mathbb{Z}_q$ $4t+3$ exp. in $\mathbb{G}$ $n+3t-2$ mul. in $\mathbb{G}$ $n-1$ add. in $\mathbb{Z}_q$
PartialEval	1 hash to $\mathbb{G}$ 3 exp. in $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$ 1 add. in $\mathbb{Z}_q$ 1 mul. in $\mathbb{Z}_q$	1 hash to $\mathbb{G}_1$ 3 exp. in $\mathbb{G}_1$ 1 hash to $\mathbb{Z}_q$ 1 add. in $\mathbb{Z}_q$ 1 mul. in $\mathbb{Z}_q$	1 hash to $\mathbb{G}$ 3 exp. in $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$ 1 mul. in $\mathbb{Z}_q$ 1 add. in $\mathbb{Z}_q$
EvalCombine	$5t$ exp. in $\mathbb{G}$ $3t-1$ mul. in $\mathbb{G}$ t hashes to $\mathbb{Z}_q$	$5t$ exp. in $\mathbb{G}_1$ $3t-1$ mul. in $\mathbb{G}_1$ t hashes to $\{0,1\}^\Lambda$	$5t$ exp. in $\mathbb{G}$ $3t-1$ mul. in $\mathbb{G}$ 1 Hash to $\mathbb{Z}_q$
PartialProofGen[b]	–	–	2 poly. in $\mathbb{Z}_q$ $6t+4$ exp. in $\mathbb{G}$ $2n+4t-2$ mul. in $\mathbb{G}$ n add. in $\mathbb{Z}_q$ 1 mul. in $\mathbb{Z}_q$ 1 hash to $\mathbb{Z}_q$
ProofCombine[b]	–	–	$6t$ exp. in $\mathbb{G}$ $2t$ mul. in $\mathbb{G}$ $t-1$ add. in $\mathbb{Z}_q$ t mul. in $\mathbb{Z}_q$ 1 Hash to $\{0,1\}^\Lambda$
Verify	$5t$ exp. in $\mathbb{G}$ $3t-1$ mul. in $\mathbb{G}$ 1 hash to $\mathbb{G}$ t hashes to $\mathbb{Z}_q$	2 pairings 1 hash to $\mathbb{G}_1$ 1 hash to $\{0,1\}^\Lambda$	4 exp. in $\mathbb{G}$ 2 mul. in $\mathbb{G}$ 1 hash to $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$ 1 hash to $\{0,1\}^\Lambda$
Interactions[c]	1 round	1 round	2 rounds
Proof size	t $\mathbb{G}$ elements	1 $\mathbb{G}_1$ element 1 Λ-bit element	1 $\mathbb{G}$ element 2 $\mathbb{Z}_q$ elements

(a) We use the following abbreviations: "poly." for "polynomial evaluations", "exp" for "exponentiations", "mul." for "multiplications", and "add." for "additions".
(b) These algorithms are not present in the corresponding DVRFs.
(c) The one-time DistKeyGen is not included.

In the DistKeyGen phase, our scheme requires the same number of operations as in DDH-DVRF, which requires less than GLOW-DVRF.

In the PartialEval and EvalCombine phases, our scheme requires the same number of operations as in DDH-DVRF and GLOW-DVRF.

Our scheme has two additional algorithms, i.e., PartialProofGen and ProofCombine. In order to create a shared challenge in PartialProofGen, our scheme requires another round of

communication amongst the n qualified parties, which makes the proof generation no longer non-interactive. All the operations in these phases can be counted as extra operations that do not exist in the other schemes in comparison. However, for example, in blockchains, these algorithms are handled off-chain; hence, no gas cost is accrued.

In the Verify phase, DVRFwCP requires fewer operations than both DDH-DVRF and GLOW-DVRF. It should be emphasized that a noteworthy achievement of our construction is that in the Verify phase, our algorithm does not use bilinear pairings, which are known to be slow; also, DVRFwCP has constant time verification as opposed to the linear time verification in the quantity of the threshold of DDH-DVRF. This is a significant improvement, especially in the context of blockchains. For example, when a smart contract acts as a public verifier, each node should run the same code for verification. Since our construction requires a constant number of group operations, the gas cost needed by the verifier smart contract would be much lower than the verification gas cost required by DDH-DVRF and GLOW-DVRF, as they require either much more expensive operations like bilinear pairings or $O(t)$ group operations, where t is the threshold number of participants, as seen in Table 3.2.

Another key achievement of our proposed scheme is that the proof size is constant (as in GLOW-DVRF), i.e., it consists of one group element and two $\mathbb{Z}_q$ elements. This is also a substantial improvement, especially for the blockchain use cases. Note that, other than the computational complexity of the verification algorithm, the size of the data (call data) that needs to be sent to the verifier smart contract also increases the gas cost. For example, in Ethereum, one should pay 16 gas per Byte of call data in addition to the verification cost of a VRF proof [5]. According to [53] (which uses additive group notation), a multiplication in $\mathbb{G}_1$ costs 500 gas, and an exponentiation in $\mathbb{G}_1$ costs $12{,}000$ gas; hence, ignoring the hash functions, the Verify of DVRFwCP costs $49{,}000$ gas. The exponentiation cost in $\mathbb{G}_2$ is $45{,}000$ gas and a pairing check operation between members of $\mathbb{G}_1$ and $\mathbb{G}_2$ is $43{,}000 \cdot k + 65{,}000$, where k is the number of pairs; this makes GLOW-DVRF verification cost about quadruple that of DVRFwCP. For more details, we refer the reader to Table 3.1 and Table 3.2.

Table 3.2: Gas Cost Estimation Comparison for the BLS12-381 Curve.

Gas cost	DDH-DVRF	GLOW-DVRF	DVRFwCP
Verification cost	$\approx 70{,}000 \cdot t$	$\approx 200{,}000$	$\approx 50{,}000$
Calldata cost	$768 \cdot t$	$1{,}280$	$1{,}792$
Total cost	$\approx 70{,}768 \cdot t$	$\approx 201{,}280$	$\approx 51{,}792$

(a) We assume that the subgroup $\mathbb{G}_1$ of the BLS12-381 curve is used as the cyclic group for both DDH-DVRF and DVRFwCP.

(b) We estimate the gas cost only for pairings, exponentiation, and multiplications in $\mathbb{G} = \mathbb{G}_1$.

(c) We take $|\mathbb{G}_1| = 48$ bytes and $|\mathbb{Z}_q| = 32$ bytes.

These figures are theoretical estimates based on the per-operation gas costs of EIP-2537. The resulting comparison, approximately $51{,}792$ gas for DVRFwCP against approximately $201{,}280$ gas for GLOW-DVRF, a factor of roughly four, is validated empirically against a

Solidity implementation on the Sepolia testnet in Chapter 6, where the contrast between the theoretical and measured costs is examined in detail.

DVRFwCP thus addresses two facets of the trilemma identified in Section 3.1: its proof is of constant size, and its verification uses no pairings. The cost, as noted above, is the second interactive round. The $\mathrm{PartialProofGen}$ phase runs an Augmented Secure-DKG among all n qualified parties for every evaluation, and this incurs $O(n^2 t)$ communication in total. Although this round is performed off-chain and therefore incurs no gas cost, it remains a latency and bandwidth bottleneck that makes DVRFwCP unsuitable for latency-sensitive deployments. Reducing the impact of this interaction, without sacrificing the compact pairing-free proof established here, is the problem addressed in Chapter 4.

CHAPTER 4

ICY-DVRF: ELIMINATING THE INTERACTION BOTTLENECK

4.1 FROST as the Key Insight

In Chapter 3, we established that DVRFwCP achieves a favorable position on the proof-size and pairing axes: its proof is a single group element with two scalars, and its verification uses no bilinear pairing. The cost was the second interactive round. To assemble the joint challenge c, the n qualified parties must run an Augmented Secure-DKG for every evaluation, generating a fresh shared ephemeral secret $\mathbf{k}$ each time. This round carries $O(n^2t)$ communication in total and cannot be precomputed, since the shared secret it produces is consumed by a single proof.

The observation that forms the groundwork for this chapter is that the cryptographic role of the Augmented Secure-DKG, namely producing a shared ephemeral commitment in two bases that no individual party can predict or control, is precisely the role that the preprocessing and nonce-binding mechanism of FROST was designed to fill in the threshold-signature setting. We propose Icy-DVRF, a DVRF system similar to DVRFwCP [2], that leverages the preprocessing and nonce-binding features of FROST [42] to reduce the number of interactions required in DVRFwCP.

Our construction initially employs the FROST distributed key generation algorithm, rather than the Secure-DKG [34] used in DVRFwCP. We replace the Augmented Secure-DKG protocol used in DVRFwCP with a nonce-generation technique that incorporates elements of FROST's binding technique.

Concretely, recall from Section 2.8 that in FROST, each participant publishes a batch of single-use commitment pairs $(D_{i,j}, E_{i,j}) = (g^{d_{i,j}}, g^{e_{i,j}})$ ahead of time, and then at signing time, these commitments are bound to the request through a deterministic value $\rho_i = H_2(i, \alpha, B)$ derived from the message and the ordered list of signers. The shared commitment $R = \prod_\ell D_\ell E_\ell^{\rho_\ell}$ is then computed locally by every party with no interaction. Icy-DVRF adapts this mechanism to the DVRF setting: the preprocessed nonces play the role of the ephemeral secret $\mathbf{k}$ that DVRFwCP regenerated interactively, and the binding value ρ_i

removes the need for the agreement round. The key insight is therefore that replacing the Augmented Secure-DKG with FROST-style preprocessed nonces eliminates the interactive proof round entirely, reducing per-evaluation communication from $O(n^2t)$ to $O(t)$, while preserving the constant-size, pairing-free proof established in Chapter 3.

4.2 Protocol Definition and Setup

We first give the formal syntax of Icy-DVRF, then fix the notation and the distributed key generation procedure used by the construction.

Definition 4.1 (Icy-Distributed Verifiable Random Function)**.** A t-out-of-n Icy-DVRF $\mathcal{V}^{\mathrm{Icy-DVRF}}$ consists of six algorithms, i.e., $\mathcal{V}^{\mathrm{Icy-DVRF}} = (\mathrm{DistKeyGen}, \mathrm{Preprocess}, \mathrm{PartialEval}, \mathrm{PartialProofGen}, \mathrm{Combine}, \mathrm{Verify})$, described below.

- $\mathrm{DistKeyGen}\left(1^\lambda, t, n\right) \rightarrow \left(\{(s_i, Y_i)\}_{i\in[n]}, Y\right)$ takes a security parameter λ, the threshold parameter t and the number of participating nodes n as inputs, and outputs a list of partial secret-public key pairs $\{(s_i, Y_i)\}_{i\in[n]}$ and a group public key Y.
- $\mathrm{Preprocess}\left(\Omega\right) \rightarrow L_i = \{(D_{i,j}, E_{i,j})\}_{j=1}^{\Omega}$ takes the number of commitment shares to be generated Ω, outputs a list L_i of commitment pairs of the form $(D_{i,j}, E_{i,j})$ for $1 \leq j \leq \Omega$.
- $\mathrm{PartialEval}\left(\alpha, j, B, s_i, Y_i\right) \rightarrow \left(\gamma_i, \pi_i, \bar{D}_i, \bar{E}_i\right)$ takes a plaintext α, a single-use index j, the ordered prover list B, a partial secret and verification keys (s_i, Y_i) as inputs, and outputs a partial evaluation γ_i, a partial evaluation NIZK proof π_i, and auxiliary values $(\bar{D}_i, \bar{E}_i)$.
- $\mathrm{PartialProofGen}\left(\alpha, B, S, s_i, Y_i, Y, \left(D_i, E_i, \bar{D}_i, \bar{E}_i\right)_{i\in S}\right) \rightarrow (z_i, c)$ takes a plaintext α, the ordered prover list B, the prover set S, a partial secret/public key pair (s_i, Y_i), public key Y, a commitment list of all users $(D_i, E_i)_{i\in S}$ and auxiliary values $(\bar{D}_i, \bar{E}_i)_{i\in S}$ as inputs and outputs a partial NIZK proof (z_i, c).
- $\mathrm{Combine}\left(\alpha, (z_i, c)_{i\in S}, Y, \left(D_i, E_i, \bar{D}_i, \bar{E}_i\right)_{i\in S}\right) \rightarrow (\beta, (\gamma, c, z))$ takes a plaintext α, a list of partial NIZK proofs $(z_i, c)_{i\in S}$, the public key Y, a commitment list of all users $(D_i, E_i)_{i\in S}$ and auxiliary values $(\bar{D}_i, \bar{E}_i)_{i\in S}$, outputs either a DVRF value β with a combined NIZK proof (γ, c, z) or aborts.
- $\mathrm{Verify}\left(\alpha, Y, \beta, (\gamma, c, z)\right) \rightarrow 0/1$ takes a plaintext α, a public key Y, a DVRF value β, a NIZK proof (γ, c, z) as inputs, it outputs $0/1$.

Icy-DVRF is required to satisfy the same DVRF security properties: correctness, consistency, public verifiability, uniqueness, and pseudorandomness as in Section 2.9. Implementations must ensure that participants are identifiable within the group so that misbehavior can be reported, communicate via a secure, authenticated channel, and validate messages before use.

4.2.1 Domain Separation of Hash Functions

To ensure cryptographic hygiene and prevent the reuse of one random-oracle output in the role of another, the random oracles of Icy-DVRF are strictly domain-separated, except for the map $H_1 : \{0,1\}^* \to \mathbb{G}$, which is a hash-to-group function modeled as a random oracle onto $\mathbb{G}$. Since H_1 is not derived from $\mathcal{H}$, it carries no tag and its queries are separated from the others by construction. The shorthands H_2 and H_3 are used for typographical clarity; each is instantiated through a single cryptographically secure function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^*$ (mapped into the relevant range) under an explicit Domain Separation Tag (DST). H_2 is queried in three structurally distinct roles, each of which receives its own tag so that their query spaces are disjoint:

- $\mathbf{H_2^{pok}}(\mathbf{i}, \mathbf{\Phi}, \phi_{\mathbf{i},\mathbf{0}}, \mathbf{R_i})$ is the Schnorr proof-of-knowledge challenge $c_i \in \mathbb{Z}_q$ in DistKeyGen:
$$c_i = \mathcal{H}(\texttt{"ICY-POK-CHL"} \parallel i \parallel \Phi \parallel \phi_{i,0} \parallel R_i).$$
- $\mathbf{H_2^{bind}}(\ell, \alpha, \mathbf{B})$ is the per-prover FROST-style binding value $\rho_\ell \in \mathbb{Z}_q$:
$$\rho_\ell = \mathcal{H}(\texttt{"ICY-FROST-BND"} \parallel \ell \parallel \alpha \parallel B).$$
- $\mathbf{H_2^{chl}}(\mathbf{g}, \mathbf{h}, \mathbf{Y}, \gamma, \mathbf{R}, \bar{\mathbf{R}}, \alpha)$ is the aggregate T-DLEQ proof challenge $c \in \mathbb{Z}_q$:
$$c = \mathcal{H}(\texttt{"ICY-DLEQ-CHL"} \parallel g \parallel h \parallel Y \parallel \gamma \parallel R \parallel \bar{R} \parallel \alpha).$$
- $\mathbf{H_3}(\gamma)$ is the final pseudorandom output $\beta \in \{0,1\}^\Lambda$:
$$\beta = \mathcal{H}(\texttt{"ICY-OUTPUT"} \parallel \gamma).$$

The three H_2 roles share the notation H_2 but never the domain: the leading DST makes the binding, challenge, and proof-of-knowledge query spaces pairwise disjoint, so no binding value can ever coincide with a proof challenge. This is the separation we rely on in Section 4.5.4, where the knowledge-soundness reduction programs the binding entries $H_2(\ell, \alpha, B)$ and the challenge entry $H_2(g, h, Y, \gamma, R, \bar{R}, \alpha)$ independently and counts them as distinct H_2 entries. The interactive variant of T-DLEQ (Section 4.5.3) additionally uses three oracles, each carrying their own DST under the same convention: $H_\rho : \{0,1\}^* \to \mathbb{Z}_q$, which on input (α, B) returns the one-time binding challenge; $H_\pi : \mathbb{G}^6 \to \mathbb{Z}_q$, which on input $(g, h, g^s, h^s, g^r, h^r)$ returns the challenge of the per-party DLEQ proof; and $H_4 : \{0,1\}^* \to \mathbb{Z}_q$, which on input (i, Φ) binds participant i to the session value Φ.

By fixing these definitions globally, the absolute separation of the random oracle query spaces is formally guaranteed without modifying the standardized shorthand notations (H_1, H_2, etc.) in the subsequent protocol descriptions, theorems, and algorithms.

4.2.2 Notation and Distributed Key Generation

In the interest of presenting a self-contained algorithm, the notation and setup fundamental to the construction are detailed below. Let $(\mathbb{G}, g, q)$ be a multiplicative group where the DDH assumption holds. Let $H_1 : \{0,1\}^* \to \mathbb{G}$, $H_2 : \{0,1\}^* \to \mathbb{Z}_q$ and $H_3 : \{0,1\}^* \to \{0,1\}^\Lambda$ be hash functions, where Λ denotes the desired output length. Additionally, assume that we have a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of participants who communicate using the secure implementation practices described earlier. Let $\mathcal{V}^{\mathrm{Icy-DVRF}}$ be defined using the following six algorithms.

DistKeyGen:

Round 1

1. Every participant P_i samples t random values $\left(a_{i,0}, \ldots, a_{i,(t-1)}\right) \xleftarrow{\$} \mathbb{Z}_q$, and uses these values as coefficients to define a degree $t-1$ polynomial

$$f_i(x) = \sum_{j=0}^{t-1} a_{i,j}\, x^j \ .$$

2. Every participant P_i computes a proof of knowledge of their secret $a_{i,0}$ by calculating $\sigma_i = (R_i, \mu_i)$, where $k \xleftarrow{\$} \mathbb{Z}_q$, and $R_i = g^k$, $c_i = H_2\left(i, \Phi, g^{a_{i,0}}, R_i\right)$, $\mu_i = k - a_{i,0} \cdot c_i$, where Φ is a context string to prevent replay attacks.

3. Every participant P_i computes public commitments $\vec{C}_i = \left\langle \phi_{i,0}, \ldots, \phi_{i,(t-1)} \right\rangle$, where $\phi_{i,j} = g^{a_{i,j}}, 0 \le j \le t-1$. In particular, note that $\phi_{i,0} = g^{a_{i,0}}$.

4. Every P_i broadcasts $\vec{C}_i, \sigma_i$ to all other participants, or to the public.

5. Upon receiving $\vec{C}_\ell, \sigma_\ell$ from participants $1 \le \ell \le n, \ell \ne i$, each participant P_i verifies $\sigma_\ell = (R_\ell, \mu_\ell)$, aborting on failure, by checking $R_\ell \stackrel{?}{=} g^{\mu_\ell} \cdot \phi_{\ell,0}^{c_\ell}$, where $c_\ell = H_2\left(\ell, \Phi, \phi_{\ell,0}, R_\ell\right)$. Upon success, participants delete $\{\sigma_\ell : 1 \le \ell \le n\}$.

Round 2

1. Each P_i securely sends to each other participant P_ℓ a secret share $(\ell, f_i(\ell))$, deleting f_i and each share afterward except for $(i, f_i(i))$, which they keep for themselves.

2. Each P_i verifies their shares by calculating

$$g^{f_\ell(i)} \stackrel{?}{=} \prod_{k=0}^{t-1} \left(\phi_{\ell,k}\right)^{i^k \bmod q}$$

   aborting if the check fails.

3. Each P_i calculates their long-lived private signing share by computing $s_i = \sum_{\ell=1}^{n} f_\ell(i)$, stores s_i securely, and deletes each $f_\ell(i)$.

4. Each P_i calculates their public verification share $Y_i = g^{s_i}$, and the group's public key $Y = \prod_{j=1}^{n} \phi_{j,0}$. Any participant can compute the public verification share of any other participant by calculating

$$Y_i = \prod_{j=1}^{n} \prod_{k=0}^{t-1} (\phi_{j,k})^{i^k \bmod q} \ .$$

4.3 The Preprocess and Binding Mechanisms

The Preprocess phase precomputes the randomness that DVRFwCP generated interactively. Each participant produces a batch of Ω single-use nonce pairs and publishes their commitments in advance without coordination. These commitments are the material from which the shared ephemeral commitment is later assembled locally, and non-interactively, at evaluation time.

Preprocess:

Each participant P_i for $i \in \{1, \ldots, n\}$ performs the steps below. Let j be a counter for a specific nonce/commitment share pair, and Ω be the number of pairs generated at a time, such that Ω is the number of tries the combiner performs before performing another preprocessing step.

1. Create an empty list L_i. Then, for $1 \leq j \leq \Omega$, perform the following:
   a) Sample single-use nonces $(d_{i,j}, e_{i,j}) \xleftarrow{\$} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$.
   b) Derive commitment shares $(D_{i,j}, E_{i,j}) = \left(g^{d_{i,j}}, g^{e_{i,j}}\right)$.
   c) Append $(D_{i,j}, E_{i,j})$ to L_i. Store these values for later use.
2. Publish (i, L_i) to a predetermined location, as specified by the implementation.

The binding mechanism is what makes these precomputed nonces safe to use. At evaluation time, rather than letting each party contribute its raw commitment $(D_{i,j}, E_{i,j})$, the protocol derives a deterministic binding value

$$\rho_\ell = H_2(\ell, \alpha, B)$$

for every prover ℓ in the session, where α is the input and B is the ordered list of participating provers. Each party's effective ephemeral secret becomes $k_i = d_i + \rho_i \cdot e_i$, and the shared commitment is the locally computable product $R = \prod_{\ell \in S} D_\ell (E_\ell)^{\rho_\ell} = g^k$ with $k = \sum_{\ell \in S} k_\ell$. Because ρ_ℓ is bound jointly to the input α and the exact set and ordering of provers B, an adversary cannot mix and match commitment shares across concurrent sessions: any change to the input or the participant list yields entirely different binding values, and hence a different

shared commitment. This is precisely the FROST defense against the concurrent-execution attacks of Drijvers et al. [27]; its formal consequence for knowledge-soundness is analyzed in Section 4.5.

4.4 Protocol Execution

We now describe the four online algorithms. The structural difference from DVRFwCP is concentrated in $\mathrm{PartialProofGen}$: DVRFwCP invokes the Augmented Secure-DKG to produce the shared challenge, whereas Icy-DVRF assembles it locally from the preprocessed, binding commitments. The auxiliary values $\bar{D}_i = h^{d_{i,j}}$ and $\bar{E}_i = h^{e_{i,j}}$ produced in $\mathrm{PartialEval}$ are the second-base counterparts of the commitments, and play the role that the h-base commitments of the Augmented Secure-DKG played in Chapter 3.

$\mathrm{PartialEval}$:

Let C be the combiner (who might be chosen amongst the participants). Let S be the set of θ participants selected for the generation of the DVRF value, where $t \leq \theta \leq n$, and let Y be the group public key set during the DistKeyGen phase.

1. C sends (α, j, B) to $P_i,\ i \in S$, where B is the ordered list of parties in S and j is the index of the single-use commitments from the Preprocess stage to be used.
2. Each P_i checks if B is valid and if α is the message they want to use. If the checks fail, abort.
3. Each P_i performs the following.
   (a) Compute $h \leftarrow H_1(\alpha) \in \mathbb{G} \setminus \{1\}$.
   (b) Compute partial evaluation $\gamma_i = h^{s_i} \in \mathbb{G}$.
   (c) Compute partial evaluation proof $\pi_i \leftarrow \mathrm{ProveEq}_{H_2}(g, h, Y_i, \gamma_i; s_i)$.
   (d) Compute $\bar{D}_i = h^{d_{i,j}}$ and $\bar{E}_i = h^{e_{i,j}}$.
   (e) Outputs $(\gamma_i, \pi_i, \bar{D}_i, \bar{E}_i)$.

$\mathrm{PartialProofGen}$:

Now that each participant P_i published $\big(\gamma_i, \pi_i, D_i, E_i, \bar{D}_i, \bar{E}_i\big)$, each P_i produces their partial DVRF proof, using the following procedure.

1. Compute the set of binding values $\rho_\ell = H_2(\ell, \alpha, B)$ for all $\ell \in S$.
2. Compute $k_i = d_i + \rho_i \cdot e_i$.
3. Compute the (public) group commitments

$$R = \prod_{\ell \in S} D_\ell \cdot (E_\ell)^{\rho_\ell} = g^k, \qquad \bar{R} = \prod_{\ell \in S} \bar{D}_\ell \cdot \left(\bar{E}_\ell\right)^{\rho_\ell} = h^k,$$

where $k = \sum_{\ell \in S} k_\ell$, unknown to individual parties.

4. Compute $\gamma = \prod_{\ell \in S} \gamma_\ell^{\lambda_\ell}$, where λ_ℓ is the ℓ^{th} Lagrange coefficient in S.
5. Compute the group challenge $c = H_2\left(g, h, Y, \gamma, R, \bar{R}, \alpha\right)$.
6. Computes their response using their long-lived partial secret key s_i by computing the partial proof $z_i = k_i + \lambda_i \cdot s_i \cdot c = d_i + \rho_i \cdot e_i + \lambda_i \cdot s_i \cdot c$, which is in additive form, where λ_i is the i^{th} Lagrange coefficient in S.
7. Securely deletes d_i, e_i from their local storage, and broadcasts (z_i, c).

Combine:

Upon receiving (z_ℓ, c) from P_ℓ for $\ell \in S$, the combiner C performs the following actions.

1. Computes the following.[1]
   (a) $\rho_\ell = H_2(\ell, \alpha, B)$ for $\ell \in S$.
   (b) $R_\ell = D_\ell \cdot (E_\ell)^{\rho_\ell}$ for $\ell \in S$.
   (c) $\bar{R}_\ell = \bar{D}_\ell \cdot \left(\bar{E}_\ell\right)^{\rho_\ell}$ for $\ell \in S$.
   (d) The group commitment $R = \prod_{\ell \in S} R_\ell$.
   (e) The group commitment $\bar{R} = \prod_{\ell \in S} \bar{R}_\ell$.
   (f) $\gamma = \prod_{\ell \in \mathcal{S}} \gamma_\ell^{\lambda_\ell}$, where λ_ℓ is the ℓ^{th} Lagrange coefficient in S.
   (g) The group challenge $c = H_2\left(g, h, Y, \gamma, R, \bar{R}, \alpha\right)$.
2. Verifies the following.
   (a) Verifies that each c received from P_ℓ matches the c value the combiner computed in Step 1(g) above.
   (b) Verifies each partial evaluation proof by checking

   $$\text{VerifyEq}_{H_2}\left(g, h, Y_\ell, \gamma_\ell, \pi_\ell\right) \stackrel{?}{=} 1$$

   for each $(\gamma_\ell, \pi_\ell), \ell \in S$, where Y_ℓ is the partial public key of P_ℓ.
   (c) Verifies the validity of the response of $P_\ell, \ell \in S$, by checking

   $$g^{z_\ell} \stackrel{?}{=} R_\ell \cdot Y_\ell^{c \cdot \lambda_\ell}$$
   $$h^{z_\ell} \stackrel{?}{=} \bar{R}_\ell \cdot \gamma_\ell^{c \cdot \lambda_\ell}$$

   for each $z_\ell, \ell \in S$.

[1] This step can be skipped entirely if C is one of the participants.

If any of these checks fail for a participant, identify and report the misbehaving participant. At the end of this step, if at least t out of the θ members of S are not qualified, abort the protocol. Otherwise, continue.

3. Compute the group response $z = \sum_{\ell \in S} z_\ell$.
4. Output the VRF value $\beta \leftarrow H_3(\gamma)$ and the proof as $\pi = (\gamma, c, z)$.

Verify:

Given the group public key Y, the proof $\pi = (\gamma, c, z)$, the VRF value β and the message α, perform verification as follows:

1. Compute $T = g^z \cdot Y^{-c}$.
2. Compute $h = H_1(\alpha)$ and $\bar{T} = h^z \cdot \gamma^{-c}$.
3. Check if $c \stackrel{?}{=} H_2(g, h, Y, \gamma, T, \bar{T}, \alpha)$ and $\beta \stackrel{?}{=} H_3(\gamma)$. If both are satisfied, output 1; otherwise, output 0.

We want to emphasize two points. First, the entire $\mathrm{PartialProofGen}$ phase is non-interactive: every quantity a party needs, namely the binding values, the shared commitments R and $\bar{R}$, the challenge c, and its own response z_i, is computed locally from already-published data, with no agreement round. This eliminates the bottleneck of DVRFwCP. Second, the Verify algorithm is identical in form to that of DVRFwCP: it recomputes $T = g^z Y^{-c}$ and $\bar{T} = h^z \gamma^{-c}$ and checks a single hash. Since $z = k + cs$, where s is the reconstructed secret key and $Y = g^s$, we have $T = g^{k+cs} g^{-cs} = g^k = R$ and likewise $\bar{T} = h^k = \bar{R}$, so the recomputed challenge matches. The proof remains one group element and two scalars, verified without pairings, so the compactness of Chapter 3 is preserved.

4.5 Formal Security Analysis

The security of Icy-DVRF is established in a few steps. We first verify correctness and consistency together in Section 4.5.1, since both follow at once from the same recomputation the verifier performs. We address uniqueness and strong pseudorandomness in Section 4.5.2: The proofs for the interactive and non-interactive variants share the same structure except for the knowledge-soundness. For this reason we prove knowledge-soundness twice: once for the interactive variant, under the plain discrete logarithm assumption (Section 4.5.3), and once for the non-interactive variant used by Icy-DVRF, under the one-more discrete logarithm assumption (Section 4.5.4). We close with the tightness of these reductions and the resulting parameter recommendations in Section 4.5.5.

4.5.1 Correctness and Consistency

During the DistKeyGen phase, let $F_1(x) = \sum_{j=1}^{n} f_j(x)$ be the polynomial defining the secret sharing of the private signing key s, where $s_i = F_1(i)$ and the secret key is $s = F_1(0)$. Similarly, during the PartialEval phase, let S be the set of θ participants, with $t \leq \theta \leq n$ selected for proof generation, each using a pair of nonces (d_i, e_i) to define a degree $\theta - 1$ polynomial $F_2(x)$, interpolating the points $\left(i, \frac{1}{\lambda_{i,S}} \left(d_i + e_i \cdot H_2(\ell, \alpha, B)\right)\right)$, so that $F_2(0) = \sum_{i \in S} d_i + e_i \cdot \rho_i = \sum_{i \in S} k_i = k$.

Now let $F_3(x) = F_2(x) + c \cdot F_1(x)$, where $c = H_2\left(g, h, Y, \gamma, R, \bar{R}, \alpha\right)$. Then, $z_i = d_i + \rho_i \cdot e_i + \lambda_{i,S} \cdot s_i \cdot c = \lambda_{i,S} \left(F_2(i) + c \cdot F_1(i)\right) = \lambda_{i,S} F_3(i)$, hence $z = \sum_{i \in S} z_i$ is merely the Lagrange interpolation of $F_3(0) = k + c \cdot s$.

Since $R = \prod_{i \in S} D_i \cdot (E_i)^{\rho_i} = g^{\sum_{i \in S} d_i + \rho_i \cdot e_i} = g^k$ and $\bar{R} = \prod_{i \in S} \bar{D}_i \cdot \left(\bar{E}_i\right)^{\rho_i} = h^{\sum_{i \in S} d_i + \rho_i \cdot e_i} = h^k$ and $c = H_2\left(g, h, Y, \gamma, R, \bar{R}, \alpha\right)$, $\pi = (\gamma, c, z)$ is a correctly formed VRF proof with VRF value $\beta = H_3(\gamma)$, and Verify evaluates to 1, proving the correctness property of the Icy-DVRF protocol.

In the creation of the final VRF output $\beta = H_3\left(\gamma\right)$, once the DistKeyGen phase is complete, the secret share s_i of each participant $P_i \in \mathcal{P}$ is set, and once the messages α and the set S are fixed in PartialEval, $h = H_1\left(\alpha\right)$ is determined, hence each $\gamma_i = h^{s_i}$. Because the (intermediate) DVRF value γ is the (t, n)-threshold combination of $\{\gamma_i\}_{i \in S}$, it is consistently generated.

4.5.2 Uniqueness and Pseudorandomness

To prove that Icy-DVRF satisfies the uniqueness and pseudorandomness properties, we turn our attention to demonstrating that the Threshold Discrete Logarithm Equality proof system employed in the Icy-DVRF protocol satisfies the properties of a Non-Interactive Zero-Knowledge (NIZK) proof. First, we establish that the interactive variant of the system is zero-knowledge.

Theorem 4.2. *The Interactive Threshold Discrete Logarithm Equality (T-DLEQ-Int) proof system in Icy-DVRF is complete, honest-verifier zero-knowledge, and knowledge-sound under the discrete logarithm assumption in the random oracle model, with the distributed key generation analyzed under the Schnorr knowledge-of-exponent assumption. The concrete extraction loss is quantified in Theorem 4.5.*

Proof. Completeness immediately follows from the Correctness and Consistency properties. To show that it is Honest Verifier Zero-Knowledge, let $\mathbb{G}$ be a cyclic group of prime order q, let $g, h \in \mathbb{G}$, and consider the threshold statement proved by the t provers, namely that the group secret $s \in \mathbb{Z}_q$ satisfies $s = \log_g(Y) = \log_h(\gamma)$ where $Y, \gamma \in \mathbb{G}$.

The adversary corrupts $t - 1$ of the t provers. It therefore chooses the nonces of those parties

itself and can compute their partial responses without assistance, so the simulator cannot sample the aggregate transcript $(R, \bar{R}, c, z)$ freely. Its only freedom is the contribution of the single honest party P_t, and the aggregate is formed from that contribution together with the corrupted ones exactly as in a real execution.

`Sim` simulates P_t alone. Given the challenge c supplied by the honest verifier, it samples $z_t \xleftarrow{\$} \mathbb{Z}_q$ and sets

$$R_t = g^{z_t} \cdot Y_t^{-c} , \qquad \bar{R}_t = h^{z_t} \cdot \gamma_t^{-c} ,$$

then publishes $(R_t, \bar{R}_t)$ in place of the commitment pair of P_t and answers with z_t.

In a real execution z_t is uniformly distributed over $\mathbb{Z}_q$ and independent of c, because the nonce of P_t is, and given c and z_t the pair $(R_t, \bar{R}_t)$ is uniquely determined by the two verification relations. The contribution of P_t is therefore identically distributed in the real and the simulated execution, and the corrupted contributions are produced by the adversary in both, so the two views coincide.

The proof of the knowledge-soundness property of T-DLEQ-Int is a direct consequence of Theorem 4.5 in Section 4.5.3. □

We now turn to the non-interactive counterpart of T-DLEQ-Int, which is the variant originally presented in Section 4.2. Its security guarantees, which rest on the one-more discrete logarithm assumption rather than the plain discrete logarithm assumption, are established by the following theorem.

Theorem 4.3. *The Threshold Discrete Logarithm Equality (T-DLEQ) proof system is a non-interactive zero-knowledge argument of knowledge: it is complete, zero-knowledge, and knowledge-sound under the one-more discrete logarithm assumption, with the distributed key generation analyzed under the Schnorr knowledge-of-exponent assumption. The concrete extraction loss is quantified in Theorem 4.13.*

Proof. Completeness follows from the Correctness property of Section 4.5.1: for an honestly generated proof (γ, c, z) with $z = k + c \cdot s$, $R = g^k$, $Y = g^s$, and $\gamma = h^s$, the verifier recomputes $T = g^z Y^{-c} = g^k = R$ and $\bar{T} = h^z \gamma^{-c} = h^k = \bar{R}$, so the challenge check $c = H_2(g, h, Y, \gamma, T, \bar{T}, \alpha)$ succeeds and **Verify** outputs 1.

To show that the system is zero-knowledge, note that for each statement (g, h, Y, γ) with $h = H_1(\alpha)$, the simulator samples $\bar{c}, \bar{z} \xleftarrow{\$} \mathbb{Z}_q$, sets $R = g^{\bar{z}} Y^{-\bar{c}}$ and $\bar{R} = h^{\bar{z}} \gamma^{-\bar{c}}$, and programs $H_2(g, h, Y, \gamma, R, \bar{R}, \alpha) \leftarrow \bar{c}$. By Lemma 4.7 of Section 4.5.4, each simulated proof is distributed identically to an honest one whenever the programmed point was previously undefined. The probability that any of the at most Q programmed points collides with a prior H_2 entry is at most Q^2/q, as in the proof of Lemma 4.12 of Section 4.5.4. Hence,

the simulated transcripts are statistically indistinguishable from real ones, and T-DLEQ is zero-knowledge in the programmable random oracle model.

Knowledge-soundness requires the full reduction, and is proved in Theorem 4.13 of Section 4.5.4. We show that any PPT adversary producing a verifying, non-trivial T-DLEQ proof can be turned into a solver for the one-more discrete logarithm (OMDL) problem, with the distributed key generation handled by the Schnorr knowledge-of-exponent assumption (implied by DL in the AGM). Concretely,

$$\mathsf{Adv}^{\text{t-dleq}}(A) \leq \mathsf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_{\mathsf{koe}}) + 2\max\{\epsilon_{\mathsf{quo}}, \epsilon_{\mathsf{sub}}\} + \frac{Q^2}{q},$$

where $\mathcal{B}_{\mathsf{koe}}$ is the Schnorr-KoE adversary, $\epsilon_{\mathsf{quo}}, \epsilon_{\mathsf{sub}}$ are the shorthand notations for the two branch bounds of Theorem 4.13. The reduction is modular using the intermediate Bi-DLEQ assumption (Definition 4.6, Lemma 4.9), which confines all rewinding to a single forking step.

Thus, having demonstrated completeness, zero-knowledge, and knowledge-soundness, we conclude that T-DLEQ is a valid NIZK proof system. □

Having established that the T-DLEQ proof system within Icy-DVRF satisfies the definition of a Non-Interactive Zero-Knowledge (NIZK) proof, we now address the remaining security properties. Structurally, Icy-DVRF diverges from DDH-DVRF [33] only in its use of a threshold mechanism to be able to aggregate the individual DLEQ proofs into a single, compact group element. Crucially, the proofs for uniqueness and strong pseudorandomness established for DDH-DVRF rely solely on the NIZK property of the DLEQ proof element(s). Consequently, because Icy-DVRF fulfills the requisite NIZK conditions, it preserves the uniqueness and strong pseudorandomness properties inherent to the DDH-DVRF scheme.

4.5.3 Knowledge-Soundness of T-DLEQ-Int

This subsection establishes the knowledge-soundness of T-DLEQ-Int (invoked in the proof of Theorem 4.2) by a direct reduction to the discrete logarithm problem, and may be replaced as a unit should the underlying threshold DLEQ proof be substituted. The reduction follows the Generalized Forking Lemma of Bellare and Neven [10], extracting a discrete logarithm from two related forgeries produced on a common random tape.

The Generalized Forking Algorithm ($GF_{\mathcal{A}}$) is designed to run an adversary $\mathcal{A}$ twice to produce two related forgeries from a single random tape Ξ. The process operates in two distinct phases. In the *initial execution*, $GF_{\mathcal{A}}$ first supplies $\mathcal{A}$ with a public key Y, the random tape Ξ, and a set of simulated random oracle outputs $(h_1, \ldots, h_{n_r})$. If $\mathcal{A}$ successfully outputs a forgery π and identifies the corresponding oracle query index J, the algorithm proceeds to the forking stage. In the *forked execution*, the second run re-executes $\mathcal{A}$ using the identical random tape and the same oracle responses up to index $J-1$; however, for indices J through n_r, it provides a fresh set of random oracle values.

Algorithm 1 Generalized Forking Algorithm $GF_{\mathcal{A}}$

Input A public key Y

Output $(1, h_J, h_{J'}, \pi, \pi')$ if $\mathcal{A}$ produces two forgeries; otherwise, $\perp$

1: Instantiate a random tape Ξ
2: $\{h_1, \ldots, h_{n_r}\} \xleftarrow{\$} H$
3: (J, π) or $\perp \leftarrow \mathcal{A}(Y, \{h_1, \ldots, h_{n_r}\}; \Xi)$
4: If $\perp$, return $\perp$
5: $\{h'_1, \ldots, h'_{n_r}\} \xleftarrow{\$} H$
6: (J', π') or $\perp \leftarrow \mathcal{A}\left(Y, \left\{h_1, \ldots, h_{J-1}, h'_J, \ldots, h'_{n_r}\right\}; \Xi\right)$
7: If $J = J'$ and $h_J \neq h'_J$, return $(1, h_J, h_{J'}, \pi, \pi')$. Else, return $\perp$.

This approach effectively *forks* the adversary at the moment of the attack, conceptually similar to the Pointcheval and Stern [49] model, without requiring explicit rewinding. If the adversary produces a second valid forgery π' at the same index J but with a distinct oracle output $(h_J \neq h'_J)$, the algorithm successfully returns the tuple (π, π', h_J, h'_J). These values provide the necessary data for the coordinator $\mathcal{C}$ to extract the discrete logarithm of the challenge value ω.

Algorithm 2 The Coordinator Algorithm $\mathcal{C}(\omega)$

Input A challenge value ω

Output The discrete logarithm of ω, or $\perp$

1: Simulate DistKeyGen embedding the challenge value ω and extract the forger's secret values $\left(Y, (a_{1,0}, \ldots, a_{(t-1),0})\right) \leftarrow SimDistKeyGen(\omega)$
2: $(1, h_J, h_{J'}, \pi, \pi')$ or $\perp \leftarrow GF_{\mathcal{A}}(Y)$
3: If not $\perp$, ExtractDiscreteLog$(\omega, h_J, h_{J'}, \pi, \pi', (a_{1,0}, \ldots, a_{(t-1),0}))$

We now present the Generalized Forking Lemma by Bellare and Neven [10] in our setup. Let H be a set of size $h \geq 2$, and adv be the advantage of solving the discrete logarithm problem (DLP) over some group $\mathbb{G}$.

Lemma 4.4 (Generalized Forking Lemma [10])**.** *If IG is a randomized input generator algorithm, define acc to be the probability of $J \geq 1$ when $x \xleftarrow{\$} IG$, $(h_1, \ldots, h_q) \xleftarrow{\$} H$, and $(J, \pi) \xleftarrow{\$} \mathcal{A}(x, h_1, \ldots, h_{n_r})$. Also, define frk as the probability of $b = 1$, when $x \xleftarrow{\$} IG$ and $(b, \pi, \pi') \xleftarrow{\$} GF_{\mathcal{A}}(x)$. Then,*

$$adv \geq frk \geq acc \cdot \left(\frac{acc}{n_r} - \frac{1}{h}\right).$$

Lemma 4.4 shows the probability adv that the $GF_{\mathcal{A}}$ will produce two valid forgeries to enable the extraction of the desired discrete logarithm. In Algorithm 3, for simplicity of notation, assume $t = n$ where n is the total number of participants and t is the threshold, and that $\mathcal{F}$ controls $P_1, \ldots P_{t-1}$, and $\mathcal{A}$ simulates the honest participant P_t to $\mathcal{F}$.

Algorithm 3 Algorithm SimDistKeyGen(ω)

Input A challenge value ω and $\{\vec{C}_i\}_{i=1}^{t-1} = \left\{\left\langle \phi_{i,0}, \ldots, \phi_{i,(t-1)} \right\rangle\right\}_{i=1}^{t-1}$
Output $\vec{C}_t, Y_t$

1: **Round 1**: $\mathcal{A}$ does the following.
2: $\bar{x}_{t,1}, \ldots, \bar{x}_{t,t-1} \xleftarrow{\$} \mathbb{Z}_q$ serving as the secret shares corresponding to $f_t(1), \ldots f_t(t-1)$
3: $\phi_{t,0} \leftarrow \omega$
4: Calculate $\phi_{t,1}, \ldots, \phi_{t,(t-1)}$ by performing Lagrange Interpolation in the exponent, i.e. $\phi_{t,j} = \omega^{\lambda_{j,t}} \cdot g^{\sum_{i=1}^{t-1} \lambda_{j,i} \cdot \bar{x}_{t,i}}$
5: Broadcasts $\vec{C}_t = \left\langle \phi_{t,0}, \ldots, \phi_{t,(t-1)} \right\rangle$.
6: **Round 2**: $\mathcal{A}$ sends $(1, \bar{x}_{t,1}), \ldots, (t-1, \bar{x}_{t,t-1})$ to the parties corrupted by $\mathcal{F}$.
7: $\mathcal{A}$ simulates proof of knowledge of $a_{t,0}$ by deriving σ_t as $c_t, \mu_t \xleftarrow{\$} \mathbb{Z}_q$; $R_t = g^z \cdot \omega^{-\mu_t}$; $\sigma_t = (R_t, \mu_t)$.
8: $\mathcal{A}$ derives the public key Y_t for the party P_t by following the same steps the other parties can calculate Y_t as $Y_t = \prod_{j=1}^{t} \prod_{k=0}^{t-1} (\phi_{j,k})^{t^k \bmod q}$.

Note that in Line 8 of Algorithm 3, $\mathcal{A}$ cannot compute Y_t directly by g^{s_t} as both s_t and the discrete logarithm of ω are unknown to them. After the SimDistKeyGen Algorithm has completed, each party can follow the DistKeyGen protocol to derive the group public key as $Y = \Pi_{j=1}^{t} \phi_{j,0}$. Additionally, $\mathcal{C}$ must obtain the secret values $a_{1,0}, \ldots a_{(t-1),0}$ from $\mathcal{F}$ using the extractor for the zero-knowledge proofs that $\mathcal{F}$ generates, and in turn $\mathcal{C}$ can use these values to extract the discrete logarithm of ω from the discrete logarithm of the group public key Y.

Algorithm 4 Algorithm ExtractDiscreteLog($\omega, (a_{1,0}, \ldots a_{(t-1),0}), h_J, h'_J, \pi, \pi'$)

Input A challenge value ω, secret values $a_{1,0}, \ldots a_{(t-1),0}$, two distinct random oracle responses h_J, h'_J and their corresponding forgeries π, π'
Output The discrete logarithm of ω

1: Parse π, π' as $(\gamma, c, z), (\gamma, c', z')$, where $c = h_J$ and $c' = h'_J$
2: Compute discrete logarithm of Y as $\log_g(Y) = \frac{z'-z}{h'_J - h_J}$
3: Compute $a_{t,0} = \log_g(Y) - \sum_{i=1}^{t-1} a_{i,0}$
4: Return $\log_g(\omega) = a_{t,0}$

Regarding Line 1 of Algorithm 4, the Icy-DVRF consistency property guarantees that any two forgeries share the same γ. This applies to the general case ($t < n$) as well as the special case ($t = n$). In Line 2, because both forgeries are valid VRF proofs, it follows that

$$R = g^z \cdot Y^{-c} = g^{z'} \cdot Y^{-c'}$$

and since $c = h_J, c' = h'_J$ with $h_J \neq h'_J$, we can write $g^z \cdot g^{-z'} = Y^{h_J} \cdot Y^{-h'_J}$, which can

be re-written as $g^{\frac{z'-z}{h'_J-h_J}} = Y$, incidentally yielding

$$\log_g(Y) = \frac{z'-z}{h'_J-h_J} \ .$$

The full adversary simulator $\mathcal{A}$, which embeds the challenge ω into the simulated protocol transcript and programs the random oracles so that the forger's queries are answered consistently, is given in Algorithm 5.

Algorithm 5 Algorithm $\mathcal{A}(Y, \{h_1, \ldots, h_{n_r}\}; \Xi)$

Input A public key Y and random oracle outputs $\{h_1, \ldots, h_{n_r}\}$
Output An index J and a forgery π, or $\perp$

Initialize $ctr = 1, T_\rho = \{\}, T_\pi = \{\}, T_1 = \{\},$
$T_2 = \{\}, T_3 = \{\}, T_4 = \{\}, J_2 = \{\}, C = \{\}, M = \{\}$
Run $\mathcal{F}$ on input Y, answering its queries as follows, until it outputs $(\alpha, \pi = (\gamma, c, z))$ or $\perp$
On simulating $H_\rho(\alpha, B)$:
  If $T_\rho[\alpha, B] = \perp$, set $T_\rho[\alpha, B] = h_{ctr}$; $ctr = ctr + 1$. Return $T_\rho[\alpha, B]$.
On simulating $H_\pi(g, h, g^s, h^s, g^r, h^r)$:
  If $T_\pi[\cdot] = \perp$, set $T_\pi[\cdot] = h_{ctr}$; $ctr = ctr + 1$. Return $T_\pi[\cdot]$.
On simulating $H_1(\alpha)$:
  If $T_1[\alpha] = \perp$, sample $r \xleftarrow{\$} \mathbb{Z}_q$; $T_1[\alpha] = g^r$. Return $T_1[\alpha]$.
On simulating $H_2(g, h, Y, \gamma, R, \bar{R}, \alpha)$:
  If $T_2[\cdot] = \perp$, set $T_2[\cdot] = h_{ctr}$, $J_2[\cdot] = ctr$; $ctr = ctr + 1$. Return $T_2[\cdot]$.
On simulating $H_3(\mathbf{X})$:
  If $T_3[\mathbf{X}] = \perp$, set $T_3[\mathbf{X}] = h_{ctr}$; $ctr = ctr + 1$. Return $T_3[\mathbf{X}]$.
On simulating $H_4(i, \Phi)$:
  If $T_4[i, \Phi] = \perp$, set $T_4[i, \Phi] = h_{ctr}$; $ctr = ctr + 1$. Return $T_4[i, \Phi]$.
On simulating Preprocess$^{\mathbf{I}}$:
  Round 1: For $1 \leq j \leq \Omega$, do:
    Set $\bar{c}_j = h_{ctr}, C[j] = ctr, ctr = ctr + 1, \bar{z}_{t,j} \xleftarrow{\$} \mathbb{Z}_q, D_{t,j} = g^{\bar{z}_{t,j}} \cdot Y_t^{-\bar{c}_j}$.
    Follow the protocol honestly to sample $e_{t,j}, a_{t,j}, b_{t,j} \xleftarrow{\$} \mathbb{Z}_q$ and derive $E_{t,j} = g^{e_{t,j}}, A_{t,j} = g^{a_{t,j}}, B_{t,j} = g^{b_{t,j}}$.
    Follow the protocol honestly to sample $k_{a_{t,j}}, k_{b_{t,j}} \xleftarrow{\$} \mathbb{Z}_q$ and derive $R_{a_{t,j}} = g^{k_{a_{t,j}}}, R_{b_{t,j}} = g^{k_{b_{t,j}}}$.
  Derive $K_t = H_3(K_{t,1}, \ldots, K_{t,\Omega})$, publish to $\mathcal{F}$, wait for all K_ℓ values from $\mathcal{F}$.
  Round 2: Derive L_t, Φ, J_t honestly. Send (t, L_t, J_t) to $\mathcal{F}$, wait to receive the (ℓ, L_ℓ, J_ℓ) tuples from $\mathcal{F}$, following the protocol for validation.
  Reprogram $T_3[K_1, \ldots, K_t] = h_{ctr}$; set $ctr = ctr + 1$. Rederive c_t and J_t honestly.
  Rewind $\mathcal{F}$ to Step 1 in Round 2 of Preprocess-Interactive, immediately before $\mathcal{F}$

queries H_3 with $(K_1, \ldots, K_t)$.

24: After $\mathcal{F}$ rewinds, use the two outputs to derive the discrete logarithm of each $A_{l,j}$ and $B_{l,j}$; store for use in the later protocols.

25: Complete the protocol honestly.

26: **On simulating PartialEval**:

27: Insert α into M.

28: $\mathcal{A}$, acting as P_t, samples $r \xleftarrow{\$} \mathbb{Z}_q$, sets $h = g^r = H_1(\alpha)$.

29: $\mathcal{A}$ simulates $\gamma_t = (Y_t)^r = (g^r)^{s_t} = h^{s_t}$.

30: $\mathcal{A}$ simulates a DLEQ proof by sampling $\bar{s}, \bar{c} \xleftarrow{\$} \mathbb{Z}_q$, deriving $R_t = g^{\bar{s}} \cdot (Y_t)^{-\bar{c}}$ and $\bar{R}_t = h^{\bar{s}} \cdot (\gamma_t)^{-\bar{c}}$ and programs the random oracle so that $\bar{c} = H_\pi(g, h, g^{\bar{s}}, h^{\bar{s}}, R_t, \bar{R}_t)$, hence $(\bar{s}, \bar{c})$ is a valid DLEQ proof.

31: Derive $\bar{D}_t = h^{\bar{z}_{t,j}} \cdot \gamma^{\bar{c}_j}$ and $\bar{E}_t = h^{e_{t,j}}$.

32: Output $(\gamma_t, \pi_t, \bar{D}_t, \bar{E}_t)$ by sending it to $\mathcal{F}$.

33: **On simulating PartialProofGen**:

34: **Round 1**: Receive $(\gamma_\ell, \pi_\ell, \bar{D}_\ell, \bar{E}_\ell)$ for $\ell = 1, \ldots, t-1$.

35: Using $(a_{\ell,j}, b_{\ell,j})$ obtained during Preprocess-Interactive, derive ρ_ℓ for $\ell = 1 \ldots, t-1$.

36: Derive $\rho_t = a_{t,j} + b_{t,j} \cdot H_\rho(\alpha, B)$, R, $\bar{R}$, and γ, honestly following the protocol.

37: Program $T_2[g, h, Y, \gamma, R, \bar{R}, \alpha] = \bar{c}_j$, $J_2[g, h, Y, \gamma, R, \bar{R}, \alpha] = C[j]$.

38: Return ρ_t.

39: **Round 2**: Validate ρ_l for $\ell = 1, \ldots, t-1$.

40: Let $z_t = \bar{z}_{t,j} + e_{t,j} \cdot \rho_t, c = \bar{c}_j$. Return (z_t, c) to $\mathcal{F}$.

41: **On simulating Combine**:

42: If $\mathcal{F}$ outputs $\perp$, then return $\perp$. Else, $\mathcal{F}$ outputs $(\alpha, (\gamma, z, c))$.

43: If $T_2[g, h, Y, \gamma, R, \bar{R}, \alpha] = \perp$, set $T_2[\cdot] = h_{ctr}$, $J_2[\cdot] = ctr$, and $ctr = ctr + 1$.

44: $c = T_2[g, h, Y, \gamma, R, \bar{R}, \alpha]$. If $R \neq g^z \cdot Y^{-c}$, or $\bar{R} \neq h^z \cdot \gamma^{-c}$ or $\alpha \in M$, then return $\perp$.

45: Let $J = J_2[g, h, Y, \gamma, R, \bar{R}, \alpha]$. Return $J, \pi = (\gamma, z, c)$.

Let n_h, n_p, and n_v denote the numbers of queries to the random oracle, the preprocess queries, and the VRF-creation queries, respectively.

Theorem 4.5. *If the discrete logarithm problem in $\mathbb{G}$ is (τ', ϵ')-hard, then the T-DLEQ-Int scheme over $\mathbb{G}$ with n proving participants, a threshold of t, and a preprocess batch size of Ω is $(\tau, n_h, n_p, n_v, \epsilon)$-secure whenever*

$$\epsilon' \leq \frac{\epsilon^2}{2n_h + \Omega \cdot n_p}$$

and

$$\tau' \leq 4 \cdot \tau + [(8t+4) \cdot \Omega \cdot n_p + (10t+12) \cdot n_v + (n+t-1)t + 6]\, t_{exp} + O(\Omega \cdot n_p + n_v + n_h),$$

where t_{exp} is the time of an exponentiation in $\mathbb{G}$, assuming the number of participants compromised by the adversary is less than the threshold t.

Proof. Assume that $\mathcal{F}$ is a $(\tau, n_h, n_p, n_v, \epsilon)$-forger for T-DLEQ-Int. We show that the algorithm $\mathcal{C}$ presented in Algorithm 2 can (τ', ϵ')-solve the discrete logarithm of an arbitrary challenge $\omega \in \mathbb{G}$.

$\mathcal{A}$ from Algorithm 5 simulates T-DLEQ-Int to $\mathcal{F}$, who controls the dishonest participants $P_1, \ldots, P_{t-1}$ while $\mathcal{A}$ simulates the single honest participant P_t in the proof creation. Let $n_r = 2n_h + \Omega \cdot n_p$ be the maximum number of random oracle outputs $\mathcal{A}$ may require. Once SimDistKeyGen from Algorithm 3 is completed, $\mathcal{A}$ invokes $\mathcal{F}$ by simulating both the role of P_t and the responses to the random oracle queries of $\mathcal{F}$ in all of the protocols to follow.

To simulate random oracle queries for $H_\rho, H_\pi, H_1, H_2, H_3$, and H_4, the algorithm $\mathcal{A}$ uses a stateful-table-based approach. For any fresh query, $\mathcal{A}$ assigns the successor value from a set $\{h_1, \ldots, h_{n_r}\}$ supplied by $GF_{\mathcal{A}}$ and records the result to ensure consistency. The interactive protocol diverges from the original by replacing the static binding hash H_1 with an interactive one-time VRF that accepts $H_\rho(\alpha, B)$ as input.

For the simulation of Preprocess-Interactive, $\mathcal{A}$ simulates P_t, faithful to the standard protocol with a specific deviation during the generation of $D_{t,j}$: $\mathcal{A}$ selects a controlled hash value $\bar{c}_j$ and a random scalar $\bar{z}_{t,j}$ to derive $D_{t,j} = g^{\bar{z}_{t,j}} \cdot Y_t^{-\bar{c}_j}$. Crucially, to extract the discrete logarithms $(a_{\ell,j}, b_{\ell,j})$ from dishonest parties controlled by $\mathcal{F}$, $\mathcal{A}$ employs a forking strategy during the second round. By rewinding $\mathcal{F}$ to the point prior to the query $\Phi = H_3(K_1, \ldots, K_t)$ and reprogramming the random oracle with a distinct output Φ', $\mathcal{A}$ forces $\mathcal{F}$ to republish commitment values, thereby solving the underlying discrete logarithms.

During the simulation of PartialEval, $\mathcal{F}$ initiates T-DLEQ-Int as the combiner by sending (j, α, B). Once again, $\mathcal{A}$ simulates P_t, staying true to the protocol except that it simulates γ_t and the associated DLEQ proof π_t. Having possession of the values $\bar{c}_j$ and $\bar{z}_{t,j}$, $\mathcal{A}$ derives $(\bar{D}_t, \bar{E}_t)$ and sends them to $\mathcal{F}$.

To simulate PartialProofGen, $\mathcal{F}$ initiates T-DLEQ-Int as the combiner once again. This time $\mathcal{A}$ uses the discrete logarithms $(a_{\ell,j}, b_{\ell,j})$ extracted previously to compute its own ρ_t and the binding values ρ_ℓ for all other participants. Now $\mathcal{A}$ can derive $R, \bar{R}$, and γ, honestly following the protocol. $\mathcal{A}$ programs the random oracle $H_2(g, h, Y, \gamma, R, \bar{R}, \alpha) = \bar{c}_j$ and saves $C[j] = ctr$ so that $\bar{c}_j = h_{ctr}$, in the table J_2.

Finally, in the simulation phase for Combine, $\mathcal{F}$ initiates T-DLEQ-Int as the combiner, yet again. Since $\mathcal{A}$ is capable of computing the commitments $R, \bar{R}$ and γ prior to $\mathcal{F}$, it thereby guarantees the successful programming of the random oracle entry $H_2(g, h, Y, \gamma, R, \bar{R}, \alpha)$.

Note that the success probability of the adversary $\mathcal{F}$ in a single execution of the simulator $\mathcal{A}$ is denoted by ϵ, given that $\mathcal{A}$ is capable of returning the correct challenge for every signing query. By applying the General Forking Lemma, which entails running $\mathcal{A}$ twice, the discrete

logarithm of ω can be extracted. The probability of successful extraction is greater than $\frac{\epsilon^2}{n_r}$; this bound neglects the negligible term $\frac{\epsilon}{h}$ under the assumption that the hash output space h is sufficiently large (typically $h \geq 2^{256}$).

The runtime complexity of $\mathcal{F}$ is derived from several factors. The base runtime of $\mathcal{F}$ is scaled by a factor of four due to the nested forking strategy, wherein $\mathcal{A}$ forks $\mathcal{F}$, and the reduction itself forks $\mathcal{A}$. Accounting for the $O(\Omega \cdot n_p + n_v + n_h)$ minor operations, such as table lookups, the total running time is as stated. □

4.5.4 Knowledge-Soundness of T-DLEQ

In this subsection, we present the proof of the knowledge-soundness clause of Theorem 4.3: any PPT adversary that produces a verifying, non-trivial, T-DLEQ proof can be turned into a solver for the one-more discrete logarithm (OMDL) problem, with the distributed key generation handled by the Schnorr knowledge-of-exponent assumption (Schnorr-KoE), which holds under the discrete logarithm assumption in the algebraic group model (AGM) [21, Thm. 1].

4.5.4.1 Overview of Proof

We present our reduction in the *single-honest-party* convention as in the interactive-security analysis (Section 4.5.3). We assume the adversary $\mathcal{F}$ controls $P_1, \ldots, P_{t-1}$ and the OMDL challenge is embedded in the one honest prover P_t. We mimic the proof of security of FROST1 in the modular style of Crites-Komlo-Maller [21] and Bellare et al. [6]. Instead of rewinding the full protocol, we introduce an intermediate *binonce DLEQ* (Bi-DLEQ) assumption (Definition 4.6) that abstracts the preprocessing structure of Icy-DVRF, so that the T-DLEQ knowledge-soundness is reduced to Bi-DLEQ by a straight-line simulation of P_t that runs the adversary only once (Lemma 4.12). Then, Bi-DLEQ reduces to OMDL (Lemma 4.9), which keeps all rewinding inside a single forking step.

The proof is assembled from five components. Lemma 4.10 justifies the single-honest-party reduction and isolates its soundness factor κ, which is 1 under the fixed-quorum convention and at most $\min\{t, n - t + 1\} \leq n$ in the general setting. Using Lemma 4.11, one can extract the corrupt key shares via Schnorr-KoE and write the honest share in the affine form $Y_t = Y^{\alpha_t} g^{\beta_t}$ that the simulation relies on, while using Lemma 4.8, we show that the second verification equation (in base h) is redundant, so the extraction reduces to the single base g.

Then, a winning forgery is split into two disjoint events: an *insufficient-quorum* forgery ($\mathcal{E}_{\mathsf{quo}}$), handled by a general forking at the challenge query, and a *token-substitution* forgery ($\mathcal{E}_{\mathsf{sub}}$), handled by a local forking (in the sense of Bellare-Dai-Li [7]) of the single binding output ρ_t, which yields a two-equation linear system in the embedded discrete logarithms. The freshness predicate is based on the aggregate commitment pair $(R^*, \bar{R}^*)$.

While [21] proves the security of the three-round interactive FROST1 scheme under the DL assumption, the security of their two-round preprocessed variant relies on OMDL. Because Icy-DVRF inherits this preprocessing structure, OMDL, along with Schnorr-KoE/AGM for the DKG, is the appropriate assumption set. This remains consistent with Theorem 4.5, which is a DL statement, because avoiding the interactive one-time VRF in the interactive version of Icy-DVRF trades the DL assumption for OMDL.

4.5.4.2 Setting, Assumptions, and the Bi-DLEQ Assumption

Icy-DVRF is a (t, n)-threshold scheme: any t of the n provers reconstruct, and a static adversary corrupts a set Cor with $|\mathrm{Cor}| \leq t-1$, leaving honest set $\mathrm{Hon} = [n] \setminus \mathrm{Cor}$. The OMDL game gives a challenge oracle $\mathsf{Chal}()$ returning a fresh g^x (x secret) and a discrete logarithm oracle $\mathsf{Dlog}(\cdot)$; with c_{id} challenge and ℓ discrete logarithm queries, the adversary wins iff $\ell < c_{\mathrm{id}}$ and it outputs all challenge logarithms; $\mathsf{Adv}^{\mathrm{omdl}}_{\mathbb{G}}(\mathcal{B})$ is the advantage of this adversary [21, Def. 2]. Schnorr-KoE provides an extractor Ext that recovers the secret key behind any verifying Schnorr proof of possession, except with probability $\mathsf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\cdot)$ [21, Assumption 4], [6, Theorem 8].

In the remainder of this chapter, $\mathbb{G} = \langle g \rangle$ has prime order q, and $H_1 : \{0,1\}^* \to \mathbb{G}$ and $H_2 : \{0,1\}^* \to \mathbb{Z}_q$ are random oracles, with $h = H_1(\alpha)$, $\rho_\ell = H_2(\ell, \alpha, B)$, and $c = H_2(g, h, Y, \gamma, R, \bar{R}, \alpha)$, observing the domain separation conventions of Section 4.2.1. Also, let n_h and n_{pr} be the number of random-oracle and proving queries, respectively, and $Q = n_h + n_{\mathsf{pr}} + 1$.

Definition 4.6. Let $\mathbb{G} = \langle g \rangle$ be a (cyclic) group of prime order q. The Bi-nonce Discrete Logarithm Equality (Bi-DLEQ) assumption holds if for all PPT adversaries $\mathcal{A}$,

$$\mathsf{Adv}^{\text{bi-dleq}}_{\mathbb{G}}(\mathcal{A}) = \Pr[\,\mathsf{G}^{\text{bi-dleq}}_{\mathbb{G}}(\mathcal{A}) \Rightarrow \mathsf{true}\,]$$

is negligible, where $\mathsf{G}^{\text{bi-dleq}}_{\mathbb{G}}$ is the game of Algorithm 6.

Algorithm 6 The Bi-DLEQ Game $\mathsf{G}^{\text{bi-dleq}}_{\mathbb{G}}$ (Binonce DLEQ; Two Bases)

Init: $\dot{x} \xleftarrow{\$} \mathbb{Z}_q$; $Y \leftarrow g^{\dot{x}}$; $Q_{\mathrm{bin}}, Q_{\mathrm{used}}, Q_{\mathrm{prove}} \leftarrow \emptyset$;
return Y

$\mathsf{BinonceO}()$: $d, e \xleftarrow{\$} \mathbb{Z}_q$; $(D, E) \leftarrow (g^d, g^e)$; $Q_{\mathrm{bin}} = Q_{\mathrm{bin}} \cup \{(D, E, d, e)\}$;
return (D, E)

$\mathsf{BiproveO}\big(k, \alpha, S, \{(\psi_i, D_i, E_i, \bar{D}_i, \bar{E}_i)\}_{i \in S}, \gamma\big)$:
  require $(D_k, E_k, d_k, e_k) \in Q_{\mathrm{bin}}$ **and** $(D_k, E_k) \notin Q_{\mathrm{used}}$
  $Q_{\mathrm{used}} = Q_{\mathrm{used}} \cup \{(D_k, E_k)\}$; $h \leftarrow H_1(\alpha)$

7: **require** $\bar{D}_k = h^{d_k} \wedge \bar{E}_k = h^{e_k}$
8: **for** $i \in S$: $\rho_i \leftarrow H_2(i, \alpha, B_S)$; $R \leftarrow \prod_i D_i E_i^{\rho_i}$; $\bar{R} \leftarrow \prod_i \bar{D}_i \bar{E}_i^{\rho_i}$
9: $c \leftarrow H_2(g, h, Y, \gamma, R, \bar{R}, \alpha)$; $z_k \leftarrow d_k + \rho_k e_k + c\,\psi_k\,\dot{x}$
10: $\mathsf{ctr}(\alpha, R, \bar{R}) = \mathsf{ctr}(\alpha, R, \bar{R}) \cup \{(k, B_S)\}$; **return** z_k

11: $\mathsf{RO}_1(x)$: **return** $H_1(x)$ $\quad$ $\mathsf{RO}_2(x)$: **return** $H_2(x)$

12: **Fin**$(\alpha^*, \gamma^*, c^*, z^*)$: $h^* \leftarrow H_1(\alpha^*)$; $T \leftarrow g^{z^*} Y^{-c^*}$; $\bar{T} \leftarrow (h^*)^{z^*} (\gamma^*)^{-c^*}$
13: **require** $c^* = H_2(g, h^*, Y, \gamma^*, T, \bar{T}, \alpha^*)$
14: **require** $\left|\{k : (k, \cdot) \in \mathsf{ctr}(\alpha^*, T, \bar{T})\}\right| < t - |\mathrm{Cor}|$
15: **return** true

The Bi-DLEQ game of Algorithm 6 gives an adversary access to two oracles, BinonceO and BiproveO, two random oracles H_1, H_2, and a challenge public key $Y = g^{\dot{x}}$, and asks it to forge a fresh two-base DLEQ proof with respect to Y. The BinonceO oracle takes no input and returns a "token", i.e., a one-time commitment pair, $(D, E) = (g^d, g^e)$, recording the discrete logarithms (d, e) for later use. The BiproveO oracle takes an index k, an input α, a prover set S, the per-prover commitments $\{(\gamma_i, D_i, E_i, \bar{D}_i, \bar{E}_i)\}_{i \in S}$, and the aggregate evaluation γ. It checks if (D_k, E_k) is a prior BinonceO response that has not been used before, returning an error otherwise. Then, it forms the bindings $\rho_i = H_2(i, \alpha, B_S)$, the aggregate commitments $R = \prod_i D_i E_i^{\rho_i}$ and $\bar{R} = \prod_i \bar{D}_i \bar{E}_i^{\rho_i}$, the challenge $c = H_2(g, h, Y, \gamma, R, \bar{R}, \alpha)$, and returns z_k such that $(R, \bar{R}, c, z_k)$ is a valid partial response for slot k. The single-use restriction ensures that BiproveO may be called at most once per commitment pair (D, E), and limits the reduction to one Dlog query per challenge. This is the source of the OMDL budget of Lemma 4.9. The scalar ψ_k lets the response be given as $z_k = d_k + \rho_k e_k + c\,\psi_k\,\dot{x}$ rather than $d_k + \rho_k e_k + c\,\dot{x}$; in the threshold setting ψ_k absorbs the Lagrange coefficient λ_k together with the affine key factor α_k of the honest-share representation (Lemma 4.11). The adversary wins if it outputs a verifying $(\alpha^*, \gamma^*, c^*, z^*)$ whose aggregate commitment pair $(T, \bar{T})$ records fewer than $t - |\mathrm{Cor}|$ honest contributions, i.e. a proof object not produced by a full honest quorum.

We present our proof of Theorem 4.3 in a modular fashion using a sequence of lemmas. Lemmas 4.7, 4.8, and 4.9 are independent of which parties are corrupt.

Lemma 4.7. *For any statement (g, h, Y_k, γ_k) with $\gamma_k = h^{s_k}$ and α such that $h = H_1(\alpha)$, the simulator that samples $\bar{c}, \bar{z} \xleftarrow{\$} \mathbb{Z}_q$, sets $R_k = g^{\bar{z}} Y_k^{-\bar{c}}$, $\bar{R}_k = h^{\bar{z}} \gamma_k^{-\bar{c}}$, and programs $H_2(g, h, Y_k, \gamma_k, R_k, \bar{R}_k, \alpha) \leftarrow \bar{c}$, yields a proof $(\bar{c}, \bar{z})$ identically distributed to an honest one, provided the programmed point was undefined.*

Proof. In an honest generation, we sample $r \xleftarrow{\$} \mathbb{Z}_q$ and output $\bar{c} = H_2(g, h, Y_k, \gamma_k, R_k, \bar{R}_k, \alpha)$, which is uniform since H_2 is a random oracle, and $\bar{z} = r - s_k \bar{c}$, which is also uniformly distributed and independent of $\bar{c}$, since r is uniformly distributed. The simulator draws $(\bar{c}, \bar{z})$ uniformly and independently, and defines $(R_k, \bar{R}_k)$ by the

verification relations $R_k = g^{\bar{z}} Y_k^{-\bar{c}}$, $\bar{R}_k = h^{\bar{z}} \gamma_k^{-\bar{c}}$, which fix them identically; programming the challenge oracle at exactly the point the verifier reads makes $(\bar{c}, \bar{z})$ pass verification. Hence the two output distributions coincide. □

Lemma 4.8. *Let $h = H_1(\alpha^*) = g^\eta$ with $\eta \in \mathbb{Z}_q^*$ known and $\varphi(x) = x^\eta$, a bijection on $\mathbb{G}$. When $\gamma^* = h^s$, we have $\gamma^* = Y^\eta$, therefore $g^z = R^* Y^{c^*}$ if and only if $h^z = \bar{R}^* (\gamma^*)^{c^*}$. Hence, the twin equation carries no information independent of the base-g equation; any discrete log extraction valid in the single-base setting transfers verbatim, with $\bar{D}_i = D_i^\eta$, $\bar{E}_i = E_i^\eta$.*

Proof. Note that φ is a homomorphism and a bijection since $\gcd(\eta, q) = 1$. Under φ, g^z is mapped to h^z, $R^* = \prod_i D_i E_i^{\rho_i}$ is mapped to $\prod_i D_i^\eta E_i^{\eta\rho_i} = \prod_i \bar{D}_i \bar{E}_i^{\rho_i} = \bar{R}^*$, and Y^{c^*} is mapped to $(Y^\eta)^{c^*} = (\gamma^*)^{c^*}$. Since applying a bijection to both sides of an equation results in an equivalence, the two verification equations are logically equivalent under the uniqueness premise. Therefore, the system a forger must satisfy, and the system an extractor solves, are identical to the single-base case. □

Lemma 4.9. *For every Bi-DLEQ adversary $\mathcal{A}$ making at most n_b queries to* BinonceO*, n_{pr} to* BiproveO*, and n_h random-oracle queries, there is an OMDL adversary $\mathcal{B}$ with*

$$\mathsf{Adv}^{\text{bi-dleq}}_{\mathbb{G}}(\mathcal{A}) \le \sqrt{Q\big(\mathsf{Adv}^{\text{omdl}}_{\mathbb{G}}(\mathcal{B}) + 3(n+1)^2 Q^2/q\big)},$$

where $Q = n_h + n_{\mathsf{pr}} + 1$, $\mathcal{B}$ making at most $2n_b + 1$ challenge and $2n_b$ discrete logarithm queries, where $\mathcal{B}$ runs in time roughly twice that of $\mathcal{A}$ plus the time to perform $O\big((n{+}1)(n_b + n_{\mathsf{pr}} + n_h)\big)$ group operations.

Proof. $\mathcal{B}$ plays OMDL and simulates $\mathsf{G}^{\text{bi-dleq}}_{\mathbb{G}}$ to $\mathcal{A}$.

Embedding the challenge. $\mathcal{B}$ calls $\mathsf{Chal}()$ once for $Y = g^{\dot{x}}$, never queries $\mathsf{Dlog}(Y)$, and programs $h = H_1(\alpha) = g^{\eta_\alpha}$ with known η_α for each fresh α. Thus, a single base g defines both the g- and h-commitments. Hence, anything simulatable at base g lifts to base h by exponentiating by η_α, which is what lets a two-base forgery yield a single-base OMDL solution.

BinonceO. On each binonce request, $\mathcal{B}$ draws a fresh challenge $\mathsf{Chal}() \to U_j = g^{u_j}$ with u_j unknown, samples $e_j \xleftarrow{\$} \mathbb{Z}_q$ itself, and answers with $(D_j, E_j) \leftarrow (U_j, g^{e_j})$. The reply is distributed exactly as in the real game, since both the challenge exponent u_j and the exponents of the game's BinonceO are uniform over $\mathbb{Z}_q$. Note that the challenge is embedded at binonce time, before $\mathcal{A}$ sees anything: the D_j that $\mathcal{A}$ receives is the value the reduction works with for the rest of the game, and no later substitution takes place. This is the reason no replacement-detection event arises. The discrete logarithm e_j is kept, since the response computation and the twin check both use it.

BiproveO. On a query consuming binonce k, $\mathcal{B}$ first simulates the membership and single-use checks directly, as it keeps Q_{bin} and Q_{used} itself. The twin check of the game reads $\bar{D}_k \stackrel{?}{=} h^{d_k}$

and $\bar{E}_k \stackrel{?}{=} h^{e_k}$, and the first of these involves the unknown $d_k = u_k$; since $h = H_1(\alpha) = g^{\eta_\alpha}$ with η_α known, the check is equivalent to $\bar{D}_k \stackrel{?}{=} D_k^{\eta_\alpha}$, which $\mathcal{B}$ evaluates from public values, and likewise $\bar{E}_k \stackrel{?}{=} E_k^{\eta_\alpha}$. For the response, the honest value is $z_k = d_k + \rho_k e_k + c\,\psi_k\,\dot{x}$, in which both $d_k = u_k$ and $\dot{x}$ are unknown to $\mathcal{B}$. Both unknowns sit in the single group element $W_k := U_k Y^{c\psi_k}$, so one query $\mathsf{Dlog}(W_k)$ returns $w_k = u_k + c\psi_k\dot{x}$, and $\mathcal{B}$ answers $z_k \leftarrow w_k + \rho_k e_k$. The twin relation holds as $h^{z_k} = (g^{z_k})^{\eta_\alpha}$. By the single-use requirement each $\mathsf{BiproveO}$ query consumes a distinct binonce, hence exactly one Dlog query; no Chal is drawn here, since the challenge was already spent at binonce time.

Unused binonces. We assume that $\mathcal{A}$ issues no $\mathsf{BiproveO}$ query after the random-oracle query at the forking index J. This is a standard normalization of the forking argument: the forgery is verified against the answer to that query, so an adversary that queries afterwards may be replaced by one that halts there, with the same success probability. All proving queries therefore fall at indices before J and are answered identically in both executions. No binonce is consumed twice, so none costs a second Dlog query. Winning the OMDL game requires $\mathcal{B}$ to output the discrete logarithm of every challenge it drew. Then, a binonce that $\mathcal{A}$ requests but never consumes in a $\mathsf{BiproveO}$ query leaves a challenge with no associated $\mathsf{Dlog}(W)$ query. Let n'_b be the number of distinct binonces created across the two executions of $\mathcal{A}$ in the forking step. Each such binonce costs exactly one Chal, since a binonce query answered before index J receives the same U_j in both executions, so $n'_b \leq 2n_b$. Let n'_{pr} be the number of $\mathsf{BiproveO}$ queries answered. After $\mathcal{A}$ terminates, $\mathcal{B}$ spends one $\mathsf{Dlog}(U_j)$ query on each of the $n'_b - n'_{\mathsf{pr}}$ binonces that were never consumed, obtaining u_j directly. Each cleanup query is matched by the Chal that created its binonce, so the only unmatched challenge remains Y.

Budget. With n'_b and n'_{pr} as above, the query counts are as follows.

phase	# Chal	# Dlog
key embedding	1	0
each BinonceO	1	0
each BiproveO	0	1
each unused binonce	0	1
forking / Fin	0	0
total	$1 + n'_b$	n'_b

The key embedding draws one Chal to obtain $Y = g^{\dot{x}}$ and never queries $\mathsf{Dlog}(Y)$. The three binonce rows record one invariant split across the moments it is paid: a binonce draws its Chal when it is created, for D_j and not for E_j, which is self-sampled, and returns its Dlog afterwards, as $\mathsf{Dlog}(W_k)$ if a $\mathsf{BiproveO}$ query consumes it and as $\mathsf{Dlog}(U_j)$ in the cleanup step if none does. Consumed and unused are exhaustive and exclusive, so the Dlog column counts every binonce exactly once and both columns are driven by n'_b alone. The forking and Fin steps recover the remaining discrete logarithms purely arithmetically, without further

oracle calls. Thus, in the notation of the OMDL game, $c_{\mathrm{id}} = 1 + n'_b$ and $\ell = n'_b$: the single unbalanced challenge is Y, whose discrete logarithm is supplied by the forgery.

Forking and extraction. Let J index the query for H_2 at $(g, h^*, Y, \gamma^*, R^*, \bar{R}^*, \alpha^*)$ whose output equals c^* in $\mathcal{A}$'s forgery. By the Generalized Forking Lemma, rerunning $\mathcal{A}$ on the same coins and oracle answers up to index $J-1$ yields, with the stated probability, a second forgery with the same $(\alpha^*, \gamma^*, R^*, \bar{R}^*)$ but $c^{*\prime} \neq c^*$, and hence

$$\dot{x} = \log_g Y = \frac{z^* - z^{*\prime}}{c^* - c^{*\prime}} \bmod q\,.$$

Then, $\mathcal{B}$ outputs $\dot{x}$, each consumed $u_j \leftarrow w_j - c_j \psi_j \dot{x}$, and each unused u_j obtained directly in the cleanup step, to win the OMDL game.

Bad-event slack. A single proving query writes, besides the one challenge entry, up to n binding entries $\rho_i = H_2(i, \alpha, B)$. Over the complete run, the game touches at most $N = (n+1)Q$ distinct H_2 entries, since each of the at most Q queries contributes one challenge entry and at most n binding entries. The simulation is perfect unless one of three bad events occurs.

- "programming collision": when $\mathcal{B}$ programs one of the at most N touched H_2 entries, the value it samples uniformly from $\mathbb{Z}_q$ coincides with an entry already defined. Each unordered pair of the N entries collides with probability $1/q$, so a union bound over all $\binom{N}{2}$ pairs gives

$$\Pr[\text{prog. collision}] \le \binom{N}{2}\frac{1}{q} = \binom{(n+1)Q}{2}\frac{1}{q} \le \frac{(n+1)^2Q^2}{2q}.$$

- "output collision": among the fewer than $2N$ random-oracle outputs consumed by the simulation across both executions, two coincide. By the same pairwise union bound over $\binom{2N}{2}$ pairs,

$$\Pr[\text{output collision}] \le \binom{2N}{2}\frac{1}{q} \le \frac{(2N)^2}{2q} = \frac{2(n+1)^2Q^2}{q}.$$

- "premature challenge-hash query": $\mathcal{A}$ queries H_2 at the forgery's challenge point $(g, h^*, Y, \gamma^*, R^*, \bar{R}^*, \alpha^*)$ before $\mathcal{B}$ has programmed it. Since R^* carries $\log_2 q$ bits of entropy unknown to $\mathcal{A}$ until programming, each of the at most n_h random-oracle queries hits this point with probability $1/q$, giving

$$\Pr[\text{premature query}] \le \frac{n_h}{q}.$$

Taking the union bound over these three events,

$$\Pr[\mathrm{BAD}] \le \frac{(n+1)^2Q^2}{2q} + \frac{2(n+1)^2Q^2}{q} + \frac{n_h}{q}\,.$$

Using $n_h \le Q$ and $n \ge 2$, noting that any threshold committee has $n \ge t \ge 2$, to absorb the lower-order terms into the dominant $(n+1)^2Q^2/q$ terms, we get

$$\Pr[\mathrm{BAD}]+1/q \le 3(n+1)^2Q^2/q\,,$$

the $1/q$ being the additive term the forking lemma contributes below, absorbed here so that the two enter the final bound as a single quantity.

Under the condition $\neg\mathrm{BAD}$, the simulation is perfect, and the forking lemma gives $\mathrm{acc}^2 \le Q(\mathrm{frk}+1/q)$ with $\mathrm{frk} \le \mathsf{Adv}^{\mathrm{omdl}}_{\mathbb{G}}(\mathcal{B}) + \Pr[\mathrm{BAD}]$, which is the stated bound. □

4.5.4.3 Knowledge-Soundness in the Single-Honest-Party Presentation

Our reduction treats the worst-case scenario when $|\mathrm{Cor}| = t-1$, in which the adversary corrupts the maximum permitted number of parties and the honest set reduces to a single prover P_t, into whose key we embed the OMDL challenge. We prioritize isolating the soundness of the single-honest-party reduction, since it dictates the tightness of the subsequent argument. If the forgery quorum contains multiple honest provers, the reduction must guess which one is substituted, incurring a multiplicative loss proportional to the quorum size. When we collapse the honest set to a single party, we are removing that guess, so the token-substitution extraction (the event $\mathcal{E}_{\mathsf{sub}}$ in the proof of Theorem 4.13) proceeds directly. This is precisely why this collapse cannot be treated as a free "without loss of generality", and its cost must be charged explicitly, which is the content of Lemma 4.10.

Lemma 4.10. *It suffices to bound* $\mathsf{Adv}^{\text{t-dleq}}(\mathcal{A})$ *for adversaries that corrupt exactly* $|\mathrm{Cor}| = t-1$ *parties, relabeled* $P_1,\ldots,P_{t-1}$*, and produce a forgery whose quorum* S^* *contains a designated honest party* P_t *whose key contains the embedded challenge. Concretely, for every PPT* $\mathcal{A}'$ *corrupting* Cor' *with* $|\mathrm{Cor}'| \le t-1$*, there is a PPT* $\mathcal{A}$ *of the restricted form with*

$$\mathsf{Adv}^{\text{t-dleq}}(\mathcal{A}') \;\le\; \kappa \cdot \mathsf{Adv}^{\text{t-dleq}}(\mathcal{A}),$$

where $\kappa = 1$ *under the fixed-quorum convention of the interactive proof: the quorum is* $\{P_1,\ldots,P_t\}$ *with corrupt set* $\{P_1,\ldots,P_{t-1}\}$*, so* P_t *is the single honest party and* $|S^* \cap \mathrm{Hon}| = 1$*. For general adversary-chosen quorums,* $\kappa \le \min\{t,\, n-t+1\} \le n$*.*

Proof. If $|\mathrm{Cor}'| = c < t-1$, the reduction corrupts $t-1-c$ further parties itself; it can do so at no cost, because by the affine representation of Lemma 4.11 applied to those parties, it already knows their secret keys, and granting the adversary additional corruptions only enlarges its view. Hence $\mathsf{Adv}^{\text{t-dleq}}(\mathcal{A}') \le \mathsf{Adv}^{\text{t-dleq}}(\mathcal{A}'')$ for an $\mathcal{A}''$ with $|\mathrm{Cor}''| = t-1$.

After the self-corruption we have $|\mathrm{Cor}''| = t-1$, and since a valid forgery quorum has $|S^*| = t$, at least one party of S^* must be honest: $|S^* \cap \mathrm{Hon}''| = t - |S^* \cap \mathrm{Cor}''| \ge 1$. We must embed the OMDL challenge in the specific honest party on which the forgery relies: the one whose published token is reused, or, if no token is reused, the one at whose challenge query the extraction forks. We designate that party as P_t. Under the fixed-quorum convention the

quorum is exactly $\{P_1, \ldots, P_t\}$ with $P_1, \ldots, P_{t-1}$ corrupt, so S^* contains the single honest party P_t, which is therefore determined without any guessing and $\kappa = 1$. For a general adversary-chosen quorum, however, S^* may contain several honest parties; we then guess in advance which one the forgery exercises, choosing uniformly among the $|S^* \cap \mathrm{Hon}''| \leq \min\{t,\, n - t + 1\}$ honest members of S^*, and the guess is correct with probability at least $1/\kappa$.

When we relabel $P_1, \ldots, P_{t-1}$ as the corrupt set, and P_t as the embedded honest party, we do not break soundness, since nothing in the proof system or in either assumption depends on the actual party indices: H_2 uses an index only as a binding tag inside $\rho_\ell = H_2(\ell, \alpha, B)$, so permuting the indices permutes the tags consistently. OMDL and Schnorr-KoE are stated without reference to party identities, and Lagrange interpolation is equivariant under any permutation of $\{1, \ldots, n\}$, so the reconstruction remains unchanged when the parties are renamed. Therefore, without loss of generality, we may assume that the designated honest party is P_t. □

Under the convention $\mathrm{Cor} = \{1, \ldots, t-1\}$, $\mathrm{Hon} = \{t\}$, the quorum threshold is $t - |\mathrm{Cor}| = 1$, so the non-triviality predicate reduces to "zero honest contributors": a forgery wins precisely when the single honest party P_t made no genuine contribution to the aggregate. Consequently, when the forgery reuses an honest party's token, that party is necessarily P_t, and there is nothing further to guess. The following two lemmas instantiate the shared simulation of Section 4.5.4.2 at the single-honest-party setting $\mathrm{Hon} = \{t\}$: Lemma 4.11 fixes the algebraic form of P_t's key, and Lemma 4.12 shows that every message P_t must produce can be simulated from that form together with the Bi-DLEQ oracles, without ever rewinding.

Lemma 4.11. *Let f be the sharing polynomial of* DistKeyGen *of degree $t-1$, with group secret $f(0) = s$, shares $f(i) = s_i$, and the group public key $Y = g^s$. Let $\mathcal{B}_{\mathsf{koe}}$ denote the Schnorr-KoE adversary that runs the Pedersen-DKG-with-proofs-of-possession simulator (PedPoP), embeds the OMDL challenge as the group key by setting $f(0) = \dot{x}$ with $Y = g^{\dot{x}}$, and recovers the corrupt shares $\{x_j\}_{j \in \mathrm{Cor}}$, where $x_j = f(j)$, in the manner set out in the proof. Let L_0 and $\{L_j\}_{j \in \mathrm{Cor}}$ denote the Lagrange basis polynomials for the node set $\{0\} \cup \mathrm{Cor}$, and define the scalars*

$$\alpha_t = L_0(t), \qquad \beta_t = \sum_{j \in \mathrm{Cor}} L_j(t)\, x_j,$$

both of which are computable by the reduction. Then the honest share and public share of P_t admit the affine representation

$$s_t = \alpha_t\, \dot{x} + \beta_t, \qquad Y_t = Y^{\alpha_t} g^{\beta_t}.$$

The distributed key generation is analyzed under Schnorr-KoE, and the analysis fails with probability at most $\mathsf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_{\mathsf{koe}})$.

Proof. At the node 0, $\mathcal{B}_{\mathsf{koe}}$ embeds the OMDL challenge as the secret. The Schnorr proof of possession σ_j certifies knowledge of the constant term $a_{j,0}$ of the dealing polynomial f_j

and of nothing else, so the extractor applied to the corrupted proofs returns exactly the scalars $\{a_{j,0}\}_{j\in\mathrm{Cor}}$. The embedding itself needs no scalars: $\mathcal{B}_{\mathsf{koe}}$ fixes its own constant term once the corrupt commitment vectors are broadcast, so that $\sum_\ell a_{\ell,0} = \dot{x}$ and hence $f(0) = \dot{x}$ with $Y = g^{\dot{x}}$. It carries $\dot{x}$ symbolically, since learning it would itself solve the discrete logarithm challenge, and publishes $\phi_{t,0} = Y \cdot \prod_{i\neq t} \phi_{i,0}^{-1}$, which is computable from public values. The remaining entries of its commitment vector are obtained by interpolation in the exponent from the shares $f_t(j)$, $j \in \mathrm{Cor}$, which $\mathcal{B}_{\mathsf{koe}}$ samples itself (Algorithm 3).

The extractor therefore does not supply the shares $x_j = f(j)$, which are obtained from the commitment vectors instead. Round 1 of DistKeyGen broadcasts $\vec{C}_j = \langle \phi_{j,0}, \ldots, \phi_{j,(t-1)} \rangle$ with $\phi_{j,k} = g^{a_{j,k}}$, and the algebraic adversary $\mathcal{F}$ publishing these must supply a representation of each entry in the group elements it has received so far. Writing $g_1, \ldots, g_m$ for those elements, the representation of $\phi_{j,k}$ is a vector $(c_{j,k,0}, \ldots, c_{j,k,m}) \in \mathbb{Z}_q^{m+1}$ with

$$\phi_{j,k} = g^{c_{j,k,0}} \prod_{i=1}^{m} g_i^{c_{j,k,i}} \ .$$

Every g_i was produced by $\mathcal{B}_{\mathsf{koe}}$ itself from an exponent it chose, so $\mu_i = \log_g g_i$ is known, and the only element carrying $\dot{x}$ is $\phi_{t,0}$, which is published after the corrupt vectors and is therefore absent from $g_1, \ldots, g_m$. Taking discrete logarithms gives

$$a_{j,k} = c_{j,k,0} + \sum_{i=1}^{m} c_{j,k,i}\, \mu_i \ ,$$

a scalar in $\mathbb{Z}_q$ rather than an affine form in $\dot{x}$. Ranging over $0 \leq k \leq t-1$ this yields every coefficient of f_j, hence f_j itself, for each $j \in \mathrm{Cor}$. The remaining polynomials need no extraction since $\mathcal{B}_{\mathsf{koe}}$ sampled f_ℓ itself for every honest $\ell \neq t$ and for every party the reduction corrupts under Lemma 4.10. Only f_t has a symbolic constant term, and there $\mathcal{B}_{\mathsf{koe}}$ chose the shares $f_t(j)$ for $j \in \mathrm{Cor}$ in the embedding above, so these are known scalars too. Since $f = \sum_{\ell=1}^{n} f_\ell$, evaluating all n dealing polynomials at j gives $x_j = \sum_{\ell=1}^{n} f_\ell(j)$ for every $j \in \mathrm{Cor}$.

The node 0 contributes the evaluation $\dot{x}$, carried symbolically, and the corrupt nodes the $t-1$ known scalars x_j. Together these are t evaluations of a polynomial of degree $t-1$, so f is fixed, and with it the honest share $f(t)$, which the reduction knows only up to the single unknown $\dot{x}$.

The Lagrange basis polynomials for $\{0, \ldots, t-1\}$ satisfy $L_0(0) = 1$ and $L_0(j) = 0$ for $j \in \mathrm{Cor} = \{1, \ldots, t-1\}$, while each L_j satisfies $L_j(j) = 1$ and vanishes on $\{0, \ldots, t-1\} \setminus \{j\}$. Therefore, Lagrange interpolation evaluated at t gives

$$s_t \;=\; f(t) \;=\; L_0(t)\, f(0) + \sum_{j\in\mathrm{Cor}} L_j(t)\, f(j) \;=\; L_0(t)\, \dot{x} + \sum_{j\in\mathrm{Cor}} L_j(t)\, x_j \;=\; \alpha_t\, \dot{x} + \beta_t.$$

The basis polynomials depend only on the node set $\{0, \ldots, t-1\}$ and the evaluation point t, all known to $\mathcal{B}_{\mathsf{koe}}$, so α_t and β_t are known scalars. The challenge $\dot{x}$ appears merely in the single term $\alpha_t\, \dot{x}$, restricting the entire dependence on the unknown to the one coefficient α_t.

Raising the affine relation to the generator g removes the need to know $\dot{x}$:

$$Y_t = g^{s_t} = g^{\alpha_t \dot{x} + \beta_t} = (g^{\dot{x}})^{\alpha_t} g^{\beta_t} = Y^{\alpha_t} g^{\beta_t},$$

with both α_t, β_t known.

The embedding and extraction are carried out by $\mathcal{B}_{\mathsf{koe}}$ as described above, and the extraction fails with probability at most $\mathsf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_{\mathsf{koe}})$. The Schnorr-KoE assumption it relies on follows from the discrete logarithm (DL) assumption in the algebraic group model (AGM) [21, Thm. 1], [6, Thm. 8]: any algebraic adversary that produces a verifying Schnorr proof of possession also yields its discrete logarithm, so no assumption beyond DL is needed for the extractor. The AGM is invoked only in this DKG phase of the reduction. □

Lemma 4.12. *Let* Sim *be the simulator that plays the honest party P_t using the key representation of Lemma 4.11 and the Bi-DLEQ oracles of Definition 4.6, while the adversary $\mathcal{F}$ runs the corrupted parties $P_1, \ldots, P_{t-1}$. Then* Sim *answers every* Preprocess*,* PartialEval*,* PartialProofGen*, and* Combine *message of P_t in a single execution of $\mathcal{A}$, without rewinding and without knowledge of the secret share s_t or the nonces of P_t. The resulting transcript is identically distributed to a real one, except on a programming-collision event of probability at most Q^2/q.*

Proof. We exhibit, for each message P_t must send, a simulation that uses only public values, the affine key representation $Y_t = Y^{\alpha_t} g^{\beta_t}$, $s_t = \alpha_t \dot{x} + \beta_t$ of Lemma 4.11, and the Bi-DLEQ oracles. None of the four steps uses s_t itself, the secret nonces of P_t, or any rewinding, and the corrupted parties need no simulation since $\mathcal{F}$ runs them.

(i) Preprocess *and* PartialEval. The token of P_t is obtained as a fresh $\mathsf{BinonceO}() \to (D_t, E_t)$, so Sim does not need the nonce discrete logarithms. To handle the input α, Sim programs the random oracle as $h = H_1(\alpha) = g^{\eta_\alpha}$ with a self-chosen, known exponent η_α, and this makes the base h a known power of g and lets every base-h value be computed from its base-g counterpart. Then, the partial evaluation γ_t of P_t becomes public, since by Lemma 4.11

$$\gamma_t = h^{s_t} = (g^{s_t})^{\eta_\alpha} = Y_t^{\eta_\alpha} = (Y^{\alpha_t} g^{\beta_t})^{\eta_\alpha},$$

which Sim evaluates from the known $\alpha_t, \beta_t, \eta_\alpha$ and the public Y, without ever using s_t. The twin commitments are computed as $\bar{D}_t = D_t^{\eta_\alpha}$ and $\bar{E}_t = E_t^{\eta_\alpha}$, and the accompanying DLEQ proof π_t is produced by the honest-verifier zero-knowledge simulator of Lemma 4.7. Here Sim treats the Bi-DLEQ oracles abstractly, as the black box of Definition 4.6, by querying BinonceO and BiproveO and using their outputs directly. The embedding of the OMDL challenge into those oracles happens inside the reduction of Lemma 4.9 and it is invisible at the level of the present simulation, where the oracles are assumed to behave as specified.

(ii) PartialProofGen. The honest response of P_t is $z_t = d_t + \rho_t e_t + \lambda_t s_t c$, which Sim cannot compute directly because s_t is unknown. Instead, Sim recovers z_t in two pieces: the $\dot{x}$-dependent part is produced by a single BiproveO call, and the known remainder is added by

hand. Setting the Bi-DLEQ scalar to $\psi_t := \lambda_t \alpha_t$, the oracle returns

$$z_t^{(\mathrm{bi})} = d_t + \rho_t e_t + c\,\lambda_t \alpha_t\,\dot{x},$$

in which the unknown challenge $\dot{x}$ appears only in the last term, $c\lambda_t\alpha_t\dot{x}$. The remaining part of the honest response is the known base-g correction $c\lambda_t\beta_t$, which Sim adds itself:

$$z_t \leftarrow z_t^{(\mathrm{bi})} + c\,\lambda_t\beta_t = d_t + \rho_t e_t + c\,\lambda_t(\alpha_t\dot{x} + \beta_t) = d_t + \rho_t e_t + c\,\lambda_t s_t\ ,$$

where the last equality uses the $s_t = \alpha_t\dot{x} + \beta_t$ representation of Lemma 4.11. The result is exactly the honest response, so the simulated z_t is distributed identically to the real one.

(iii) Combine. This step is public and needs no further secret material. The aggregate evaluation satisfies

$$\gamma = \prod_{i\in S} \gamma_i^{\lambda_i} = h^{\sum_{i\in S}\lambda_i s_i} = h^s,$$

with the corrupt partial evaluations γ_j supplied by $\mathcal{F}$ from its known keys. Also, the group response is the public sum $z = \sum_{\ell\in S} z_\ell$, which the combiner now can form because the honest z_t that is not otherwise publicly computable was provided by step (ii) on top of the corrupt z_j coming from $\mathcal{F}$. Since the simulated z_t is identically distributed to the honest one and the aggregation is a deterministic public function of the per-party values, the combined proof (γ, c, z) and the output $\beta = H_3(\gamma)$ are distributed exactly as in a real execution.

Concurrency and the collision bound. Because Sim simulates only P_t, the family of simulated proofs is over a single party, and Sim neither rewinds $\mathcal{A}$ nor depends on the number of concurrently open sessions. Therefore, Sim runs $\mathcal{A}$ exactly once. The only way the transcript can veer from a real one is a programming collision (PC): a point Sim wishes to program in the H_2 table is already defined. This happens in two ways:

- through a proof Sim has previously simulated, contributing at most $\binom{n_{\mathsf{pr}}}{2}$ pairs, each colliding with probability at most $1/q$, since $R_t = g^{\bar{z}} Y_t^{-\bar{c}}$ is uniform;
- through a prior query of $\mathcal{A}$ to H_2, contributing at most $n_h n_{\mathsf{pr}}$ pairs, each colliding with probability at most $1/q$.

The per-proof challenge points $(g, h, Y_t, \gamma_t, \cdot, \cdot)$ that Sim programs are disjoint from the aggregate points $(g, h, Y, \gamma, \cdot, \cdot)$ because $Y_t \neq Y$, so no collision arises between the two kinds. Summing the two contributions,

$$\Pr[\mathrm{PC}] \le \left(\tbinom{n_{\mathsf{pr}}}{2} + n_h\, n_{\mathsf{pr}}\right)/q \le Q^2/q.$$

Conditioned on the complement of this event, the simulation by Sim is perfect, which gives the claimed bound. □

Algorithm 7 combines all of the reduction steps for Theorem 4.13.

Algorithm 7 Composed Reduction $\mathcal{R}'$ for Theorem 4.13

1: **Reduction to a single honest party.** Using Lemma 4.10, reduce to $|\mathrm{Cor}| = t - 1$ by self-corruption; designate the honest party P_t of the forgery quorum (factor κ).
2: **Challenge embedding and key extraction.** Receive $Y = g^{\dot{x}}$; run Lemma 4.11 to embed Y and extract $\{x_j\}_{j=1}^{t-1}$; obtain (α_t, β_t). **If** the KoE extractor fails, abort (cost $\mathsf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_{\mathsf{koe}})$).
3: **Straight-line simulation.** Simulate P_t by Lemma 4.12 where $\mathcal{F}$ runs $P_1, \ldots, P_{t-1}$. (Gap $\leq Q^2/q$, one run, no rewinding).
4: **Forgery.** $\mathcal{A}$ outputs $(\alpha^*, \gamma^*, c^*, z^*)$; abort if it does not verify or is trivial.
5: **Freshness dichotomy.** In the event where quorum size is insufficient ($\mathcal{E}_{\mathsf{quo}}$): general fork at the c^*-query. In the event of token substitution ($\mathcal{E}_{\mathsf{sub}}$): local-fork ρ_t with no slot guessing, solve the two-equation system.
6: **Discrete logarithm recovery.** Recover all challenge discrete logarithms arithmetically and submit to Fin, or output the KoE break if the extraction step is triggered.

We are finally in a position to prove the knowledge-soundness of the T-DLEQ proof system.

Theorem 4.13. *In the random oracle model, the non-interactive T-DLEQ proof system of Icy-DVRF is knowledge-sound, assuming the one-more discrete logarithm (OMDL) assumption for the proof system and the Schnorr knowledge-of-exponent (Schnorr-KoE) assumption for the distributed key generation, the latter implied by the discrete logarithm assumption in the algebraic group model. Under the fixed-quorum convention of the interactive proof, for which $\kappa = 1$ in Lemma 4.10, for every PPT adversary $\mathcal{A}$ corrupting up to $t - 1$ parties, there exist a Schnorr-KoE adversary $\mathcal{B}_{\mathsf{koe}}$ and OMDL adversaries $\mathcal{B}, \mathcal{B}'$ such that*

$$\begin{aligned}\mathsf{Adv}^{\text{t-dleq}}(\mathcal{A}) \;\leq\; & \mathsf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_{\mathsf{koe}}) \\ +\; 2\max\Bigg\{ & \sqrt{Q\big(\mathsf{Adv}^{\text{omdl}}_{\mathbb{G}}(\mathcal{B}) + 3(n+1)^2Q^2/q\big)}, Q\sqrt{2\big(\mathsf{Adv}^{\text{omdl}}_{\mathbb{G}}(\mathcal{B}') + 1/q\big)} \Bigg\} \;+\; \frac{Q^2}{q},\end{aligned}$$

where $Q = n_h + n_{\mathsf{pr}} + 1$ and n is the number of provers. The first maximand bounds the insufficient-quorum event $\mathcal{E}_{\mathsf{quo}}$ and the second bounds the token-substitution event $\mathcal{E}_{\mathsf{sub}}$. Without the fixed-quorum convention, the $\mathcal{E}_{\mathsf{sub}}$ term is multiplied by $\kappa \leq \min\{t,\, n-t+1\} \leq n$, which recovers, and never exceeds, the explicit-set bound.

Proof. We run the composed reduction $\mathcal{R}'$ of Algorithm 7, and assemble the bound in five steps below.

Step 1. By Lemma 4.11, the adversary $\mathcal{B}_{\mathsf{koe}}$ embeds the OMDL challenge as the group key Y and recovers the $t - 1$ corrupt shares as set out in that lemma. This step succeeds except with

probability at most $\mathsf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_{\mathsf{koe}})$. If successful, the honest party's key has the known affine representation $Y_t = Y^{\alpha_t} g^{\beta_t}$ with $s_t = \alpha_t \dot{x} + \beta_t$, where $\dot{x} = \log_g Y$ is carried symbolically. This is the only place the AGM is used, and it is confined to the distributed key generation phase.

Step 2. By Lemma 4.12, the simulator answers every message of the honest party P_t in a single run of $\mathcal{A}$, without rewinding and without access to the secret share, and the simulated transcript is identically distributed to a real one except on a programming collision event of probability at most Q^2/q. Unless that collision occurs, the simulation is perfect, and the probability of $\mathcal{A}$ producing a forgery equals its advantage in the real game. Lemma 4.10 further reduces a general adversary to one of the single-honest-party form at the cost of the factor κ, which is 1 under the fixed-quorum convention. We adopt this convention so that the displayed bound has $\kappa = 1$, and restore the general factor in Step 5.

Step 3. Suppose $\mathcal{A}$ outputs a verifying, non-trivial forgery $(\alpha^*, \gamma^*, c^*, z^*)$. Uniqueness of the VRF output forces $\gamma^* = (h^*)^s$, so the forgery is a fresh *proof object* for an output that is already fixed. The verifier recomputes the aggregate commitments $R^* = g^{z^*} Y^{-c^*}$ and $\bar{R}^* = (h^*)^{z^*} (\gamma^*)^{-c^*}$, which are exactly the values $T, \bar{T}$ formed in the Fin stage of the Bi-DLEQ game (Algorithm 6). For a verifying forgery the challenge satisfies $c^* = H_2(g, h^*, Y, \gamma^*, R^*, \bar{R}^*, \alpha^*)$, so $(R^*, \bar{R}^*)$ are exactly the commitments the forged challenge binds. Therefore, *freshness* is defined in terms of this aggregate $(R^*, \bar{R}^*)$, which a forgery can genuinely vary, rather than the fixed pair (α^*, γ^*). Since $t - |\mathrm{Cor}| = 1$, the winning condition

$$|\{k : (k, \cdot) \in \mathsf{ctr}(\alpha^*, R^*, \bar{R}^*)\}| < t - |\mathrm{Cor}|$$

requires fewer than one recorded honest slot, that is, none at all, meaning P_t made no genuine contribution to the forged aggregate. In other words, either P_t's published token does not appear in the aggregate at all, or it appears but was driven on an aggregate P_t never authorized. These two cases are mutually exclusive, and they cover all possibilities: if neither held, P_t's token would appear and would carry a genuine P_t response, so P_t would be a true contributor. Together with the $t-1$ corrupt parties, a full quorum would then hold consistent contributions, which by correctness forces the honest proof and contradicts non-triviality. Hence, a winning forgery falls into exactly one of the two disjoint events

- *Insufficient quorum* ($\mathcal{E}_{\mathsf{quo}}$): P_t did not contribute to $(R^*, \bar{R}^*)$.
- *Token substitution* ($\mathcal{E}_{\mathsf{sub}}$): P_t's published token was reused in an aggregate that it never authorized.

Because the two events require different forking strategies, a general fork for $\mathcal{E}_{\mathsf{quo}}$ and a local one for $\mathcal{E}_{\mathsf{sub}}$, we bound each one separately and take the maximum. Since the events are disjoint, $\mathsf{Adv}^{\text{t-dleq}}(\mathcal{A}) \le \Pr[\mathcal{E}_{\mathsf{quo}}] + \Pr[\mathcal{E}_{\mathsf{sub}}]$, and hence

$$\mathsf{Adv}^{\text{t-dleq}}(\mathcal{A}) \le 2 \max\{\Pr[\mathcal{E}_{\mathsf{quo}}], \Pr[\mathcal{E}_{\mathsf{sub}}]\} \ .$$

Step 4. We perform extraction in each branch separately.

Branch $\mathcal{E}_{\mathsf{quo}}$. In this event, P_t did not contribute, so the forged response z^* is not a BiproveO output. Thus, the random-oracle query $H_2(g, h^*, Y, \gamma^*, R^*, \bar{R}^*, \alpha^*)$ whose answer is c^* is a genuine forking point, not one programmed by the simulator. Forking at that point and extracting $\log_g Y$ is exactly the reduction of Lemma 4.9, so

$$\Pr[\mathcal{E}_{\mathsf{quo}}] \le \sqrt{Q\big(\mathsf{Adv}^{\mathrm{omdl}}_{\mathbb{G}}(\mathcal{B}) + 3(n+1)^2Q^2/q\big)}.$$

Branch $\mathcal{E}_{\mathsf{sub}}$. In this branch, P_t's token is reused in an aggregate for which it was never authorized. By Lemma 4.8, given the base-g verification, the base-h verification equation is redundant, so the extraction is single-base. The reduction $\mathcal{B}'$ plays OMDL and embeds two kinds of challenge: the key $Y = g^{\dot{x}}$, and a fresh $E_t \leftarrow V_t = g^{v_t}$ in every binonce of P_t. Its other half $D_t = g^{a_t}$ is self-sampled with a_t known, and the twins are supplied by Lemma 4.8. The E-component carries the challenge rather than D_t because only the E_t factor is weighted by the binding ρ_t: in the aggregate $R^* = \prod_i D_i E_i^{\rho_i}$, the half D_t enters unweighted while E_t enters as $E_t^{\rho_t}$, so resampling ρ_t perturbs the aggregate through E_t alone and leaves D_t untouched. Keeping $D_t = g^{a_t}$ known lets $\mathcal{B}'$ answer P_t's genuine signing query without a further Dlog call, while the unknown $v_t = \log_g E_t$ is the term that the fork exposes. $\mathcal{B}'$ is not required to guess the slot, since by Lemma 4.10 the only honest slot is t; this is where the single-honest-party convention is useful. Because the binding $\rho_t = H_2(t, \alpha^*, B)$ contributes to the aggregate $R^* = \prod_i D_i E_i^{\rho_i}$ *only* via the factor $E_t^{\rho_t}$, resampling the single random-oracle output ρ_t perturbs the aggregate in exactly that one factor,

$$R^{*\prime} = R^* \cdot E_t^{\rho_t' - \rho_t}, \qquad \bar{R}^{*\prime} = \varphi(R^{*\prime}),$$

where φ is the base-g-to-base-h map of Lemma 4.8, and the newly acquired challenge becomes a fresh $c^{*\prime}$. Subtracting the two verifying transcripts gives

$$z^* - z^{*\prime} = (\rho_t - \rho_t')\, v_t + (c^* - c^{*\prime})\, \dot{x}, \tag{4.1}$$

a single equation in the two unknowns $\dot{x} = \log_g Y$ and $v_t = \log_g E_t$. A single equation does not determine two unknowns, which is why *local* forking (the Local Forking Lemma of Bellare-Dai-Li [7]) rather than general forking is essential: local forking resamples only the ρ_t entry and leaves the genuine response that P_t gave in the session where it did contribute intact. The reduction answered that response with one Dlog query and so knows a second, independent linear relation

$$w_t = \rho_t^{(0)}\, v_t + \psi_t\, c^{(0)}\, \dot{x}, \tag{4.2}$$

where $\rho_t^{(0)} = H_2(t, \alpha^{(0)}, B^{(0)})$ and $c^{(0)}$ are the binding and challenge of that honest session, a *distinct* H_2 entry untouched by the fork. Equations (4.1) and (4.2) form a linear system in $(v_t, \dot{x})$ with determinant

$$\Delta = (\rho_t - \rho_t')\, \psi_t\, c^{(0)} - \rho_t^{(0)}\, (c^* - c^{*\prime}).$$

Since the four random-oracle outputs $\rho_t, \rho_t', c^{(0)}, c^* - c^{*\prime}$ are uniform and independent, $\Delta = 0$ with probability at most $1/q$. Whenever $\Delta \neq 0$, the system determines $\dot{x} = \log_g Y$ and v_t. It

remains to check that $\mathcal{B}'$ wins the OMDL game, which requires both $\ell < c_{\mathrm{id}}$ and a discrete logarithm for every challenge drawn. We adopt the normalization of Lemma 4.9 here as well, so that no binonce is consumed twice across the fork. Let $n'_{b,t}$ be the number of binonces of P_t created across the two executions of the local fork. $\mathcal{B}'$ draws one Chal for the key Y and one for the E-component of each binonce of P_t, so $c_{\mathrm{id}} = 1 + n'_{b,t}$. Every binonce of P_t costs one Dlog, spent either in the query that yields Equation (4.2) or, for a binonce left unconsumed when $\mathcal{A}$ terminates, in the cleanup step. Hence $\ell = n'_{b,t}$. The output tuple is complete: $\dot{x}$ and v_t come from the system, each remaining consumed v_j is recovered by substituting $\dot{x}$ into its own instance of Equation (4.2), and the unconsumed v_j are returned by the cleanup queries. Both requirements hold, so $\mathcal{B}'$ wins OMDL. Let $\mathrm{acc} = \Pr[\mathcal{E}_{\mathsf{sub}}]$ be the probability that $\mathcal{A}$ produces a forgery of this type, and frk the probability that the local fork on the entry $H_2(t, \alpha^*, B)$ returns a second verifying transcript. The Local Forking Lemma of [7], in the variant used for this purpose in [6], gives

$$\mathrm{frk} \;\geq\; \frac{\mathrm{acc}^2}{2Q^2}\,.$$

Whenever the fork succeeds, $\mathcal{B}'$ solves the system of Equations (4.1) and (4.2) and wins OMDL unless $\Delta = 0$, so $\mathrm{frk} \leq \mathsf{Adv}^{\mathrm{omdl}}_{\mathbb{G}}(\mathcal{B}') + 1/q$. Rearranging $\mathrm{acc}^2 \leq 2Q^2\big(\mathsf{Adv}^{\mathrm{omdl}}_{\mathbb{G}}(\mathcal{B}') + 1/q\big)$ gives the bound below, which is that of [6, Lem. 7] with the number of slots set to 1.

$$\Pr[\mathcal{E}_{\mathsf{sub}}] \leq Q\sqrt{2\big(\mathsf{Adv}^{\mathrm{omdl}}_{\mathbb{G}}(\mathcal{B}') + 1/q\big)}.$$

The single-base system above is exactly the one solved by the FROST1 local-forking reduction [6, Lem. 7], under the correspondence $d_i \mapsto \rho_i$, $H_{\mathrm{non}}(vk, lr, i) \mapsto H_2(i, \alpha, B)$, $n_s \mapsto 1$ (the slot forced to $m = t$), $q \mapsto Q$, and $p \mapsto q$. The three conditions of that lemma are satisfied in our case:

- $\rho_\ell = H_2(\ell, \alpha, B)$ is keyed by the index ℓ and enters the aggregate only through $E_\ell^{\rho_\ell}$, so resampling ρ_t localizes to a single factor.
- $H_2(t, \alpha^*, B)$ is a distinct, identifiable table entry, so the single-output resampling condition of [7] holds.
- The OMDL challenge is embedded in the E-component of P_t, so after Lemma 4.8 removes the twin equation, the residual system is precisely the one [6, Lem. 7] solves, with target $\log_g Y$.

Therefore, the cited constant transfers with the slot factor set to 1.

Step 5. Combining the setup loss $\mathsf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_{\mathsf{koe}})$ in Step 1, the Step-2 simulation gap Q^2/q, and the per-branch bounds of Step 4 using the maxima of Step 3, we get

$$\begin{aligned}\mathsf{Adv}^{\text{t-dleq}}(\mathcal{A}) \;&\leq\; \mathsf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_{\mathsf{koe}}) \\ &+\; 2\max\left\{\sqrt{Q\big(\mathsf{Adv}^{\mathrm{omdl}}_{\mathbb{G}}(\mathcal{B}) + 3(n+1)^2Q^2/q\big)},\, Q\sqrt{2\big(\mathsf{Adv}^{\mathrm{omdl}}_{\mathbb{G}}(\mathcal{B}') + 1/q\big)}\right\} \;+\; \frac{Q^2}{q}.\end{aligned}$$

Reinstating the general factor of Lemma 4.10 requires multiplying the $\mathcal{E}_{\mathsf{sub}}$ term by κ, which is 1 under the fixed-quorum convention and at most $\min\{t, n-t+1\} \leq n$ in general. Finally, since Schnorr-KoE is implied by the discrete logarithm assumption in the algebraic group model, the distributed key generation introduces no assumption beyond the discrete logarithm assumption and the algebraic group model, and the proof system itself rests on OMDL. □

Remark 4.14. Two random-oracle roles appear in the proof, and only the first separates the FROST variants introduced in [21, 6]. The bindings $\rho_\ell = H_2(\ell, \alpha, B)$ are *per-prover* and tied to the ordered set B: this is the original FROST design, later named FROST1, which Icy-DVRF uses and never replaces by the single shared binding of FROST2. This choice is what fixes the constants in Theorem 4.13. In the $\mathcal{E}_{\mathsf{quo}}$ branch, a single proving query evaluates up to n per-prover bindings in addition to the one challenge, so the honest collision count carries the factor $(n+1)^2$, matching the FROST1 reduction of [6, Lem. 6]. Equivalently, one may absorb the factor by defining $Q := n_h + (n+1)n_{\mathsf{pr}} + 1$ so that internal binding evaluations are counted directly. We keep the explicit $(n+1)^2$ for transparency. In the $\mathcal{E}_{\mathsf{sub}}$ branch, the per-prover binding is exactly what makes token substitution extractable by the local fork of [7], and the resulting bound is the FROST1 single-base local-forking bound of [6, Lem. 7] with the slot factor specialized to 1 by the convention of Lemma 4.10. The OMDL-based intermediate assumption is independent of the choice of binding: it was introduced for FROST2 in [21, 6] and applies verbatim to our case.

Remark 4.15. The non-triviality condition of Theorem 4.13, that fewer than $t - |\mathrm{Cor}|$ honest provers contributed to the *exact* aggregate $(R^*, \bar{R}^*)$, is the VRF analogue of the strongest unforgeability level, TS-SUF-3, of the threshold-signature hierarchy of [6]. Because the VRF output is unique, the strong-unforgeability target of a "new signature on an already-signed message" is vacuous here: there is only one valid output per input. What our reduction rules out instead is a new *proof object*, namely a fresh aggregate commitment pair, so freshness is defined in terms of $(R^*, \bar{R}^*)$ rather than (α^*, γ^*). This level 3 guarantee is precisely resistance to *token substitution* (the event $\mathcal{E}_{\mathsf{sub}}$): the per-prover binding prevents an honest prover's published commitment from being reused in a set it did not authorize. The single-shared-binding optimization of FROST2 would weaken this to level 2 [6, Fig. 8], which is the concrete reason Icy-DVRF keeps the per-prover binding of Remark 4.14.

4.5.5 Security Tightness and Parameter Recommendations

The security reduction for the interactive version Icy-DVRF presented in Section 4.5.3 relies on the Generalized Forking Lemma, which inherently implies a non-tight reduction. Specifically, the advantage ϵ' of solving the Discrete Logarithm Problem (DLP) is bounded by

$$\epsilon' \leq \frac{\epsilon^2}{2n_h + \Omega \cdot n_p} ,$$

where Ω denotes the preprocess batch size, n_h represents the number of hash queries and $\Omega \cdot n_p$ accounts for the total number of preprocessed commitment pairs. The preprocessing phase introduces specific constraints on simulation complexity, as the simulator must program the random oracle for every pre-published commitment $D_{i,j}$ and $E_{i,j}$. This requirement increases the denominator in the advantage equation, leading to a quadratic loss that slightly loosens the security bound compared to standard, non-preprocessed DLEQ proofs.

To mitigate this effect and preserve a 128-bit security level, we recommend using elliptic curves with at least a 256-bit prime order q. Under these parameters, the security margin remains robust for decentralized applications even when accounting for up to 2^{60} oracle queries, ensuring the protocol's practical viability despite the increased complexity of the reduction.

In the variant of Icy-DVRF described in Section 4.2, the reduction is also non-tight: it rewinds once through a forking step, incurring a quadratic loss. Inverting the dominant maximand of Theorem 4.13, if OMDL is $(\tau', \epsilon'_{\text{omdl}})$-hard and Schnorr-KoE is $(\tau', \epsilon'_{\text{koe}})$-hard in $\mathbb{G}$, then the non-interactive T-DLEQ proof system over $\mathbb{G}$ with n provers, threshold t, and preprocess batch size Ω is knowledge-sound with forgery advantage ϵ bounded by

$$\epsilon \;\leq\; \epsilon'_{\text{koe}} \;+\; 2\sqrt{Q\Big(\epsilon'_{\text{omdl}} + \frac{3(n+1)^2Q^2}{q}\Big)} \;+\; \frac{Q^2}{q},$$

where $Q = n_h + n_{\mathsf{pr}} + 1$ counts the random-oracle (n_h) and proving (n_{pr}) queries. Equivalently, solving OMDL succeeds with advantage at least

$$\epsilon'_{\text{omdl}} \;\gtrsim\; \frac{\epsilon^2}{4Q} - \frac{3(n+1)^2Q^2}{q},$$

the same quadratic ϵ^2/Q shape as in the bound of the interactive variant, now also featuring the FROST1 internal-binding constant $(n+1)^2$ (Remark 4.14) in place of the interactive denominator's preprocessing term.

The preprocess batch size Ω does not appear in the later bounds, since, unlike the interactive analysis where each of the Ω pairs per round is simulated and forked over, the non-interactive reduction charges only the pairs an adversary *consumes*, which are counted directly by n_{pr}.

To preserve a 128-bit security level against up to 2^{60} oracle queries, we again recommend a curve with at least a 256-bit prime order q, under which the quadratic loss and the factor of $(n+1)^2$ leave a comfortable margin for the committee sizes considered in Section 4.6.

4.6 Comparison

Before Chapter 6 where we make an in depth comparison, we briefly state the improvements of Icy-DVRF against its predecessor DVRFwCP, and where relevant, GLOW-DVRF, along three axes: mathematical operation counts, communication bandwidth, and theoretical on-

chain gas cost. We instantiate all protocols using the BLS12-381 curve, a Type-3 pairing-friendly curve widely adopted in blockchain ecosystems and standardized for Ethereum in EIP-2537 [53]. This curve satisfies the Symmetric External Diffie-Hellman (SXDH) assumption, implying that the DDH problem (Definition 2.1) is hard in both source groups $\mathbb{G}_1$ and $\mathbb{G}_2$. To optimize for gas efficiency, we map the underlying group $\mathbb{G}$ to the subgroup $\mathbb{G}_1$, since element representations are smaller and group operations are significantly cheaper than in $\mathbb{G}_2$.

4.6.1 Mathematical Comparison

By incorporating FROST-style preprocessing, Icy-DVRF outperforms DVRFwCP in both proof generation complexity and communication cost. This advantage necessitates a one-time post-DistKeyGen computation, a requirement not present in DVRFwCP. In the PartialEval phase, DVRFwCP is slightly more computationally efficient, requiring 3 exponentiations in $\mathbb{G}$ compared to 5 for Icy-DVRF. However, for a (t, n)-threshold DVRF proof generation, in the PartialProofGen phase, Icy-DVRF demonstrates superior scalability by requiring only $3t$ exponentiations per party, whereas DVRFwCP is computationally heavier with $3nt + 4n + 3t$ exponentiations due to the overhead of its nonce generation mechanism. Finally, the trade-off shifts back in favor of DVRFwCP during the Combine phase, where it requires $6t$ exponentiations and $2t$ multiplications, notably less than Icy-DVRF, which demands $9t$ exponentiations, $8t - 3$ multiplications, and additional hashing operations. Overall, for verifiable randomness and proof generation, Icy-DVRF is much lower in computational complexity than DVRFwCP. The analysis is detailed in Table 4.1.

4.6.2 Communication Cost

A critical distinction arises in the communication overhead during the evaluation and proof generation phases, specifically regarding the broadcast of group elements versus scalar elements. During the PartialEval phase, DVRFwCP is more bandwidth-efficient, requiring participants to broadcast a partial evaluation and an NIZK proof for a total of $1\mathbb{G} + 2\mathbb{Z}_q$ elements. In contrast, Icy-DVRF requires participants to broadcast the partial evaluation along with two additional commitments $(\bar{D}_i, \bar{E}_i)$ to facilitate the subsequent single-round proof generation, resulting in a higher initial payload of $3\mathbb{G} + 2\mathbb{Z}_q$ elements. However, the bandwidth trade-off reverses significantly during the PartialProofGen phase. To generate the shared challenge in DVRFwCP, participants must collectively run Augmented Secure-DKG, which requires broadcasting polynomial commitments proportional to the threshold t, transmitting $2t$ group elements along with the proof share, for a total of $2t\mathbb{G} + 2\mathbb{Z}_q$ per participant. Conversely, Icy-DVRF participants compute the shared challenge locally using the pre-broadcast commitments, so the only broadcast required is the additive proof share and the hash, totaling merely 2 elements of $\mathbb{Z}_q$. Consequently, while Icy-DVRF incurs a slightly higher constant cost during PartialEval, it is significantly more scalable for large

Table 4.1: Mathematical Operations of DVRFwCP and Icy-DVRF Schemes.

	DVRFwCP	**Icy-DVRF**
DistKeyGen	$2nt + 3n + 2t$ exp. in $\mathbb{G}$ $2nt + t - 1$ mul. in $\mathbb{G}$	$nt + 3n - 1$ exp. in $\mathbb{G}$ $nt + n - t - 1$ mul. in $\mathbb{G}$ n hash to $\mathbb{Z}_q$
Preprocess	–	2Ω exp. in $\mathbb{G}$
PartialEval	3 exp. in $\mathbb{G}$ 1 hash to $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$	5 exp. in $\mathbb{G}$ 1 hash to $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$
EvalCombine[(a)]	$5t$ exp. in $\mathbb{G}$ $3t - 1$ mul. in $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$	–
PartialProofGen	$3nt + 4n + 3t$ exp. in $\mathbb{G}$ $3nt + t - 2$ mul. in $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$	$3t$ exp. in $\mathbb{G}$ $3t - 3$ mul. in $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$
Combine[(b)]	$6t$ exp. in $\mathbb{G}$ $2t$ mul. in $\mathbb{G}$ 1 hash to $\{0,1\}^\Lambda$	$9t$ exp. in $\mathbb{G}$ $8t - 3$ mul. in $\mathbb{G}$ 1 hash to $\{0,1\}^\Lambda$ $t + 2$ hash to $\mathbb{Z}_q$
Verify	4 exp. in $\mathbb{G}$ 2 mul. in $\mathbb{G}$ 1 hash to $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$ 1 hash to $\{0,1\}^\Lambda$	4 exp. in $\mathbb{G}$ 2 mul. in $\mathbb{G}$ 1 hash to $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$ 1 hash to $\{0,1\}^\Lambda$
Interactions[(c)]	2 rounds	1 round
Proof size	1 $\mathbb{G}$ element 2 $\mathbb{Z}_q$ elements	1 $\mathbb{G}$ element 2 $\mathbb{Z}_q$ elements

(a) These algorithms are not present in the corresponding DVRFs.
(b) The corresponding algorithm is named ProofCombine in DVRFwCP.
(c) The one-time DistKeyGen is not included.

distributed committees: the total communication cost of DVRFwCP scales linearly with the threshold ($(2t + 1)\mathbb{G}$), whereas Icy-DVRF maintains a constant communication overhead of $3\mathbb{G}$ elements per participant, regardless of the committee size. The results are summarized in Table 4.2.

Table 4.2: Per Participant Communication Cost of DVRFwCP and Icy-DVRF.

Phase	**DVRFwCP**	**Icy-DVRF**
PartialEval	$1\mathbb{G} + 2\mathbb{Z}_q$	$3\mathbb{G} + 2\mathbb{Z}_q$
PartialProofGen	$2t\mathbb{G} + 2\mathbb{Z}_q$	$0\mathbb{G} + 2\mathbb{Z}_q$
Total Broadcast	$(2t + 1)\mathbb{G} + 4\mathbb{Z}_q$	$3\mathbb{G} + 4\mathbb{Z}_q$

4.6.3 Theoretical Gas Cost

Both DVRFwCP and Icy-DVRF achieve constant-time, pairing-free verification, and their Verify algorithms are structurally identical, so their theoretical on-chain costs coincide. A full four-scheme comparison, including DDH-DVRF and GLOW-DVRF, together with the empirical validation on the Sepolia testnet, is presented in Chapter 6.

To close this chapter, we note that Icy-DVRF thus achieves the central improvement stated at the beginning of the chapter: it retains the constant-size, pairing-free proof and constant-time verification of DVRFwCP, while reducing the per-evaluation communication from $(2t + 1)\mathbb{G}$ to a committee-independent $3\mathbb{G}$ and the interaction from two rounds to one. This efficiency comes at a cost. Like FROST, Icy-DVRF adopts an abort-and-identify model rather than guaranteeing output in the presence of faults; the robustness implications of this design choice, together with the accountability mechanisms that make it viable in deployment, are the subject of Chapter 7. A second limitation is more fundamental to the present state of how VRFs and DVRFs are situated in a blockchain smart-contract setup: the DVRF output β becomes public as soon as the proof is verified on-chain. For applications such as on-chain gaming, in which the requester must learn the randomness before the rest of the network, this immediacy is itself the bottleneck, and it is the problem addressed in Chapter 5.

CHAPTER 5

ICYVEIL: OUTPUT PRIVACY VIA PAIRING-FREE BLINDING

5.1 Motivation and Positioning

In Chapter 4, we presented a distributed VRF with a compact, pairing-free proof and committee-independent communication. But this leaves one limitation unaddressed, and this limitation is tied to the blockchain behavior rather than incidental: as soon as an Icy-DVRF proof is verified on-chain, its output $\beta = H_3(\gamma)$ becomes public to the entire network. For a broad class of applications, this behavior is undesirable.

There is a practical limitation to the utility of smart-contract-based VRF services, especially in gaming services: the requester cannot submit the request in advance to have it ready when the online game begins, because the VRF output is immediately made public on the blockchain upon protocol completion. This means the request must be synchronized with VRF generation, introducing latency and halting gameplay. Also, since the VRF output is public, the requester is prohibited from reusing this VRF value in the future: they cannot use a pseudorandom number generator with the VRF output as the seed to generate multiple successive random values for later use. This forces gaming platforms to design their games so that players cannot exploit publicly available VRF inputs, ensuring that latency and cost overheads remain affordable.

To overcome this critical limitation, the concept of Output-Private VRFs (Pri-VRF) and Output-Private DVRFs (Pri-DVRFs) were introduced by the FlexiRand [40] framework, which are used in the Supra VRF service [52]. FlexiRand ensures that while the pseudo-randomness and public verifiability properties of VRFs are maintained, only the requester can observe the final output. It achieves this through a novel cryptographic blinding and unblinding mechanism: the user blinds their input before sending it to the network, and the smart contract verifies a blinded version of the final output. This allows randomness to be generated asynchronously and reused securely without exposing it to the public. However, the distributed instantiations of this privacy primitive are based on GLOW-DVRF, which rely on computationally expensive bilinear pairings, limiting their practical efficiency in gas-constrained blockchain environments.

In this work, we propose IcyVeil [4], an *output-private* distributed verifiable random function that fuses the output privacy of FlexiRand with the verification efficiency of Icy-DVRF. IcyVeil integrates blinding and unblinding operations directly into the pairing-free architecture of Icy-DVRF. Under our protocol, the user blinds their initial input with a private nonce, and the distributed VRF committee computes partial evaluations and Non-Interactive Zero-Knowledge (NIZK) proofs of equality of discrete logarithms over the blinded values. The aggregated *veiled* proof is verified efficiently without pairings and is subsequently *unveiled* exclusively by the requester. This partnership allows IcyVeil to deliver unpredictable, output-private randomness with constant-size proofs. Although Icy-DVRF, and hence IcyVeil, incurs an additional communication round among the DVRF clan compared to FlexiRand, the veiling/unveiling mechanism eliminates the latency handicap of this interaction, resulting in a highly scalable and cost-effective DVRF primitive for decentralized applications.

5.2 The Blinding Mechanism

The idea behind IcyVeil, and the component that distinguishes it from Icy-DVRF, is a pair of user-side operations, Blind and $\mathrm{Unblind}$, executed by the user U that request the blinded DVRF value, and built entirely from the same discrete-logarithm machinery used in this thesis without introducing pairings.

The mechanism is built on a single algebraic observation. Let $h = H_1(\alpha)$ be the formatted input and s the clan's shared secret key, so that the true (unblinded) evaluation is $\gamma = h^s$. If the user first raises h to a private scalar κ, producing $\hat{h} = h^\kappa$, and the clan evaluates on $\hat{h}$ instead, the result is

$$\hat{\gamma} = \hat{h}^{\,s} = h^{\kappa s} = (h^s)^\kappa = \gamma^\kappa.$$

The blinded output $\hat{\gamma}$ is therefore the true output γ raised to the user's secret κ. An on-chain observer sees only $(\hat{h}, \hat{\gamma}) = (h^\kappa, h^{\kappa s})$; recovering $\gamma = h^s$ from this pair without κ is precisely a Computational Diffie-Hellman instance. The user, however, knows κ and recovers the output by a single exponentiation, $\gamma = \hat{\gamma}^{\,\kappa^{-1}}$.

Two zero-knowledge proofs hold this mechanism together, both instantiated from the proof systems of Chapter 2. During blinding, the user proves knowledge of κ relative to $(h, \hat{h})$ using the Schnorr proof of Section 2.7.2, so that smart-contract and the VRF clan can verify and be sure the blinded input is well-formed rather than an arbitrary group element chosen to cunningly extract information. During unblinding, the user proves that the same κ relates the public pair $(h, \hat{h})$ and the recovered pair $(\gamma, \hat{\gamma})$, using the Chaum-Pedersen DLEQ proof of Section 2.7.3; this is what lets anyone confirm that the revealed γ is the genuine unblinding of the on-chain $\hat{\gamma}$ without the user ever disclosing κ. The two exclusively user-side algorithms are as follows.

Blind:

On input $h \in \mathbb{G}$:

1. Choose a random secret value $\kappa \xleftarrow{\$} \mathbb{Z}_q$.
2. Compute $\hat{h} := h^{\kappa}$.
3. Compute a NIZK proof $\hat{\pi} \leftarrow \mathrm{ProveExp}_{H_2}\left(h, \hat{h}; \kappa\right)$.
4. Output $(\hat{h}, \hat{\pi})$.

Unblind:

On input the blinded proof $\Pi = (\hat{\gamma}, c, \hat{z})$:

1. Check whether $\mathrm{PreVerify}(\hat{\gamma}, c, \hat{z}) = 1$. If not, abort.
2. Use κ to unblind $\hat{\gamma}$, i.e., $\gamma = (\hat{\gamma})^{\kappa^{-1}}$.
3. Generate $\pi_{\text{eq}} = \mathrm{ProveEq}_{H_2}(h, \gamma, \hat{h}, \hat{\gamma}; \kappa)$, proving that $\log_h \hat{h} = \kappa = \log_{\gamma} \hat{\gamma}$.
4. Compute the final DVRF output $\beta = H_3(\gamma)$.
5. Output $(\beta, \gamma, \pi_{\text{eq}})$.

5.3 The Smart Contract Framework

We use a smart contract framework similar to FlexiRand [40] for IcyVeil, which comprises three entities: the **User**, the **Smart Contract**, and the **VRF Clan**. The User requests randomness and is the sole party that learns the output; the Smart Contract formats inputs, verifies blinding and the blinded proof, and stores the blinded result; and the VRF Clan is the threshold committee of Chapter 4 that performs the distributed evaluation.

In the baseline framework, the interaction between the three entities proceeds in a single, direct pass: The User initiates a request by submitting an initial input α to the Smart Contract. The Smart Contract builds the VRF input h by combining α with metadata such as a block hash, a unique nonce, the chain identifier, and the requester's address; this formatting both fixes the input and prevents replay or pre-computation attacks. The VRF Clan then fetches h from the contract, and each node computes a partial evaluation of h under its secret-key share together with a proof of correct evaluation. Once at least t valid partial evaluations are available, they are combined into the final output γ and an accompanying proof Π, which the VRF clan posts back to the Smart Contract. The contract runs Verify on (γ, Π) and, if the check succeeds, records the output and invokes the requester's callback function with γ. The defining characteristic of this baseline flow is that verification and disclosure occur in the

same on-chain step: the moment the proof is verified, the output γ is written to the public ledger and is visible to the entire network.

IcyVeil decouples these two steps. The user interposes the blinding step of Section 5.2 between formatting and evaluation. The user fetches h and blinds it to $\hat{h}$; the clan then evaluates on $\hat{h}$ and produces the blinded proof $\Pi = (\hat{\gamma}, c, \hat{z})$. The consequence for the on-chain logic is that verification must now be split into two algorithms that the baseline combined into one. The smart-contract can, and must, check that the blinded proof was computed correctly with respect to the VRF clan's public key, which is the role of $\mathrm{PreVerify}$. The smart-contract performs this pre-verification *without* learning the output, since $\hat{\gamma}$ remains blinded. Recovery of the actual output is deferred to $\mathrm{Unblind}$, which only requesting user U can run, since only they hold κ. This is the reason $\mathrm{PreVerify}$ and $\mathrm{Unblind}$ are separate algorithms: the first establishes the public correctness of the veiled proof on-chain, the second performs the private disclosure off-chain, and the final Verify links the two through the unblinding DLEQ proof π_{eq}.

This framework prevents replay attacks. The smart contract carefully constructs h from a context string that reflects both the owner's identity and other environment-related nonces, so that forging ownership of the initial input is intractable.

The complete IcyVeil message flow, illustrated in Figure 5.1, proceeds in the following nine steps.

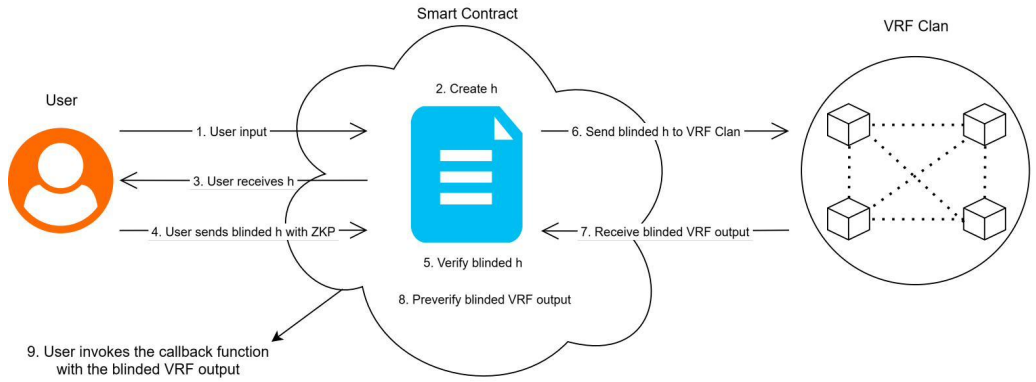

Figure 5.1: Message Flow for IcyVeil via Smart Contract.

1. **User** initiates the process, sending an initial input α to **Smart Contract**.
2. **Smart Contract** processes this input by combining it with metadata like block hashes and nonces to create a formatted VRF input, referred to as h.
3. **User** fetches h from **Smart Contract**.

4. **User** *blinds* h by applying a secret random masking factor κ. They then send this blinded h, named $\hat{h}$, back to the **Smart Contract** along with a NIZK proof of knowledge of κ. This proof allows the service to verify that the blinding was done correctly.

5. **Smart Contract** verifies the zero-knowledge proof of correct blinding. If verification fails, **Smart Contract** halts the process.

6. **VRF Clan**, which is a committee of nodes, fetches $\hat{h}$ from **Smart Contract**.

7. **VRF Clan** performs the following.

   (a) Each node in **VRF Clan** verifies the blinding ZKP.

   (b) If valid, **VRF Clan** proceeds to generate Icy-DVRF output on input of $\hat{h}$.

   (c) Upon success, **VRF Clan** submits the blinded output $\hat{\gamma}$ to **Smart Contract**.

8. **Smart Contract** performs a "pre-verification" of the blinded output. Because the output remains blinded, the contract can verify that it was computed correctly using the **VRF Clan**'s group public key without knowing the random value. If this pre-verification evaluates to false, **Smart Contract** halts the process.

9. **Smart Contract** stores the blinded DVRF output $\hat{\gamma}$ along with its proof components, i.e., $\Pi = (\hat{\gamma}, c, \hat{z})$.

At the end of this procedure, the **User** can use their secret masking factor to *unblind* the value they received from the **Smart Contract**, finally obtaining the actual DVRF output for their private use. During the unblinding process, the **User** generates an NIZK proof of discrete logarithm equality, proving the integrity of the blinding and unblinding operations. When **User** wants to use the unblinded DVRF output, they have the option to publicize this NIZK proof for verification. While the user could alternatively reveal the secret masking factor to verify the computation, the NIZK proof is the preferred method for maintaining privacy.

5.4 Protocol Definition

In the interest of presenting a self-contained algorithm, the notation and setup fundamental to the construction are detailed below. Let $(\mathbb{G}, g, q)$ be a multiplicative group where the DDH assumption holds. Let $H_1 : \{0,1\}^* \to \mathbb{G}$, $H_2 : \{0,1\}^* \to \mathbb{Z}_q$ and $H_3 : \{0,1\}^* \to \{0,1\}^\Lambda$ be hash functions, where Λ denotes the desired output length. Additionally, assume that we have a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of participants, the *DVRF clan*, who communicate using the secure implementation practices described earlier. Also, assume that we have a user U who requests the blinded DVRF value and is likely not in $\mathcal{P}$. Then, let $\mathcal{V}^{\mathrm{IcyVeil}}$ be defined using the following nine algorithms, where the algorithms Blind and $\mathrm{Unblind}$ are called only by U. We envision that the DVRF clan is formed before U submits a request.

The first two of the nine algorithms, $\mathrm{DistKeyGen}$ and $\mathrm{Preprocess}$, are identical to their Icy-DVRF counterparts in Sections 4.2.2 and 4.3 and are not reproduced here; the VRF clan's key material and preprocessed nonce batches are generated exactly as in Chapter 4. The two user-side algorithms, Blind and $\mathrm{Unblind}$, were presented in Section 5.2. The remaining five algorithms, in which IcyVeil differs from Icy-DVRF by operating on the blinded input $\hat{h}$ throughout, are specified below.

$\mathrm{PartialEval}$:

Let C be the combiner (who might be chosen amongst the participants). Let S be the set of θ participants selected for the generation of the DVRF value, where $t \leq \theta \leq n$, and let Y be the group public key set during the DistKeyGen phase.

1. C sends $\left(\left(\alpha, h, \hat{h}, \hat{\pi}\right), j, B\right)$ to $P_i,\ i \in S$, where $\left(\alpha, h, \hat{h}, \hat{\pi}\right)$ is the request, B is the ordered list of parties in S, and j is the index of the selected single-use commitments from the Preprocess stage.
2. Each P_i performs the following.
   (a) Checks if B is valid. If this check fails, abort.
   (b) Checks if $\mathrm{VerifyExp}_{H_2}\left(h, \hat{h}; \hat{\pi}\right) = 1$. If this check fails, abort.
   (c) Computes partial evaluation $\hat{\gamma}_i = \hat{h}^{s_i} \in \mathbb{G}$.
   (d) Computes partial evaluation proof $\hat{\pi}_i \leftarrow \mathrm{ProveEq}_{H_2}\left(g, \hat{h}, Y_i, \hat{\gamma}_i; s_i\right)$.
   (e) Computes $\bar{D}_i = \hat{h}^{d_{i,j}}$ and $\bar{E}_i = \hat{h}^{e_{i,j}}$.
   (f) Outputs $(\hat{\gamma}_i, \hat{\pi}_i, \bar{D}_i, \bar{E}_i)$.

$\mathrm{PartialProofGen}$:

Now that each participant P_i published $\left(\hat{\gamma}_i, \hat{\pi}_i, D_i, E_i, \bar{D}_i, \bar{E}_i\right)$, each P_i produces their partial DVRF proof, using the following procedure.

1. Compute the set of binding values $\rho_\ell = H_2(\ell, \alpha, B)$ for all $\ell \in S$.
2. Compute $k_i = d_i + \rho_i \cdot e_i$.
3. Compute the (public) group commitments

$$R = \prod_{\ell \in S} D_\ell \cdot (E_\ell)^{\rho_\ell} = g^k, \qquad \bar{R} = \prod_{\ell \in S} \bar{D}_\ell \cdot \left(\bar{E}_\ell\right)^{\rho_\ell} = \hat{h}^k,$$

   where $k = \sum_{\ell \in S} k_\ell$, unknown to individual parties.
4. Compute $\hat{\gamma} = \prod_{\ell \in S} \hat{\gamma}_\ell^{\lambda_\ell}$, where λ_ℓ is the ℓ^{th} Lagrange coefficient in S.
5. Compute the group challenge $c = H_2\left(g, \hat{h}, Y, \hat{\gamma}, R, \bar{R}, \alpha\right)$.

6. Computes their response using their long-lived partial secret key s_i by computing the partial proof $\hat{z}_i = k_i + \lambda_i \cdot s_i \cdot c = d_i + \rho_i \cdot e_i + \lambda_i \cdot s_i \cdot c$, which is in additive form, where λ_i is the i^{th} Lagrange coefficient in S.

7. Securely deletes d_i, e_i from their local storage, and broadcasts $(\hat{z}_i, c)$.

Combine:

Upon receiving $(\hat{z}_\ell, c)$ from P_ℓ for $\ell \in S$, the combiner C performs the following actions.

1. Computes the following.[1]
   (a) $\rho_\ell = H_2(\ell, \alpha, B)$ for $\ell \in S$.
   (b) $R_\ell = D_\ell \cdot (E_\ell)^{\rho_\ell}$ for $\ell \in S$.
   (c) $\bar{R}_\ell = \bar{D}_\ell \cdot \left(\bar{E}_\ell\right)^{\rho_\ell}$ for $\ell \in S$.
   (d) The group commitment $R = \prod_{\ell \in S} R_\ell$.
   (e) The group commitment $\bar{R} = \prod_{\ell \in S} \bar{R}_\ell$.
   (f) $\hat{\gamma} = \prod_{\ell \in S} \hat{\gamma}_\ell^{\lambda_\ell}$, where λ_ℓ is the ℓ^{th} Lagrange coefficient in S.
   (g) The group challenge $c = H_2\left(g, \hat{h}, Y, \hat{\gamma}, R, \bar{R}, \alpha\right)$.

2. Verifies the following.
   (a) Verifies that each c received from P_ℓ matches the c value the combiner computed in Step 1(g) above.
   (b) Verifies each partial evaluation proof by checking

$$\text{VerifyEq}_{H_2}\left(g, \hat{h}, Y_\ell, \hat{\gamma}_\ell, \hat{\pi}_\ell\right) \stackrel{?}{=} 1$$

   for each $(\hat{\gamma}_\ell, \hat{\pi}_\ell), \ell \in S$, where Y_ℓ is the partial public key of P_ℓ.
   (c) Verifies the validity of the response of $P_\ell, \ell \in S$, by checking

$$g^{\hat{z}_\ell} \stackrel{?}{=} R_\ell \cdot Y_\ell^{c \cdot \lambda_\ell}$$
$$\hat{h}^{\hat{z}_\ell} \stackrel{?}{=} \bar{R}_\ell \cdot \hat{\gamma}_\ell^{c \cdot \lambda_\ell}$$

   for each $\hat{z}_\ell, \ell \in S$.

   If any of these checks fail for a participant, identify and report the misbehaving participant. At the end of this step, if at least t out of the θ members of S are not qualified, abort the protocol. Otherwise, continue.

3. Compute the group response $\hat{z} = \sum_{\ell \in S} \hat{z}_\ell$.

4. Output the core DVRF value $\hat{\gamma}$ and the blinded proof as $\Pi = (\hat{\gamma}, c, \hat{z})$.

[1] This step is optional if C is a participant.

$\mathrm{PreVerify}$ (anyone can execute this algorithm):

Upon receiving Π, the Smart Contract executes the pre-verification algorithm to take the input $\Pi = (\hat{\gamma}, c, \hat{z})$ and verify that $\hat{\gamma}$ and $\hat{z}$ were created properly for the input $\hat{h}$ by performing the following steps.

1. Compute $T = g^{\hat{z}} \cdot Y^{-c}$.
2. Compute $\bar{T} = \hat{h}^{\hat{z}} \cdot \hat{\gamma}^{-c}$.
3. Check if $c \stackrel{?}{=} H_2\left(g, \hat{h}, Y, \hat{\gamma}, T, \bar{T}, \alpha\right)$. If the check is satisfied, output 1; otherwise, output 0.

Verify (anyone can execute this algorithm):

Given the group public key Y, the blinded proof $\Pi = (\hat{\gamma}, c, \hat{z})$, and the Unblind output $(\beta, \gamma, \pi_{\text{eq}})$, where all parties have access to $\hat{h}, \hat{\gamma}$, perform verification as follows.

1. $h \leftarrow H_1(\alpha)$.
2. Check whether $\beta = H_3(\gamma)$.
3. Check whether $\mathrm{PreVerify}(\hat{\gamma}, c, \hat{z}) = 1$. If not, abort.
4. Check if $\mathrm{VerifyEq}_{H_2}(h, \gamma, \hat{h}, \hat{\gamma}, \pi_{\text{eq}})$, guaranteeing $\log_h(\hat{h}) = \log_\gamma(\hat{\gamma})$.

5.5 Security

The security of IcyVeil is based on three hardness assumptions outlined in Section 2.3, each capturing a distinct aspect of the protocol. First, the **Decisional Diffie-Hellman (DDH)** assumption (Definition 2.1) upholds the pseudorandomness of the base Icy-DVRF construction: it ensures that the blinded partial evaluations and the group challenge computed during $\mathrm{PartialProofGen}$ are computationally indistinguishable from random group elements, hence no adversary can predict or bias the DVRF output. Because IcyVeil's $\mathrm{PartialProofGen}$ and $\mathrm{Combine}$ are structurally those of Icy-DVRF applied to the blinded input $\hat{h}$, the pseudorandomness and uniqueness arguments of Section 4.5.2 carry over unchanged; we therefore focus here on the two properties that are genuinely new to the output-private setting, namely output privacy and unforgeability. That inheritance is conditional rather than direct: uniqueness and strong pseudorandomness transfer from DDH-DVRF once the aggregated T-DLEQ proof is established as a NIZK argument of knowledge, and the knowledge-soundness half of that step rests in turn on the one-more discrete logarithm assumption (Definition 2.3), with the distributed key generation handled by the Schnorr knowledge-of-exponent assumption, itself implied by the discrete logarithm assumption in the algebraic group model. Neither layer is invoked directly here: the group challenge is uniform by

the random-oracle modeling of H_2, and pseudorandomness of the final output $\beta = H_3(\gamma)$ follows from the output-privacy analysis below. Theorems 5.1, 5.2 and 5.3 require neither the one-more discrete logarithm assumption nor the algebraic group model. The aggregate proof enters our reductions only through its Fiat-Shamir soundness and never through witness extraction, which is what keeps all three reductions straight-line.

Output-privacy, i.e., the guarantee that an on-chain observer cannot recover the unblinded output $\gamma = h^s$ from the published pair $(\hat{h}, \hat{\gamma}) = (h^\kappa, h^{\kappa s})$, reduces to the **Computational Diffie-Hellman (CDH)** assumption (Definition 2.2). Intuitively, recovering h^s from h^κ and $h^{\kappa s}$ without knowing the user's secret blinding scalar κ is precisely the CDH problem in $\mathbb{G}$.

Unforgeability, i.e., the guarantee that an adversary who observes k valid blinded output pairs cannot produce a $(k+1)$-st valid pair for a fresh input, requires a stronger interactive assumption. If the adversary is purely external and observes only the smart contract state, this reduces to the **Gap One-More CDH (Gap-OM-CDH)** assumption (Definition 2.4), since the clan submits only the final aggregated blinded output $\hat{\gamma} = \hat{h}^s$ on-chain and the threshold structure remains hidden. However, if the adversary additionally corrupts up to $f \leq t-1$ clan members and observes their partial blinded outputs $\hat{h}^{s_i}$, the reduction requires the stronger **Gap Threshold One-More CDH (Gap-T-OM-CDH)** assumption (Definition 2.5): one in which the adversary additionally obtains evaluations of the underlying Shamir polynomial at f points and must still be unable to reconstruct a new full evaluation $\hat{h}^s$ beyond those it has legitimately obtained. Both assumptions are stated in fixed-key, chosen-target form, which is what the protocol exercises: the secret s of IcyVeil is fixed across all evaluations, and the only computational help an adversary obtains is exponentiation by that single s. Correspondingly, the decision oracle DDH_g is restricted to that fixed key, answering only whether $b = a^s$, which is the minimal decision capability our analysis requires. The formal treatment of both properties occupies the remainder of this section.

5.5.1 The Output-Privacy Game

We formalize output-privacy as a real-or-random game $\mathrm{Priv}^b_{\mathcal{A}}$, given in Algorithm 8, in which the adversary sees the entire public execution and may corrupt clan members.

Algorithm 8 The output-privacy game $\mathrm{Priv}^{b}_{\mathcal{A}}$

Input A security parameter 1^{λ} and a bit b

Output The guess b' of $\mathcal{A}$

1: $(\mathcal{C}, \{s_i\}_{i\in\mathcal{C}}, \alpha^{\star}) \leftarrow \mathcal{A}(1^{\lambda})$
2: $\big(\{(s_i, Y_i)\}_{i\in[n]}, Y\big) \leftarrow \mathrm{DistKeyGen}(1^{\lambda}, t, n)$ subject to $\{s_i\}_{i\in\mathcal{C}}$
3: $\mathrm{st} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{eval}}}\big(Y, \{Y_i\}_{i\in[n]}, \{s_i\}_{i\in\mathcal{C}}\big)$
4: $\kappa \xleftarrow{\$} \mathbb{Z}_q^{*}$; $\quad h^{\star} \leftarrow H_1(\alpha^{\star})$; $\quad \hat{h}^{\star} \leftarrow (h^{\star})^{\kappa}$
5: $T^{\star} \leftarrow \Big(h^{\star}, \hat{h}^{\star}, \hat{\pi}^{\star}, \{(D_i, E_i, \bar{D}_i, \bar{E}_i, \hat{\gamma}_i^{\star}, \hat{\pi}_i^{\star}, \hat{z}_i^{\star})\}_{i\in S}, \Pi^{\star}\Big)$
6: $\gamma^{\star} \leftarrow (h^{\star})^{s}$; $\quad \beta_0^{\star} \leftarrow H_3(\gamma^{\star})$; $\quad \beta_1^{\star} \xleftarrow{\$} \{0,1\}^{\Lambda}$
7: $b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{eval}}}(\mathrm{st}, T^{\star}, \beta_b^{\star})$
8: Return b'

The evaluation oracle $\mathcal{O}_{\mathrm{eval}}(\alpha)$, defined for any $\alpha \neq \alpha^{\star}$, has the challenger play the honest user with a fresh blind, run one session, and return the full session transcript; on request it also opens the session by returning $(\gamma, \beta, \pi_{\mathrm{eq}})$.

The corruption set and its share values are chosen by $\mathcal{A}$ in line 1, and the sharing in line 2 is sampled subject to them, as in Definition 2.5. The challenge input $\alpha^{\star}$ is committed in the same line, before any oracle access. Corrupt members of S follow the protocol within the challenge session. Neither κ nor π_{eq} of that session is revealed, so line 5 is the entire public view. Requests in which $\mathcal{A}$ supplies its own blinded base are the subject of the unforgeability games below, not of this one. The advantage is defined as

$$\mathsf{Adv}^{\text{out-priv}}_{\mathbb{G}}(\mathcal{A}) = \big|\Pr[\mathrm{Priv}^{0}_{\mathcal{A}} = 1] - \Pr[\mathrm{Priv}^{1}_{\mathcal{A}} = 1]\big| .$$

Theorem 5.1 (Output-privacy)**.** *In the random oracle model, for any PPT adversary $\mathcal{A}$ playing the committed-challenge game of Algorithm 8, corrupting up to $t-1$ clan members and never learning κ or π_{eq}, there is a CDH adversary $\mathcal{B}$ with*

$$\mathsf{Adv}^{\text{out-priv}}_{\mathbb{G}}(\mathcal{A}) \le q_{H_3} \cdot \mathsf{Adv}^{\mathrm{cdh}}_{\mathbb{G}}(\mathcal{B}) + \frac{c\, Q_{\mathrm{tot}}^{2}}{q},$$

where q_{H_3} counts the H_3 queries of $\mathcal{A}$, Q_{tot} its total oracle queries, and c is a small constant. Given the decision oracle DDH_g of Definition 2.4, the factor q_{H_3} improves to 1.

Because the transcript of game $\mathbf{G_1}$ in Section 5.5.3 is a function of the public keys and of the fresh scalar r, the corrupt shares $\{s_i\}_{i\in\mathcal{C}}$ never enter it. Output-privacy therefore holds against a coalition of up to $t-1$ clan members, not merely against an external on-chain observer, at no additional assumption cost.

5.5.2 One-More Unforgeability

We formalize unforgeability as two games, $\mathrm{Unf}^{\mathrm{ext}}_{\mathcal{A}}$ and $\mathrm{Unf}^{\mathrm{clan}}_{\mathcal{A}}$, given in Algorithm 9. They share all machinery except the corruption step and the budget.

Algorithm 9 The one-more unforgeability games $\mathrm{Unf}^{\mathrm{ext}}_{\mathcal{A}}$ and $\mathrm{Unf}^{\mathrm{clan}}_{\mathcal{A}}$

Input A security parameter 1^λ

Output 1 if $\mathcal{A}$ wins, and 0 otherwise

1: $(\mathcal{C}, \{s_i\}_{i\in\mathcal{C}}) \leftarrow \mathcal{A}(1^\lambda)$, in the clan game only, with $|\mathcal{C}| = f \le t-1$
2: $\big(\{(s_i, Y_i)\}_{i\in[n]}, Y\big) \leftarrow \mathrm{DistKeyGen}(1^\lambda, t, n)$ subject to $\{s_i\}_{i\in\mathcal{C}}$
3: $E \leftarrow \emptyset$; $\quad m \leftarrow 0$; $\quad \vec{q} \leftarrow (0, \ldots, 0)$
4: $\Big\{(\alpha_j, \beta_j, \gamma_j, \pi_{\mathrm{eq},j}, \hat{h}_j, \hat{\gamma}_j, \Pi_j)\Big\}_{j\in[\ell]} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{req}}, \mathcal{O}_{\mathrm{part}}}\big(Y, \{Y_i\}_{i\in[n]}, \{s_i\}_{i\in\mathcal{C}}\big)$
5: Return 1 if and only if every tuple passes Verify, the indices $d(\alpha_j)$ are pairwise distinct and lie outside E, and $\ell > B$

The request oracle $\mathcal{O}_{\mathrm{req}}(\alpha, \hat{h}, \hat{\pi})$ aborts unless $\mathrm{VerifyExp}_{H_2}(h, \hat{h}; \hat{\pi}) = 1$ for $h = H_1(\alpha)$; otherwise the clan evaluates, the platform publishes $(\hat{\gamma}, \Pi)$, and the game sets $E \leftarrow E \cup \{d(\alpha)\}$ and $m \leftarrow m + 1$. The partial oracle $\mathcal{O}_{\mathrm{part}}(i, \hat{h})$, available in the clan game only, has honest node i return its partial $\hat{\gamma}_i$ with the per-node proof and sets $q_i \leftarrow q_i + 1$. The budget is $B = m$ in $\mathrm{Unf}^{\mathrm{ext}}$ and $B = C_{t-f}(\vec{q})$ of Definition 2.5 in $\mathrm{Unf}^{\mathrm{clan}}$.

In $\mathrm{Unf}^{\mathrm{ext}}$ the clan is honest, $\mathcal{C} = \emptyset$, and $\mathcal{A}$ observes only the public smart contract state, so $\mathcal{O}_{\mathrm{part}}$ is unavailable. In $\mathrm{Unf}^{\mathrm{clan}}$, $\mathcal{A}$ observes every honest round message, including each honest partial $\hat{\gamma}_i$. Since each request carries a unique nonce inside its context string, H_1 assigns every request a distinct index $d(\alpha)$, so E records exactly the inputs on which $\mathcal{A}$ completed a full blinded evaluation, and a tuple is fresh if and only if its index lies outside E. The advantages $\mathsf{Adv}^{\text{unf-ext}}_{\mathbb{G}}(\mathcal{A})$ and $\mathsf{Adv}^{\text{unf-clan}}_{\mathbb{G}}(\mathcal{A})$ are the respective probabilities that the game returns 1.

Theorem 5.2 (External one-more unforgeability)**.** *Assume the blinding NIZK $\hat{\pi}$ is sound, and that the DLEQ and aggregate pre-verification proofs are sound, which is the statistical soundness of Fiat-Shamir Chaum-Pedersen proofs in the random oracle model, from which uniqueness of the DVRF output also follows. If Gap OM-CDH (Definition 2.4) is hard in $\mathbb{G}$, then for any PPT adversary $\mathcal{A}$ in $\mathrm{Unf}^{\mathrm{ext}}$ there is a reduction $\mathcal{B}$ with*

$$\mathsf{Adv}^{\text{unf-ext}}_{\mathbb{G}}(\mathcal{A}) \le \mathsf{Adv}^{\text{gap-om-cdh}}_{\mathbb{G}}(\mathcal{B}) + \frac{c\, Q_{\mathrm{tot}}^2}{q},$$

with $N = q_{H_1}$ and $Q = m$. The reduction is straight-line: it never rewinds $\mathcal{A}$, and it uses the blinding NIZK only through soundness, not through extractability.

In the fixed-quorum model, the combiner commits the responder set S to every participant before any partial evaluation is computed, exactly as in step 1 of $\mathrm{PartialEval}$.

Theorem 5.3 (Clan one-more unforgeability, fixed quorum)**.** *Under the assumptions of Theorem 5.2, in the fixed-quorum model, if Gap Threshold OM-CDH (Definition 2.5) is hard in $\mathbb{G}$, then for any PPT adversary $\mathcal{A}$ in $\mathrm{Unf}^{\mathrm{clan}}$ corrupting $f \le t-1$ members there is a reduction $\mathcal{B}$ with*

$$\mathsf{Adv}^{\text{unf-clan}}_{\mathbb{G}}(\mathcal{A}) \le \mathsf{Adv}^{\text{gap-t-om-cdh}}_{\mathbb{G}}(\mathcal{B}) + \frac{c\,(n-f)\, Q_{\mathrm{tot}}^2}{q},$$

with $N = q_{H_1}$ and Q the realized budget $C_{t-f}(\vec{q})$. The reduction is again straight-line.

Necessity of the gap versions. Definitions 2.4 and 2.5 both hand the adversary the decision oracle DDH_g for free. This is not a convenience but the price of working without a pairing. In a pairing-based output-private DVRF such as FlexiRand [40], the pairing is the query $\mathsf{DDH}_g(\hat{h}, \hat{\gamma})$: the public check $e(\hat{\gamma}, g) = e(\hat{h}, Y)$ accepts exactly when $\hat{\gamma} = \hat{h}^{\,s}$, and anyone can compute it from the public key alone. A pairing-free group offers no such test, so the capability must be postulated rather than inherited. The postulate is standard: under the parameter map $t_{\mathrm{JKKX}} = t - 1$ and $t'_{\mathrm{JKKX}} = f$, Definition 2.5 is the Gap Threshold One-More Diffie-Hellman assumption of Jarecki et al. [38], with Definition 2.4 the case $f = 0$, $t = 1$, and an adversary restricted to r generic group operations has advantage $O(Q\, r^2/q)$ for $r \geq Q \geq N$ [39]. The import is conservative: their gap oracle decides Diffie-Hellman tuples in general, whereas DDH_g decides only membership relative to the fixed key s, and within Theorems 5.2 and 5.3 it serves only as a certification filter.

Modeling and tightness. Fixed-quorum is the truthful description of IcyVeil rather than a strengthening: the quorum, and hence the Lagrange coefficients, are committed before partial evaluations exist, so the reduction has no responder set to guess. Both reductions are tight, that is, linear in the advantage of the adversary, because DDH_g recognizes one-more solutions directly and so replaces the rewinding-based extraction that would otherwise cost a square-root factor. A second convention concerns commitment timing: the consumed pair (D_i, E_i) is first revealed within the evaluation itself, rather than published at preprocessing as in Chapter 4. Straight-line reductions from Definitions 2.4 and 2.5, whose oracles return group elements, cannot produce response shares $\hat{z}_i$ against commitments fixed in advance, whereas the discrete-logarithm-based proofs of FROST [42] and of Icy-DVRF can, because their helper oracle returns scalars, which is why those protocols may publish early. Since the blinded halves $\bar{D}_i$ and $\bar{E}_i$ force a per-request first round in any case, the change costs no communication round, leaves every on-chain algorithm and hence all gas figures of Section 5.6 untouched, and matches the normative two-round flow of the FROST specification [19].

5.5.3 Proof Sketch of Output-Privacy

The argument proceeds by three game hops. $\mathbf{G_0}$ is Priv^0. In $\mathbf{G_1}$ the challenger renders the entire view of $\mathcal{A}$ without knowing s or any honest share: the honest Y_j are fixed by Lagrange interpolation in the exponent through $0 \mapsto Y$ and the corrupt shares, and the blinded base is re-anchored as $\hat{h}^\star := g^r$ for a fresh known r, so that the blind $\kappa = r/\log_g h^\star$ is implicit, uniform, and never needed. Every blinded value is then computable from the public keys alone, namely $\hat{\gamma}_i^\star = Y_i{}^r$ and $\hat{\gamma}^\star = Y^r$, and each barred commitment is the r-th power of its unbarred counterpart. All proofs, that is $\hat{\pi}^\star$, the per-node transcripts and $\Pi^\star$, are simulated by the standard honest-verifier zero-knowledge simulators with H_2 programmed at the read point, and evaluations on $\alpha \neq \alpha^\star$ are answered by setting $H_1(\alpha) := g^{r_\alpha}$ with known r_α, an opened

output being computable as $\gamma = Y^{r_\alpha}$. The view is identically distributed up to random-oracle programming collisions, which are absorbed into $c\,Q_{\mathrm{tot}}^2/q$, and $\gamma^\star$ is never computed. In $\mathbf{G_2}$ the value $\beta^\star$ is replaced by a uniform string. Games $\mathbf{G_1}$ and $\mathbf{G_2}$ are identical until $\mathcal{A}$ queries H_3 at $\gamma^\star$, and $\mathbf{G_2}$ is exactly Priv^1 rendered through the same simulator.

To bound that bad event, $\mathcal{B}$ embeds a CDH instance (g, g^a, g^b) by programming $H_1(\alpha^\star) := g^a$ and publishing $Y := g^b$, a placement made free by the advance commitment to $\alpha^\star$. It then runs $\mathbf{G_2}$ verbatim and outputs one of the q_{H_3} queried points, which equals $\gamma^\star = g^{ab}$ with probability $1/q_{H_3}$ conditioned on the bad event. Given DDH_g, the reduction instead recognizes the correct point directly, which removes the factor q_{H_3}.

5.5.4 Proof Sketch of One-More Unforgeability

External adversary. For Theorem 5.2, the reduction $\mathcal{B}$ publishes the Gap OM-CDH key Y as the group public key and answers the d-th fresh H_1 query with the d-th challenge element g_d. Each accepted evaluation is served by one helper query $\hat{\gamma} \leftarrow \mathcal{O}_s(\hat{h})$, with the aggregate proof simulated by honest-verifier zero-knowledge and H_2-programming. Soundness of the pre-verification and DLEQ proofs forces every verifying tuple to satisfy $\gamma_j = g_{d_j}^s$, which $\mathcal{B}$ recognizes through DDH_g, and soundness of $\hat{\pi}$ guarantees that each accepted $\hat{h}$ is a power of its own base, so each of the $Q = m$ helper queries touches only one non-fresh target. A winning $\mathcal{A}$ therefore hands $\mathcal{B}$ solutions to $Q + 1$ distinct challenge elements using Q helper calls, and the last term of the bound absorbs random-oracle programming collisions. No separate hash-guessing term arises, since every verifying tuple exhibits γ_j and Verify itself checks $\beta_j = H_3(\gamma_j)$.

Clan adversary. For Theorem 5.3, only two elements change. First the key: $\mathcal{B}$ forwards the corrupt set and chosen shares of $\mathcal{A}$ to its challenger and publishes Y together with all Y_i, so that honest shares are never known. Second the helper: each honest partial evaluation is served by one per-slot query $\hat{\gamma}_i \leftarrow \mathcal{O}_F(i, \hat{h})$ with a simulated per-node DLEQ proof. The count is won by threshold alignment. Soundness of $\hat{\pi}$ forces each accepted $\hat{h}$ to be a power of a single base, so a slot-i query advances the reconstruction of exactly one target; a malformed $\hat{h}$ spanning several bases would otherwise let one query contribute across targets and break the budget. Completing any target requires $t - f$ honest partial evaluations, so $\mathcal{A}$ completes at most $C_{t-f}(\vec{q})$ targets and a win hands $\mathcal{B}$ one more. The factor $n - f$ reflects the simulation of up to $n - f$ per-node proofs per request. Setting $f = 0$ and $t = 1$ collapses the per-slot oracle, the budget and Definition 2.5 to their counterparts in Theorem 5.2.

5.6 Comparison

IcyVeil provides an output-privacy layer for the pairing-free Icy-DVRF, exactly as FlexiRand provides one for the pairing-based GLOW-DVRF. Therefore, the comparison turns on the cost

of the privacy layer relative to the underlying DVRF. Once again, we assume our protocols are instantiated on the BLS12-381 curve, a Type-3 pairing-friendly curve standardized in EIP-2537 [53]. Under the Symmetric External Diffie-Hellman (SXDH) assumption, the DDH problem is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$. We further assume the hardness of the CDH and OM-CDH problems in $\mathbb{G}_1$ to guarantee protocol integrity. To optimize for gas efficiency, we map the underlying group $\mathbb{G}$ to the subgroup $\mathbb{G}_1$, since element representations are smaller and group operations are significantly cheaper than in $\mathbb{G}_2$.

As shown in Table 5.1, based on the findings of Chapter 4, while GLOW-DVRF achieves constant-size proofs, its reliance on computationally expensive bilinear pairing operations drives its theoretical verification cost to roughly $201{,}280$ gas (Table 6.1). By relying entirely on more efficient multiscalar multiplications and hashes, Icy-DVRF reduces the estimated baseline verification cost to approximately $51{,}792$ gas, representing nearly a fourfold theoretical efficiency improvement.

Table 5.1: Theoretical Gas Cost Estimations of Verification Costs on BLS12-381.

Gas cost	**Verification cost**	**Calldata cost**	**Total cost**
GLOW-DVRF	$\approx 200{,}000$	$1{,}280$	$\approx \mathbf{201{,}280}$
Icy-DVRF	$\approx 50{,}000$	$1{,}792$	$\approx \mathbf{51{,}792}$

The core divergence between Icy-DVRF and IcyVeil lies in how users interact with the VRF service smart contract. While Icy-DVRF evaluates a public input directly, IcyVeil requires the user to mask the formatted input h with a secret random scalar κ to produce a blinded value $\hat{h}$. To prevent malicious inputs, the user must also compute a Schnorr-based NIZK proof ($\hat{\pi}$) demonstrating knowledge of κ. The VRF Clan subsequently evaluates this blinded input and returns a blinded proof that only the user can locally unveil.

Since IcyVeil delegates the unblinding phase to the user, the primary additional burden on the smart contract, compared to standard Icy-DVRF, is verifying the user's initial NIZK blinding proof ($\hat{\pi}$). Verifying this proof via $\mathrm{VerifyExp}$ requires the smart contract to compute $\overline{u} \leftarrow g^{\hat{z}}\hat{h}^{c}$, which amounts to a multiscalar multiplication in the $\mathbb{G}$ group alongside a hash evaluation. Assuming the use of EIP-2537 precompiles on the BLS12-381 curve [53], where a $\mathbb{G}_1$ exponentiation costs approximately $12{,}000$ gas, this extra blinding proof verification accounts for roughly $12{,}000$ to $15{,}000$ additional gas per request.

The empirical benchmarks underpinning these figures, namely Icy-DVRF at $88{,}803$ gas against the $155{,}838$ gas of GLOW-DVRF, a 43.02% reduction, are presented in full in Chapter 6. Even when the blinding-proof overhead is included, the total on-chain execution remains well below the pairing-based cost of FlexiRand, so IcyVeil provides output privacy at nearly the cost of the underlying pairing-free DVRF.

With IcyVeil, we complete the progression of this thesis: it adds output privacy to the compact, pairing-free, low-interaction Icy-DVRF of Chapter 4 without reintroducing pairings, and is, to the best of our knowledge, the first pairing-free output-private DVRF. It does, however, inherit

one limitation from Icy-DVRF, namely the abort-and-identify robustness model. The tension between this efficiency-driven design choice and full robustness, is the problem examined in Chapter 7.

CHAPTER 6

EVALUATION AND COMPARISON

We concluded each of the preceding construction chapters by mentioning the results in this one. We now present the evaluation in two parts. In Section 6.1, we consolidate the theoretical operation-count and gas estimates developed across the construction chapters, so that all four schemes, namely DDH-DVRF, GLOW-DVRF, DVRFwCP, and Icy-DVRF, can be compared on a single basis. Then, in Section 6.2, we report the measurements from a Solidity deployment on the Ethereum Sepolia testnet. As the measurements show, the theoretical efficiency claims hold in a real execution environment, where precompile pricing, calldata encoding, and implementation overheads all are accounted for. In the case of the pairing-based baseline, the live environment is in fact more favorable to Icy-DVRF, relative to GLOW-DVRF, than the theoretical model predicts.

6.1 Theoretical Analysis

As before, we instantiate all protocols on the BLS12-381 curve and assume the EIP-2537 [53] precompile prices for group operations, with calldata charged at the EIP-2028 [5] rate of 16 gas per non-zero byte. Under these conventions, an exponentiation in $\mathbb{G}_1$ costs approximately $12{,}000$ gas and a multiplication approximately 500 gas, while a bilinear pairing check is far more expensive. Table 6.1 collects the resulting per-verification estimates established in Chapters 3 and 4.

6.1.1 DVRFwCP and Icy-DVRF Verification Cost

As established in Section 3.6, the Verify algorithm of DVRFwCP performs four exponentiations and two multiplications in $\mathbb{G}$, for a verification cost of approximately $4 \times 12{,}000 + 2 \times 500 = 49{,}000$ gas, to which the calldata for a proof of one group element and two scalars adds approximately $1{,}792$ gas, giving a total of about $51{,}792$ gas. The key observation is that Icy-DVRF inherits this figure exactly. Its Verify algorithm, presented in Section 4.4, is structurally identical to that of DVRFwCP. Therefore, the two schemes share an identical theoretical verification cost, and differ only off-chain, in how the shared challenge

Table 6.1: Consolidated Theoretical Gas Estimates for On-Chain Verify on BLS12-381.

Scheme	Verification cost	Calldata cost	Total cost
DDH-DVRF	$\approx 70{,}000 \cdot t$	$768 \cdot t$	$\approx \mathbf{70{,}768 \cdot t}$
GLOW-DVRF	$\approx 200{,}000$	$1{,}280$	$\approx \mathbf{201{,}280}$
DVRFwCP	$\approx 50{,}000$	$1{,}792$	$\approx \mathbf{51{,}792}$
Icy-DVRF	$\approx 50{,}000$	$1{,}792$	$\approx \mathbf{51{,}792}$

is produced: interactively for DVRFwCP, and via preprocessing for Icy-DVRF. Since the empirical study of the next section (Section 6.2) measures precisely this common Verify logic, its results apply equally to both schemes.

6.1.2 Icy-DVRF Communication Improvement over DVRFwCP

The communication cost of a DVRF protocol is determined by what each participant must broadcast per evaluation request. As established in Section 4.6.2, the two schemes share the same proof size on output, one group element and two scalars, but differ fundamentally in what they must transmit during proof generation. The difference traces directly to how the shared challenge c is produced. In DVRFwCP, producing c requires running the Augmented Secure-DKG, which forces each participant to broadcast polynomial commitments proportional to the threshold: the per-participant total comes to $(2t+1)\mathbb{G}+4\mathbb{Z}_q$, growing linearly in t. In Icy-DVRF, the FROST-style preprocessing replaces this interactive round entirely. The shared commitment R is assembled locally from commitments published in advance, so at proof-generation time each participant broadcasts only its partial response and the challenge hash, for a committee-independent total of $3\mathbb{G} + 4\mathbb{Z}_q$.

The net effect is a shift from linear to constant per-participant communication: at $(n, t) = (50, 34)$, DVRFwCP requires $3{,}440$ bytes per node while Icy-DVRF requires 272 bytes, a $12.65\times$ reduction. The trade-off is a slightly higher payload at $\mathrm{PartialEval}$ ($3\mathbb{G}$ versus $1\mathbb{G}$), since Icy-DVRF must also transmit the auxiliary commitments $(\bar{D}_i, \bar{E}_i)$. This constant overhead is recovered within a single PartialProofGen phase for any $t \geq 2$, and becomes negligible at the committee sizes relevant to practical deployments. The concrete byte counts across five committee sizes are reported in Table 6.6.

6.1.3 IcyVeil Overhead Extrapolation

As presented in Chapter 5, IcyVeil does not alter the VRF clan-side Verify logic: the blinded proof is verified by $\mathrm{PreVerify}$ is the same computation that underlies Icy-DVRF's Verify. The only additional on-chain burden is the smart contract's verification of the user's Schnorr blinding proof $\hat{\pi}$. Verifying this proof via $\mathrm{VerifyExp}$ requires the smart contract to compute $\overline{u} \leftarrow g^{\hat{z}}\hat{h}^{c}$, translating to a multiscalar multiplication in $\mathbb{G}$ alongside a hash evaluation.

Assuming the EIP-2537 precompiles, where a $\mathbb{G}_1$ exponentiation costs approximately $12{,}000$ gas, this extra blinding-proof verification imposes a minimal theoretical overhead of roughly $12{,}000$ to $15{,}000$ gas per request.

This range is an extrapolation rather than a direct measurement: it is obtained by pricing the single additional multiscalar multiplication at the EIP-2537 rate and allowing a margin for the surrounding Solidity logic and hashing. The margin is calibrated against the difference, documented in Section 6.2, between the precompile-only cost of Icy-DVRF's Verify and its full measured cost on Sepolia. The same difference quantifies the per-operation framing overhead a real contract incurs on top of raw precompile prices, and the same overhead is expected to apply to the blinding-proof check. IcyVeil therefore inherits Icy-DVRF's measured verification cost plus this bounded, pairing-free increment.

6.1.4 Scalability Under Committee Growth

The principal theoretical distinction amongst these verification procedures lies not in the constant terms but in how the cost scales with the committee. As Table 6.1 records, DDH-DVRF incurs a verification cost of approximately $70{,}768 \cdot t$ gas, growing linearly in the threshold t because its proof consists of t group elements, each of which must be verified separately on-chain. For a committee of even modest size this becomes prohibitively expensive. GLOW-DVRF removes the linear dependence by compressing the proof to a constant size, but pays a large fixed cost of approximately $201{,}280$ gas for the bilinear pairings its verification requires. DVRFwCP and Icy-DVRF achieve the best of both: a constant proof size *and* pairing-free verification, for a committee-independent total of approximately $51{,}792$ gas.

This is exactly the "trilemma" of Chapter 1 viewed in terms of on-chain cost. Linear proofs (DDH-DVRF) and pairing cost (GLOW-DVRF) are precisely the two facets that the constructions of this thesis avoid, and the gas figures make this quantitative: the verification cost is constant in the committee size and free of the pairing construction. The third facet, interaction overhead, is off-chain and therefore is not counted towards the on-chain cost, which is why the elimination of this overhead in Icy-DVRF (Chapter 4) carries no on-chain penalty.

6.2 Empirical Results

6.2.1 On-Chain Evaluation and Benchmarks

To validate the theoretical efficiency of our proposal, we implemented the verification logic for Icy-DVRF and GLOW-DVRF in a single Solidity smart contract on the Sepolia testnet, using the EIP-2537 precompiles [53] for BLS12-381 curve operations. Our empirical study

specifically focused on the on-chain verification phase for the following reasons. First, there is a *cost asymmetry*: while the preprocessing and proof generation phases occur entirely off-chain, incurring zero gas costs for the decentralized network, the verification phase is a mandatory on-chain operation that consumes restricted block space. Second, *bottleneck identification*: in production environments, the primary scalability bottleneck is the high cost of on-chain execution and data availability, whereas the off-chain computation is measured in mere milliseconds, even when the number of nodes scales up to hundreds.

It should be noted that our Solidity contract code and the benchmarks are not on par with production code in terms of optimization and security, but are written merely for on-chain benchmark and demonstration purposes. The implementation of the smart contract and benchmark functions, along with the Sepolia testnet address of the deployed contract, can be found at https://github.com/abozer/DVRF_Verify.

Since Icy-DVRF and DVRFwCP share an identical Verify algorithm (Section 6.1.1), and IcyVeil adds only the bounded blinding-proof increment of Section 6.1.3, the single benchmarked Icy-DVRF verifier represents all three constructions of this thesis. The comparison that matters is therefore against the pairing-based baseline, GLOW-DVRF.

Verification gas costs. Experimental results for the Verify function on the Sepolia testnet demonstrate superior performance for Icy-DVRF, which maintained a constant execution cost of $88{,}803$ gas. In contrast, GLOW-DVRF exhibited a significantly higher execution cost of $155{,}838$ gas. Icy-DVRF thus achieves a reduction of $67{,}035$ gas, a 43.02% improvement over its pairing-based counterpart, as summarized in Table 6.2.

Table 6.2: Measured Full Verification Gas on the Sepolia Testnet.

Scheme	**Verify gas (Sepolia)**	**Reduction vs. GLOW**	**%**
GLOW-DVRF	$155{,}838$	–	–
Icy-DVRF	$88{,}803$	$67{,}035$	43.02%

We note that the measurement figures do not match the theoretical estimates of Table 6.1, and examining where the mismatch comes from is informative. The BLS12-381 precompile handles pairing operations more cheaply than the theoretical worst-case price presumes, bringing the measured cost of GLOW-DVRF below the $201{,}280$ gas estimate. This is due to implementation-level optimizations within the precompile that reduce the effective computational overhead of pairing-based comparisons relative to standard pricing models. Even with these optimizations, Icy-DVRF retains a 43% advantage over this reduced baseline. We do not include a direct benchmark against DDH-DVRF, whose verification costs scale linearly with the threshold number t of provers, making it increasingly expensive as the committee grows.

Calldata and precompile analysis. To pinpoint the source of the efficiency gain, we decompose the verification cost into its two principal components: the calldata required to transmit the proof, and the precompile operations required to verify it. The calldata gas consumption was evaluated in accordance with the gas pricing specifications defined in EIP-2028. As detailed in Table 6.3, the Icy-DVRF Verify function requires $6{,}784$ gas for calldata, whereas the GLOW-DVRF counterpart consumes $7{,}224$ gas, a reduction of 440 gas in favor of Icy-DVRF. Icy-DVRF achieves this despite transmitting slightly more total bytes (772 versus 708); the saving emerges from the zero/non-zero byte composition of the encoded arguments, which EIP-2028 prices at 4 and 16 gas respectively.

Table 6.3: Calldata Comparison for Icy-DVRF and GLOW-DVRF Verify.

Metric	**Icy-DVRF**	**GLOW-DVRF**
Selector	4B	4B
Head	192B	128B
Tail Arguments	Y: 160B	pk: 288B
	γ: 160B	x: 64B
	c: 64B	v: 64B
	z: 64B	π: 160B
	α: 64B	
	β: 64B	
Tail (Total)	576B	576B
Total Bytes	**772B**	**708B**
Calldata Gas (EIP-2028)	**6,784**	**7,224**

(a) The names of the arguments are from each respective algorithm.

(b) In EIP-2028, zero-bytes and non-zero-bytes cost 4 and 16 gas, respectively.

By isolating precompile costs, we can evaluate the intrinsic cryptographic complexity of each protocol independent of implementation-specific overhead or Solidity logic. This comparison, detailed in Table 6.4, highlights the impact of primitive operations: the Verify function of Icy-DVRF achieves a significant efficiency advantage at $51{,}984$ gas by avoiding the expensive elliptic curve pairings that drive GLOW-DVRF to $120{,}908$ gas, a reduction of $68{,}924$ gas in core cryptographic operations. The single entry, the $103{,}000$ gas that comes up from the pairing that Icy-DVRF never performs, accounts for the entire advantage. The multiscalar multiplication that Icy-DVRF uses in its place is more expensive ($45{,}752$ versus $12{,}100$ gas), but not nearly enough to offset the pairing it eliminates.

Storage and state management. Beyond per-verification gas, a deployment must be able to sustain the off-chain preprocessing state. Storing preprocessed commitment pairs $(D_{i,j}, E_{i,j})$ does not impose a significant burden on provers: on the BLS12-381 curve [28], a compressed $\mathbb{G}_1$ element occupies 48 bytes, so each commitment pair requires 96 bytes. Even for a substantial batch size of $\Omega = 10{,}000$, a participant maintains only about 0.96 MB of local storage, confirming that the preprocessing model is sustainable even for nodes with limited

Table 6.4: Comparison of Precompile-Only Costs Excluding Verify Framing Overhead.

Operation	**Icy-DVRF**	**GLOW-DVRF**	**Difference**
Hash-to-$\mathbb{G}_1$[(a)]	5,600	5,600	0
G1MSM[(b)]	45,752	12,100	−33,652
SHA-256	632	208	−424
Pairing in BLS12-381	0	103,000	+103,000
Total (precompiles)	**51,984**	**120,908**	**+68,924**

(a) In Icy-DVRF, $\mathbb{G} = \mathbb{G}_1$. (b) Multiscalar multiplication in $\mathbb{G}_1$.
(c) Varied SHA-256 costs reflect the different number of inputs and total data size processed by each protocol.

capacity. To ensure that all participants use the same precomputed nonces for a given round, the combiner includes a single-use index j in its request, allowing each prover to retrieve the corresponding pair by direct lookup; this index-based synchronization adds negligible latency relative to the cryptographic operations. Finally, in large committees ($n > 100$), where bandwidth rather than computation tends to dominate, each participant broadcasts a fixed 272 bytes ($3\mathbb{G}_1 + 4\mathbb{Z}_q$) during the initial phases and only 64 bytes ($2\mathbb{Z}_q$) during final proof generation, independent of committee size.

6.2.2 Off-Chain Evaluation and Benchmarks

Having established the on-chain verification advantage of Icy-DVRF above, we now quantify its off-chain efficiency by benchmarking the full protocol across a range of committee sizes (n, t). All measurements were performed on a MacBook Air M4 (32 GB) using a single-process simulation with BLS12-381 $\mathbb{G}_1$ operations via `py_ecc` [30] and a network model with a 120 ms latency per interaction cycle. The reported figures are the means over 100 iterations for each benchmark. The implementation of the benchmark functions is available at https://github.com/abozer/DVRF_Comparison.

Runtime comparison. We benchmark the full per-request runtime of both protocols at five committee sizes; Icy-DVRF wins at every one (Table 6.5). Icy-DVRF is consistently faster, with the advantage growing superlinearly: the total speedup rises from $1.40\times$ at $(5, 3)$ to $7.14\times$ at $(50, 34)$, while the per-node speedup rises from $1.46\times$ to $4.43\times$ over the same range. The growth is explained by DVRFwCP's Augmented Secure-DKG round, whose cost scales with n and t: at $(50, 34)$, its computation time of 90,433 ms dwarfs Icy-DVRF's 12,471 ms, whereas the network latency difference (one fewer round at 120 ms) is negligible by comparison. The per-node column accounts for the fact that in a real deployment all n parties compute in parallel, so the realistic latency is Comp.$/n$ plus network rounds; even on this per-node basis the advantage persists and widens with committee size.

The reported figures are from a single-process simulation that runs all n parties sequentially on a single machine using one CPU core. The *Comp.* column is the cryptographic computation

Table 6.5: Off-Chain Per-Request Cost: DVRFwCP vs. Icy-DVRF.

	DVRFwCP			**Icy-DVRF**				
(n, t)	**Comp.** (ms)	**Total** (ms)	**Per-node** (ms)	**Comp.** (ms)	**Total** (ms)	**Per-node** (ms)	**Speedup**	**Per-node Speedup**
$(5, 3)$	420.2	780.2	444.0	315.5	555.5	303.1	$1.40\times$	$1.46\times$
$(7, 4)$	620.3	980.3	448.6	447.8	687.8	304.0	$1.43\times$	$1.48\times$
$(10, 7)$	1378.2	1738.2	497.8	936.0	1176.0	333.6	$1.48\times$	$1.49\times$
$(25, 17)$	9019.3	9379.3	720.8	3644.4	3884.4	385.8	$2.41\times$	$1.87\times$
$(50, 34)$	90433.5	90793.5	2168.7	12471.1	12711.1	489.4	$7.14\times$	$4.43\times$

(a) The one-time DistKeyGen and the offline Preprocess of Icy-DVRF are not included.
(b) Total Speedup is the ratio of the two Total columns.
(c) Per-node Speedup is the ratio of the two Per-node columns.

Table 6.6: Per-Node Post-Setup Broadcast Instantiating the Formulas of Table 4.2.

(n, t)	**DVRFwCP** (bytes)	**Icy-DVRF** (bytes)	**Improvement**
$(5, 3)$	464	272	$1.71\times$
$(7, 4)$	560	272	$2.06\times$
$(10, 7)$	848	272	$3.12\times$
$(25, 17)$	1,808	272	$6.65\times$
$(50, 34)$	3,440	272	$12.65\times$

(a) Using compressed BLS12-381: $1\mathbb{G}_1 = 48\,\text{B}$, $1\mathbb{Z}_q = 32\,\text{B}$.
(b) Per-node totals: DVRFwCP $(2t{+}1)\mathbb{G}_1 + 4\mathbb{Z}_q$, Icy-DVRF $3\mathbb{G}_1 + 4\mathbb{Z}_q$.

time per request, and the *Total* column adds 120 ms per interaction round (three rounds for DVRFwCP, two for Icy-DVRF). In a real-world deployment the per-party computation runs in parallel while the network rounds are incurred once, so we estimate the realistic per-node latency as Comp.$/n$ + (rounds $\times$ 120 ms), reported in the *Per-node* columns.

Communication cost at concrete committee sizes. We instantiate the broadcast formulas of Section 4.6.2 at five committee sizes to see what the asymptotics mean in bytes (Table 6.6). DVRFwCP's per-node broadcast grows as $(2t+1)\mathbb{G}_1 + 4\mathbb{Z}_q$, reaching 3,440 bytes at $(50, 34)$, while Icy-DVRF's is fixed at $3\mathbb{G}_1 + 4\mathbb{Z}_q = 272$ bytes regardless of committee size, a $12.65\times$ improvement at the largest configuration. The constant overhead at $\mathrm{PartialEval}$ ($3\mathbb{G}_1$ vs. $1\mathbb{G}_1$) is recovered within the first $\mathrm{PartialProofGen}$ phase for any $t \geq 2$, confirming that the trade-off is unambiguously in Icy-DVRF's favor at all practical committee sizes.

Setup cost and preprocess amortization. We now turn to the setup costs, where Icy-DVRF spends rather than saves: it has higher fixed costs, and pays once for a key and a reusable

Table 6.7: One-Time Setup Cost for DVRFwCP and Icy-DVRF.

	DistKeyGen		**Icy-DVRF Preprocess**		
(n, t)	**DVRFwCP** (ms)	**Icy-DVRF** (ms)	**Comp.** (ms)	**Storage** (KB)	**Amort.** (ms/req)
$(5, 3)$	180.3	243.2	4234.3	46.9	42.3
$(7, 4)$	335.8	439.0	5886.5	65.6	58.9
$(10, 7)$	780.4	984.8	8328.8	93.8	83.3
$(25, 17)$	8330.6	12712.1	20648.0	234.4	206.5
$(50, 34)$	104506.4	197054.0	42421.4	468.8	424.2

(a) DistKeyGen runtime is shown for both protocols. Preprocess is offline and unique to Icy-DVRF.
(b) $\Omega = 100$ pairs/party. (c) Amort. is Preprocess compute divided over a batch of $\Omega = 100$ requests.
(d) Storage $= n \times \Omega \times 96$ B (compressed BLS12-381).

batch of nonces, and we account for that here so the comparison stays honest (Table 6.7). The DistKeyGen cost is higher for Icy-DVRF at every committee size, reflecting the more complex FROST-based key generation relative to DVRFwCP's Secure-DKG; at $(50, 34)$ the gap reaches $197{,}054$ ms vs. $104{,}506$ ms. However, both are one-time costs that are fully amortized over the lifetime of the shared key. The Preprocess phase is also one-time in the sense that it produces a batch of Ω nonce pairs that serve Ω future requests; amortized over $\Omega = 100$ requests, the per-request cost ranges from 42.3 ms at $(5, 3)$ to 424.2 ms at $(50, 34)$. Although Preprocess is absent in DVRFwCP, the cost of it is well below the per-request savings shown in Table 6.5. Storage requirements remain modest: even at $(50, 34)$ with $\Omega = 100$, each participant stores only 468.8 KB of preprocessed material.

Per-phase breakdown. Breaking the runtime down by phase pinpoints exactly where Icy-DVRF earns its advantage. We measure each phase at $(n, t) = (10, 7)$, complementing the operation counts of Table 4.1 (Table 6.8). The dominant saving is concentrated in $\mathrm{PartialProofGen}$, which falls from 874.7 ms in DVRFwCP to 441.6 ms in Icy-DVRF, which is a reduction of factor 1.98 in the most expensive online phase, and the direct consequence of replacing the Augmented Secure-DKG with local nonce assembly. The Verify times are nearly identical (18.9 ms vs. 19.1 ms), confirming the theoretical claim that the two protocols share the same on-chain verification logic. The additional $\mathrm{Combine}$ cost in Icy-DVRF (312.9 ms vs. 150.9 ms) reflects the extra aggregation work over the preprocessed commitments, but this is more than offset by the savings in $\mathrm{PartialProofGen}$.

Measurement library. Tables 6.9 and 6.10 were measured with `py-arkworks-bls12381`, a native binding to the arkworks BLS12-381 implementation, while Tables 6.5–6.8 were measured with the pure-Python `py_ecc` [30]. We changed the library because throughput figures are only meaningful at deployment speeds, and `py_ecc`

Table 6.8: Measured Per-Phase Computation Time (ms) at $(n, t) = (10, 7)$.

Phase	**DVRFwCP**	**Icy-DVRF**
DistKeyGen	780.4	984.8
Preprocess[(a)]	–	8328.8
PartialEval	147.3	162.4
EvalCombine[(b)]	186.3	–
PartialProofGen	874.7	441.6
Combine[(c)]	150.9	312.9
Verify	18.9	19.1

(a) This algorithm is not present in DVRFwCP.
(b) This algorithm is not present in Icy-DVRF.
(c) The corresponding algorithm is named $\mathrm{ProofCombine}$ in DVRFwCP.

is too slow to give them. At the three committee sizes the two sets share, the compute time falls by factors of fifty to over three hundred, so absolute figures from one set cannot be compared with the other. The ratios within each set still hold, since every construction there was measured under the same conditions.

Throughput under slot budgets. We now place the off-chain figures against the interval a deployed beacon actually has to work with, and we widen the comparison to include GLOW-DVRF, which has no interactive round to remove and therefore does not appear in the preceding tables. We report throughput against two slot budgets. The first is the 12 s slot time that Ethereum uses at the time of writing, which is the interval within which a beacon must produce a fresh output if it is to serve one value per block. The second is 6 s, which anticipates EIP-7782 [1], a proposal to reduce the slot time from 12 s to 6 s in order to lower confirmation latency and shorten the epoch, and a candidate for the Glamsterdam upgrade. The halved budget is the more demanding test because off-chain latency here is dominated by network round trips rather than by computation: each additional round costs a fixed 120 ms that does not shrink when the slot does. Under the 6 s budget every construction still delivers more than one output per slot at all committee sizes measured, so the halving does not by itself exclude any of them. What it does is compress the margin, and the ordering of that margin is set by the number of rounds rather than by the cost of the cryptography. This is why GLOW-DVRF leads every column of Table 6.9: its single round costs 120 ms of network time against 240 ms for Icy-DVRF and 360 ms for DVRFwCP, and that fixed gap dominates the sub-millisecond differences in group arithmetic. The advantage is confined to the off-chain path, and it is bought with the pairing-based verification whose on-chain cost we measured in Section 6.2.1.

Table 6.9: Off-Chain Throughput Under Ethereum Slot Budgets.

(n, t)	**Scheme**	**Rounds**	**Compute** (ms)	**Latency** (ms)	**Runs** (per 12 s)	**Runs** (per 6 s)
$(5, 4)$	Icy-DVRF	2	7.20	247.2	48.5	24.3
	DVRFwCP	3	7.89	367.9	32.6	16.3
	GLOW-DVRF	1	4.10	124.1	96.7	48.3
$(7, 5)$	Icy-DVRF	2	8.53	248.5	48.3	24.1
	DVRFwCP	3	10.25	370.3	32.4	16.2
	GLOW-DVRF	1	4.64	124.6	96.3	48.1
$(10, 7)$	Icy-DVRF	2	11.49	251.5	47.7	23.9
	DVRFwCP	3	13.73	373.7	32.1	16.1
	GLOW-DVRF	1	5.87	125.9	95.3	47.7
$(16, 11)$	Icy-DVRF	2	17.51	257.5	46.6	23.3
	DVRFwCP	3	21.74	381.7	31.4	15.7
	GLOW-DVRF	1	8.18	128.2	93.6	46.8
$(25, 17)$	Icy-DVRF	2	26.46	266.5	45.0	22.5
	DVRFwCP	3	179.85	539.9	22.2	11.1
	GLOW-DVRF	1	54.86	174.9	68.6	34.3
$(50, 34)$	Icy-DVRF	2	258.19	498.2	24.1	12.0
	DVRFwCP	3	275.64	635.6	18.9	9.4
	GLOW-DVRF	1	103.13	223.1	53.8	26.9

(a) Measured with `py-arkworks-bls12381`, a native binding to the arkworks BLS12-381 implementation.
(b) Compute is the median over 50 iterations after 5 warmup runs, with the outermost 5% trimmed.
(c) Latency = compute + rounds $\times$ 120 ms, the assumed per-round network delay.
(d) Committee sizes follow a threshold fraction of 0.67, so the pairs differ from those of Table 6.5.

Batching. Because the network term is paid once per session rather than once per request, issuing a batch of B requests within a single session amortizes it. Table 6.10 sweeps B from 10 to 1000. Throughput rises steeply at first and then flattens as the amortized round-trip term falls below the per-request compute time, which is the point at which the protocol becomes compute-bound rather than latency-bound. At $(10, 7)$ Icy-DVRF moves from 28.5 to 88.3 requests per second, a factor of 3.1, and the corresponding figures for GLOW-DVRF are 56.6 and 171.3. Both gain by roughly the same factor because batching removes the same fixed cost from each, and GLOW-DVRF keeps the lead it holds in Table 6.9.

Table 6.10: Batched Throughput (requests/s) as a Function of Batch Size B.

(n, t)	**Scheme**	**Throughput (requests/s) at Batch Size B**				
		10	50	100	500	1000
$(5, 4)$	Icy-DVRF	32.5	86.3	108.9	138.1	142.4
	GLOW-DVRF	62.5	156.3	192.7	236.0	242.9
$(7, 5)$	Icy-DVRF	30.9	76.3	93.5	114.3	117.7
	GLOW-DVRF	60.6	144.4	175.2	211.9	218.2
$(10, 7)$	Icy-DVRF	28.5	63.0	74.1	86.4	88.3
	GLOW-DVRF	56.6	123.5	145.3	169.4	171.3
$(16, 11)$	Icy-DVRF	24.6	46.7	52.6	58.6	59.5
	GLOW-DVRF	50.2	96.9	109.4	122.6	124.7
$(25, 17)$	Icy-DVRF	20.4	33.5	36.4	39.2	39.6
	GLOW-DVRF	42.5	71.8	78.6	85.4	86.0
$(50, 34)$	Icy-DVRF	12.8	18.2	19.3	20.3	20.3
	GLOW-DVRF	29.8	41.6	43.8	45.8	46.1

(a) Throughput $= 1000$ / amortized per-request latency; batch latency $=$ batch compute $+$ one set of round trips.
(b) Rounds per session: Icy-DVRF two, GLOW-DVRF one; each amortized over the batch.
(c) DVRFwCP not measured in this sweep.
(d) Conditions as in Table 6.9.

6.3 Summary

When we combine the theoretical and empirical parts of the evaluation, we unveil a consistent picture, which we summarize in four findings.

On-chain economy: The measured verification cost of $88{,}803$ gas represents a 43.02% reduction over the pairing-based GLOW-DVRF baseline, and the precompile decomposition attributes this advantage to the elimination of the $103{,}000$ gas bilinear pairing. Thus, the theoretical promise of pairing-free verification is confirmed in a live environment.

Bandwidth efficiency. The constructions transmit a constant-size proof of one group element and two scalars, and the calldata analysis shows this translates into a genuine on-chain saving even against a pairing-based proof of comparable nominal size. Off-chain, in Icy-DVRF, the per-participant broadcast is fixed at 272 bytes regardless of committee size, and the preprocessing state remains under one megabyte for batches of ten thousand.

Optimal scaling. Verification cost is constant in the committee size t, in contrast to the $\sim 70{,}768 \cdot t$ linear growth of the pairing-free but linear-proof DDH-DVRF. This confirms the

central claim of Chapter 1: the constructions of this thesis are simultaneously free of linear proofs and bilinear pairings.

Off-chain efficiency. The elimination of DVRFwCP's Augmented Secure-DKG round yields total per-request speedup factors of 1.40 at $(n, t) = (5, 3)$, 1.43 at $(7, 4)$, and 7.14 at $(50, 34)$, with corresponding per-node speedup factors of 1.46, 1.48, and 4.43 (Table 6.5). The per-node broadcast shrinks from $(2t+1)\mathbb{G}_1$ in DVRFwCP to a committee-independent $3\mathbb{G}_1$, as confirmed by Table 6.6. The one-time Preprocess cost amortizes to under 85 ms per request for committees up to $(10, 7)$, and remains bounded at 424 ms even at $(50, 34)$ (Table 6.7). The per-phase breakdown at $(10, 7)$ in Table 6.8 shows that the dominant saving comes from the PartialProofGen phase (874.7 ms in DVRFwCP, 441.6 ms in Icy-DVRF), precisely where the Augmented Secure-DKG round was the bottleneck.

CHAPTER 7

ROBUSTNESS, ACCOUNTABILITY, AND THE RANDOMNESS-BEACON LANDSCAPE

7.1 The Robustness/Efficiency Tradeoff

We have stated that Icy-DVRF (Chapter 4) and IcyVeil, which is built on the core of Icy-DVRF (Chapter 5), are not robust and inherited from the abort-and-identify model of FROST. This model, mentioned in Section 2.8, is a deliberate design choice rather than an oversight, and it is the cost of the on-chain efficiency documented in Chapter 6. In this chapter, we examine this cost and argue that it is acceptable in practice by adopting appropriate accountability mechanisms.

The problem is simply stated as follows: A *fully robust* threshold scheme guarantees that a valid output is produced even when up to $t - 1$ participants actively misbehave, but it obtains this guarantee through additional communication rounds and reconstruction machinery. Icy-DVRF construction precisely removes this overhead by adopting an *abort-and-identify* approach: if a participant submits an invalid share, the session is aborted, the responsible party is identified, and an honest retry follows. The output guarantee is thereby weakened from "always achieve an output" to "achieve an output, or find the faulty party, then retry".

In the remainder of this chapter, we argue that this weaker guarantee is sufficient for the deployment setting the thesis targets, decentralized blockchain services, because the missing robustness can be recovered at a different layer. Identifiability turns aborts into accountable events; economic staking turns accountable events into deterrents; combiner redundancy removes the obvious single point of failure; and the host chain's consensus layer turns objective fault evidence into enforceable exclusion. Although the mechanisms are described in the vocabulary of Icy-DVRF, they apply equally to IcyVeil, whose VRF clan-side protocol is identical.

7.2 Cryptographic Identifiability and Economic Disincentives

Unlike fully robust threshold schemes that guarantee output even in the presence of faults, Icy-DVRF and IcyVeil adopt an abort-and-identify model to prioritize on-chain efficiency. While this introduces a potential Denial-of-Service (DoS) vulnerability, the protocol is designed with two primary defense layers.

- **Cryptographic Identifiability:** Each participant P_i provides a partial evaluation γ_i and a corresponding NIZK proof π_i. During the Combine phase, the combiner verifies these proofs individually. If any share is invalid, the specific misbehaving participant is cryptographically identified and excluded from the session.
- **Economic Disincentives (Slashing):** In practical decentralized deployments, such as blockchain-based randomness beacons, this identifiability allows for the integration of a slashing mechanism. By requiring participants to stake collateral, any identified fault or malicious abort can result in an automated financial penalty. This economic pressure ensures that the cost of a sustained DoS attack far outweighs any potential benefit for the adversary, effectively neutralizing the robustness trade-off in production environments.

The two layers complement each other: identifiability is a cryptographic property of the protocol that holds unconditionally, whereas slashing is an economic mechanism added on top of it by the deployment. Identifiability is what makes slashing possible, since a fault that cannot be cryptographically attributed cannot be penalized; Slashing is what converts the ability to identify a faulty node into an active disincentive against acting faulty.

7.3 Resilience against Malicious Combiners

A natural objection to the combiner-centric description of a distributed protocol is that the combiner $\mathcal{C}$ appears to be a single point of failure. While the combiner $\mathcal{C}$ is responsible for collecting and combining partial proofs, its potential for malicious behavior is restricted to service disruption: it cannot forge an output or learn a secret, only stall. To mitigate such Denial-of-Service risks, the protocol provides three structural safeguards.

- **Aggregator Redundancy:** The protocol does not rely on a single combiner; multiple participants can act as aggregators simultaneously. If one $\mathcal{C}$ fails to publish the final VRF output β and proof π, the provers can re-send their shares to an alternative combiner or broadcast them to a public bulletin board.
- **Public Verifiability of Misbehavior:** Since all partial proofs (z_i, c) and commitments are publicly verifiable, a combiner (or any other member) cannot falsely claim a participant is malicious without providing a cryptographic proof of failure. Any

such false report can be detected by honest participants, resulting in the combiner's immediate disqualification.

- **Statelessness:** The combiner $\mathcal{C}$ does not hold any long-lived secret state. This allows the committee to rotate the combiner role frequently or select it randomly for each session, preventing a single point of failure from persistently stalling the proof network.

When working hand-in-hand, these safeguards reduce the combiner from a trusted coordinator to a matter of convenience: a well-behaved combiner makes the common case efficient, but no single combiner is necessary for liveness, and no node (including the combiner) can falsely accuse a participant without detectable evidence.

7.4 Consensus on Node Exclusion

While the constructions identify misbehaving nodes via cryptographic proofs, the final decision to exclude a node from the committee is delegated to the underlying *consensus layer* of the hosting network. To prevent network splits during this process, the protocol provides *publicly verifiable evidence of failure*. Specifically, if a participant P_i submits an invalid partial proof π_i or an incorrect response z_i, the combiner or any honest observer can broadcast this invalid share alongside the prover's public key Y_i.

Because the verification of these shares is deterministic and based on globally known public parameters, all honest nodes will independently reach the same conclusion regarding the node's fault. This *objective fault attribution* ensures that the exclusion process does not rely on subjective leader-based reports, thereby mitigating the risk of forks or network splits during committee reconfigurations.

This design choice is what makes the overall approach coherent: the protocol does not attempt to *enforce* robustness internally, which would reintroduce the overhead it was designed to avoid. Instead, it produces objective, portable evidence of failure and delegates the enforcement decision to the host chain's consensus layer, which already exists to make such decisions network-wide. Robustness is therefore not abandoned but delegated to the layer where it is least expensive to provide.

7.5 Operational Fault-Tolerance Model

To maintain liveness under adversarial or unstable network conditions, Icy-DVRF and IcyVeil operate under an explicit abort-and-identify recovery paradigm rather than deploying heavy threshold reconstruction cycles. We formalize the handling of operational anomalies as follows.

1. **Non-Response Management:** If a chosen participant P_i fails to submit its partial evaluation share (γ_i, π_i) within a synchronized network delta Δ, the combiner drops the current session. Because the protocol requires at least t active shares to combine the random value without structural degradation, the missing entity cannot be bypassed seamlessly in the middle of a phase.

2. **Retry Logic and Session Initialization:** Upon detecting a timeout or an invalid share verification via the partial-proof checks of the $\mathrm{Combine}$ algorithm, the combiner broadcasts an authenticated `ABORT` signal carrying a monotonic epoch counter increment $\tau + 1$. The honest nodes clear the state registers allocated for session τ, reset their pointer states, and await a new execution request.

3. **Combiner Withholding Protection:** A malicious combiner might deliberately withhold valid shares or isolate subsets of nodes to stall the beacon or cause network splits. To mitigate this risk, any participant can act as a secondary combiner by tracking public ledger entries. If the primary combiner does not publish the aggregated output within the designated time window, backup nodes broadcast their independently collected valid share pools, triggering a deterministic aggregation event on-chain.

4. **Committee Exclusion Strategy:** Misbehaving or unresponsive nodes are definitively isolated from future participation. Upon an abort event caused by a cryptographic failure, such as an invalid NIZK proof, the explicit proof of failure is submitted to the base consensus layer. The governance state machine then updates the active participant bitmask, excluding the faulty key from the next active committee matrix without requiring a full DKG key refresh.

7.6 Positioning within the Randomness-Beacon Landscape

A Distributed Randomness Beacon (DRB) is a decentralized system that produces a continuous stream of randomness using cryptographic primitives such as DVRFs; the two are closely related but distinct. Following the recent systematization of the distributed randomness landscape by Choi et al. [18], DRB protocols are classified based on their entropy provision and recovery strategies.

The constructions presented in this thesis are proposed as high-performance engines for the category of DRBs that utilize "No Marginal Entropy" [18]. In these systems, such as **drand** or the **Internet Computer (Dfinity)**, the beacon output is generated through the collective computation of a DVRF or threshold signature primitive without requiring fresh external input in each epoch [18]. Choi et al. identify two primary existing primitives in this space: DDH-DVRF, which suffers from $O(t)$ linear proof sizes affecting verification scalability, and GLOW-DVRF, which offers constant-size proofs but incurs high computational overhead due to bilinear pairing operations [18]. The protocols DVRFwCP and Icy-DVRF bridge this gap by offering a third alternative: primitives that achieve constant-size proofs without the use

of pairings, and in the case of IcyVeil, also achieving output privacy. By optimizing the underlying cryptographic engine, these constructions provide a more efficient building block for production-grade DRB architectures that seek to minimize on-chain settlement costs on high-throughput blockchains.

7.7 Conclusion

We conclude by stating clearly what we affirm in this chapter. We do not warrant that IcyDVRF or IcyVeil are fully robust in the strong cryptographic sense; they are not, and this was a deliberate trade-off made in exchange for the constant-size, pairing-free, low-interaction verification that is the central contribution of these constructions. Instead, we express that the abort-and-identify model, rather than being a liability, composes naturally with the accountability primitives that blockchain deployments already provide. Cryptographic identifiability is exactly the input that a slashing contract requires; objective fault attribution is precisely the input that a consensus layer requires; combiner statelessness is precisely what allows the role rotation that removes the last apparent single point of failure.

One limitation nonetheless remains: In a setting that does not provide an economic or consensus layer to rely on, a fully robust scheme is preferable to the abort-and-identify model, and closing this gap without sacrificing efficiency is an open problem. We return to this problem among the directions for future work in Chapter 8.

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 Summary of Contributions

The constructions in this thesis arose from a mere observation, framed Chapter 1 as a "dilemma": prior distributed verifiable random functions suitable for on-chain use were forced to adopt one of the two constructions, DDH-DVRF or GLOW-DVRF. DDH-DVRF is pairing-free but has an $O(t)$ proof whose on-chain verification cost grows linearly with the size of the committee threshold t. GLOW-DVRF condenses the proof to constant size using bilinear pairings, whose verification cost is large and fixed.

In Chapter 3 we presented DVRFwCP, which addresses these two facets simultaneously. By layering a t-out-of-n threshold structure over a Chaum-Pedersen DLEQ proof and supplying the joint ephemeral commitment via an interaction amongst the parties via Augmented Secure-DKG, DVRFwCP produces a proof of one group element and two scalars, verified without pairings in constant time. This interaction's residual cost is in $O(n^2t)$ group elements, required per evaluation to construct the shared challenge.

In Chapter 4 we presented Icy-DVRF, where we were able to reduce this residual cost. Observing that the role of the Augmented Secure-DKG is exactly the role of the preprocessing and nonce-binding mechanism of FROST, we reduced the interaction from two rounds to one, via precomputed, binding-bound nonces. This way, we were able to decrease the per-evaluation communication costs from $O(n^2t)$ to $O(t)$ while preserving the compact pairing-free proof. The security analysis reduces knowledge-soundness to the discrete logarithm problem through a self-contained Generalized Forking Lemma argument, and explains why the FROST-style binding prevents the Drijvers/Wagner style concurreny attack.

In Chapter 5 we presented IcyVeil, which reinforces Icy-DVRF with output-privacy. Adapting the blinding mechanism of FlexiRand [40] using merely discrete logarithms, IcyVeil lets a requester blind the input, have the VRF clan take the blinded value as input, and in the final stage unblind the result privately, so that an on-chain observer cannot recover the final VRF value, assuming Computational Diffie-Hellman is hard in the group. To the best of our knowledge, IcyVeil is the first pairing-free output-private DVRF.

In Chapter 6, we validate these theoretical claims through live-environment measurements. A Solidity smart-contract deployment on the Ethereum Sepolia testnet confirmed a 43.02% reduction in verification gas relative to the pairing-based GLOW-DVRF.

In Chapter 7, we address the theoretical lack of robustness in Icy-DVRF by demonstrating how the abort-and-identify model integrates naturally with the cryptographic identifiability, economic slashing, and consensus-layer fault attribution inherent in existing blockchain deployments.

8.2 Directions for Future Work

Despite the progress in our vision of scaling DVRFs, several directions remain open, ranging from foundational cryptographic questions to engineering concerns of deployment.

Closing the robustness gap. As identified in Chapter 7, the abort-and-identify model is inherently weaker than full robustness when an underlying economic or consensus layer is absent. It remains an open question whether full robustness, i.e., guaranteed output despite up to $t - 1$ active faults, can be achieved without compromising the constant-size, pairing-free, and low-interaction verification established in this work. Since typical robust mechanisms require costly reconstruction rounds, achieving full robustness while preserving Icy-DVRF's off-chain preprocessability would be a major breakthrough. We are actively making progress toward a positive resolution to this question.

Tightening the security reduction. Our proof of the knowledge-soundness reduction of Section 4.5.5 is not tight: The Generalized Forking Lemma incurs a quadratic loss, also this is further loosened by the need to program the random oracle for every preprocessed commitment. A proof in a stronger model that avoids rewinding, would allow smaller security parameters. This problem is of independent theoretical interest.

Production hardening. The Solidity smart-contracts benchmarked in Chapter 6 were written solely for demonstration rather than production: they are neither gas-optimized nor security-audited. A hardened, security audited implementation, together with an end-to-end integration into a live randomness-beacon service, would be necessary to move these constructions from research prototype to deployable infrastructure.

Accountable DVRFs. While every DVRF discussed so far guarantees that at least a threshold t of participants contributed to the generation of the output, none of these constructions can identify which specific parties participated. To the best of our knowledge, no accountable DVRF schemes currently exist, and designing one without increasing verification costs remains an open problem.

Richer output-privacy policies. IcyVeil grants the privacy of the output to a single requester holding the blinding scalar κ. Generalizations such as threshold unblinding among several parties, time-locked or conditional revealing, or revocable privacy under auditor oversight

would broaden the applicability of the pairing-free output-private randomness.

Post-Quantum instantiation. Construction in this thesis rest ultimately on the hardness of the discrete logarithm problem or its derivatives, and is therefore vulnerable to a sufficiently large quantum computer. The post-quantum verifiable random functions surveyed in Chapter 2, based on lattices [29] or symmetric primitives [15], do not suggest a direct path to a post-quantum distributed VRF that achieves low costs in verification. To the best of our knowledge, this problem remains open.

The title of this thesis names three properties that prior distributed verifiable random functions could not hold at once: pairing-free verification, compact proofs, and output privacy. The progression from DVRFwCP through Icy-DVRF to IcyVeil shows that these properties can in fact be satisfied simultaneously, at the cost of an interaction round that is performed off-chain, and that doing so yields a primitive measurably cheaper to verify on-chain than the pairing-based state of the art. These constructions are not the final word, since robustness, tightness, and post-quantum security all remain open, but they demonstrate that the initial dilemma which has shaped distributed VRF design admits a concrete and efficient resolution.

REFERENCES

[1] B. Adams and D. Feist, EIP-7782: Reduce Block Latency, Ethereum Improvement Proposals, no. 7782, https://eips.ethereum.org/EIPS/eip-7782, Last accessed on 28/07/2026, 2024.

[2] A. R. Ağırtaş, A. B. Özer, Z. Saygı, and O. Yayla, Distributed Verifiable Random Function with Compact Proof, in S. Dolev, M. Elhadad, M. Kutyłowski, and G. Persiano, editors, *Cyber Security, Cryptology, and Machine Learning*, pp. 119–134, Springer Nature Switzerland, Cham, 2025.

[3] A. R. Ağırtaş, A. B. Özer, Z. Saygı, and O. Yayla, Icy-DVRF: A Distributed Verifiable Random Function based on FROST signatures, IEEE Access, 14, pp. 105882–105909, 2026.

[4] A. R. Ağırtaş, A. B. Özer, Z. Saygı, and O. Yayla, IcyVeil: An Output-Private DVRF Based on Icy-DVRF, To appear in BCAA 2026. Cryptology ePrint Archive, Paper 2026/1138, https://eprint.iacr.org/2026/1138, Last accessed on 07/07/2026, 2026.

[5] A. Akhunov, E. Ben Sasson, T. Brand, L. Guthmann, and A. Levy, EIP-2028: Transaction data gas cost reduction, Ethereum Improvement Proposals, no. 2028, https://eips.ethereum.org/EIPS/eip-2028, Last accessed on 07/07/2026, 2019.

[6] M. Bellare, E. C. Crites, C. Komlo, M. Maller, S. Tessaro, and C. Zhu, Better than Advertised Security for Non-interactive Threshold Signatures, in Y. Dodis and T. Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pp. 517–550, Springer, Cham, 2022.

[7] M. Bellare, W. Dai, and L. Li, The Local Forking Lemma and Its Application to Deterministic Encryption, in S. D. Galbraith and S. Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pp. 607–636, Springer, Cham, 2019.

[8] M. Bellare and O. Goldreich, On Defining Proofs of Knowledge, in *Advances in Cryptology – CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pp. 390–420, Springer, Berlin, Heidelberg, 1992.

[9] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko, The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme, Journal of Cryptology, 16(3), pp. 185–215, 2003.

[10] M. Bellare and G. Neven, Multi-Signatures in the Plain Public-Key Model and a General Forking Lemma, in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pp. 390–399, 2006.

[11] F. Benhamouda, T. Lepoint, J. Loss, M. Orrù, and M. Raykova, On the (in)Security of ROS, Journal of Cryptology, 35(4), p. 25, 2022.

[12] D. Boneh, The Decision Diffie-Hellman Problem, in *Algorithmic Number Theory (ANTS 1998)*, volume 1423 of *Lecture Notes in Computer Science*, pp. 48–63, Springer, Berlin, Heidelberg, 1998.

[13] D. Boneh, B. Lynn, and H. Shacham, Short Signatures from the Weil Pairing, Journal of Cryptology, 17(4), pp. 297–319, 2004.

[14] L. Breidenbach, C. Cachin, B. Chan, A. Coventry, S. Ellis, A. Juels, F. Koushanfar, A. Miller, B. Magauran, D. Moroz, et al., Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks, Whitepaper, https://research.chain.link/whitepaper-v2.pdf, Last accessed on 07/07/2026, 2021.

[15] M. Buser, R. Dowsley, M. F. Esgin, S. Kasra Kermanshahi, V. Kuchta, J. K. Liu, R. C.-W. Phan, and Z. Zhang, Post-Quantum Verifiable Random Function from Symmetric Primitives in PoS Blockchain, in *European Symposium on Research in Computer Security*, pp. 25–45, Springer, 2022.

[16] J. Camenisch, A. Cerulli, D. Derler, M. Drijvers, M. Dubovitskaya, J. Groth, T. Hanke, G. Neven, Y.-A. Pignolet, V. Shoup, B. Tackmann, and D. Williams, The Internet Computer for Geeks, Cryptology ePrint Archive, Paper 2022/087, https://eprint.iacr.org/2022/087, Last accessed on 07/07/2026, 2022.

[17] D. Chaum and T. P. Pedersen, Wallet Databases with Observers, in *Advances in Cryptology — CRYPTO '92*, pp. 89–105, Springer, 1992.

[18] K. Choi, A. Manoj, and J. Bonneau, SoK: Distributed Randomness Beacons, in *2023 IEEE Symposium on Security and Privacy (SP)*, pp. 75–92, 2023.

[19] D. Connolly, C. Komlo, I. Goldberg, and C. A. Wood, The Flexible Round-Optimized Schnorr Threshold (FROST) Protocol for Two-Round Schnorr Signatures, RFC 9591, RFC Editor, https://www.rfc-editor.org/info/rfc9591, Last accessed on 30/07/2026, 2024.

[20] R. Cramer, I. Damgård, and B. Schoenmakers, Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols, in *Advances in Cryptology – CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pp. 174–187, Springer, Berlin, Heidelberg, 1994.

[21] E. Crites, C. Komlo, and M. Maller, How to Prove Schnorr Assuming Schnorr: Security of Multi- and Threshold Signatures, Cryptology ePrint Archive, Paper 2021/1375, https://eprint.iacr.org/2021/1375, Last accessed on 07/07/2026, 2021.

[22] I. Damgård, On Σ-protocols, Lecture Notes, Cryptologic Protocol Theory, Aarhus University, https://www.cs.au.dk/~ivan/Sigma.pdf, Last accessed on 07/07/2026, 2010.

[23] B. David, P. Gaži, A. Kiayias, and A. Russell, Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain, in J. B. Nielsen and V. Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pp. 66–98, Springer International Publishing, Cham, 2018.

[24] W. Diffie and M. E. Hellman, New Directions in Cryptography, IEEE Transactions on Information Theory, 22(6), pp. 644–654, 1976.

[25] Y. Dodis, Efficient Construction of (Distributed) Verifiable Random Functions, in *Public Key Cryptography — PKC 2003*, pp. 1–17, Springer, 2003.

[26] Y. Dodis and A. Yampolskiy, A Verifiable Random Function with Short Proofs and Keys, in *Public Key Cryptography – PKC 2005*, pp. 416–431, Springer, 2005.

[27] M. Drijvers, K. Edalatnejad, B. Ford, E. Kiltz, J. Loss, G. Neven, and I. Stepanovs, On the Security of Two-Round Multi-Signatures, in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1084–1101, 2019.

[28] B. Edgington, BLS12-381 For The Rest Of Us, Online article, https://hackmd.io/@benjaminion/bls12-381, Last accessed on 07/07/2026, 2022.

[29] M. F. Esgin, V. Kuchta, A. Sakzad, R. Steinfeld, Z. Zhang, S. Sun, and S. Chu, Practical Post-quantum Few-Time Verifiable Random Function with Applications to Algorand, in N. Borisov and C. Diaz, editors, *Financial Cryptography and Data Security*, pp. 560–578, Springer Berlin Heidelberg, Berlin, Heidelberg, 2021.

[30] Ethereum Foundation, `py_ecc`: Python implementation of ECC pairing and BLS12-381 curve operations, GitHub repository, https://github.com/ethereum/py_ecc, Last accessed on 07/07/2026, 2024.

[31] P. Feldman, A practical scheme for non-interactive verifiable secret sharing, in *28th Annual Symposium on Foundations of Computer Science (SFCS 1987)*, pp. 427–438, 1987.

[32] A. Fiat and A. Shamir, How to Prove Yourself: Practical Solutions to Identification and Signature Problems, in *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pp. 186–194, Springer, Berlin, Heidelberg, 1987.

[33] D. Galindo, J. Liu, M. Ordean, and J.-M. Wong, Fully Distributed Verifiable Random Functions and their Application to Decentralised Random Beacons, in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 88–102, 2021.

[34] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, Secure Distributed Key Generation for Discrete-Log Based Cryptosystems, Journal of Cryptology, 20, pp. 51–83, 2007.

[35] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, Algorand: Scaling Byzantine Agreements for Cryptocurrencies, in *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP)*, pp. 51–68, ACM, 2017.

[36] S. Goldberg, L. Reyzin, D. Papadopoulos, and J. Včelák, Verifiable Random Functions (VRFs), RFC 9381, https://www.rfc-editor.org/info/rfc9381, Last accessed on 28/06/2026, 2023.

[37] S. Goldwasser, S. Micali, and C. Rackoff, The Knowledge Complexity of Interactive Proof Systems, SIAM Journal on Computing, 18(1), pp. 186–208, 1989.

[38] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu, TOPPSS: Cost-Minimal Password-Protected Secret Sharing Based on Threshold OPRF, in *Applied Cryptography and Network Security (ACNS 2017)*, pp. 39–58, Springer, Cham, Switzerland, 2017.

[39] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu, TOPPSS: Cost-Minimal Password-Protected Secret Sharing Based on Threshold OPRF, Cryptology ePrint Archive, Paper 2017/363, https://eprint.iacr.org/2017/363, Last accessed on 30/07/2026, 2017.

[40] A. Kate, E. V. Mangipudi, S. Maradana, and P. Mukherjee, FlexiRand: Output Private (Distributed) VRFs and Application to Blockchains, in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1776–1790, 2023.

[41] H. Kılınç Alper, BABE: Blind assignment for blockchain extension, Web3 Foundation Research, https://research.parity.io/Polkadot/protocols/block-production/Babe, Last accessed on 07/07/2026, 2019.

[42] C. Komlo and I. Goldberg, FROST: Flexible Round-Optimized Schnorr Threshold Signatures, in *Selected Areas in Cryptography: 27th International Conference (SAC 2020)*, pp. 34–65, Springer-Verlag, Berlin, Heidelberg, 2020.

[43] V. Kuchta and M. Manulis, Unique Aggregate Signatures with Applications to Distributed Verifiable Random Functions, in M. Abdalla, C. Nita-Rotaru, and R. Dahab, editors, *Cryptology and Network Security*, pp. 251–270, Springer International Publishing, Cham, 2013.

[44] S. Micali, M. Rabin, and S. Vadhan, Verifiable random functions, in *40th Annual Symposium on Foundations of Computer Science*, pp. 120–130, 1999.

[45] D. Papadopoulos, D. Wessels, S. Huque, M. Naor, J. Včelák, L. Reyzin, and S. Goldberg, Making NSEC5 Practical for DNSSEC, Cryptology ePrint Archive, Paper 2017/099, https://eprint.iacr.org/2017/099, Last accessed on 07/07/2026, 2017.

[46] S. Patel, G. Persiano, J. Y. Seo, and K. Yeo, Efficient Secret Sharing for Large-Scale Applications, in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, p. 3065–3079, Association for Computing Machinery, New York, NY, USA, 2024.

[47] T. P. Pedersen, A Threshold Cryptosystem without a Trusted Party, in D. W. Davies, editor, *Advances in Cryptology — EUROCRYPT '91*, pp. 522–526, Springer Berlin Heidelberg, Berlin, Heidelberg, 1991.

[48] T. P. Pedersen, Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing, in J. Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pp. 129–140, Springer Berlin Heidelberg, Berlin, Heidelberg, 1992.

[49] D. Pointcheval and J. Stern, Security Arguments for Digital Signatures and Blind Signatures, Journal of Cryptology, 13(3), pp. 361–396, 2000.

[50] C.-P. Schnorr, Efficient Identification and Signatures for Smart Cards, in *Advances in Cryptology — CRYPTO '89*, pp. 239–252, Springer, 1990.

[51] A. Shamir, How to share a secret, Communications of the ACM, 22(11), pp. 612–613, 1979.

[52] Supra Research, Supra VRF Service, Technical Whitepaper, https://supra.com/documents/VRF-Service-Whitepaper.pdf, Last accessed on 07/07/2026, 2023.

[53] A. Vlasov, K. Olson, A. Stokes, and A. Sanso, EIP-2537: Precompile for BLS12-381 curve operations, Ethereum Improvement Proposals, no. 2537, https://eips.ethereum.org/EIPS/eip-2537, Last accessed on 07/07/2026, 2020.

[54] Y. Zhao, Z. Wan, Z. Guan, G. Li, and M. Guo, ClusterGuard: Secure Clustered Aggregation for Federated Learning With Robustness, IEEE Transactions on Dependable and Secure Computing, 23(3), pp. 7043–7058, 2026.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Özer, Arda Buğra
Nationality: Turkish

EDUCATION

Ph.D.	Cryptography, METU IAM	Ongoing
M.Sc.	Mathematics, METU	2002
B.Sc.	Mathematics, METU	1999
High School	Ankara Atatürk Anadolu High School	1994

PROFESSIONAL EXPERIENCE

2015–2023	METU Northern Cyprus Campus	Senior Instructor
2011–2015	METU Northern Cyprus Campus	Teaching Assistant
2010–2011	Atılım University	Visiting Instructor
2002–2009	Michigan State University	Graduate Teaching Assistant
1999–2002	METU	Graduate Teaching Assistant

PUBLICATIONS

A. R. Ağırtaş, A. B. Özer, Z. Saygı, O. Yayla. *Distributed Verifiable Random Function with Compact Proof.* In: Cyber Security, Cryptology, and Machine Learning (CSCML 2025), Springer, 2025, pp. 119–134.

A. R. Ağırtaş, A. B. Özer, Z. Saygı, O. Yayla. *Icy-DVRF: A Distributed Verifiable Random Function based on FROST Signatures.* IEEE Access, vol. 14, 2026, pp. 105882–105909.

A. R. Ağırtaş, A. B. Özer, Z. Saygı, O. Yayla. *IcyVeil: An Output-Private DVRF Based on Icy-DVRF.* To appear in BCAA 2026.

A. B. Özer, E. Saygı, Z. Saygı. *Domination type parameters of Pell graphs.* ARS Math. Contemp. 23 (2023), #1.03.

TECHNICAL SKILLS

- **Core Languages:** Proficient in C/C++, Python, Rust, Go
- **Blockchain & ZK:** Solidity, Cairo, Noir, ZoKrates; Hands-on with Hardhat, Ethers, Viem, libSTARK
- **Cryptography:** Experience with SAGEmath; Familiarity with FHE libraries (OpenFHE, HELib, SEAL)
- **Data Analysis:** NumPy, Pandas
- **Machine Learning & AI:** TensorFlow, Scikit-learn, Deep Learning, NLP, Agentic AI, LLMs, LangChain, LangGraph, OpenAI API, Decentralized AI (ZKML)
- **Development Tools:** Git, Docker, LaTeX, Microsoft Office

LANGUAGES

- **Turkish:** Native
- **English:** Fluent