

COLLUSION-RESISTANT TLS ATTESTATION PROTOCOLS: A VERIFIABLE,
MODULAR FRAMEWORK FOR DECENTRALIZED APPLICATIONS

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

BY

UĞUR ŞEN

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

FEBRUARY 2026

Approval of the thesis:

**COLLUSION-RESISTANT TLS ATTESTATION PROTOCOLS: A
VERIFIABLE, MODULAR FRAMEWORK FOR DECENTRALIZED
APPLICATIONS**

submitted by **UĞUR ŞEN** in partial fulfillment of the requirements for the degree of
**Doctor of Philosophy in Cryptography Department, Middle East Technical Uni-
versity** 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, **Institute of Applied Mathematics, METU**

Assist. Prof. Dr. Murat Osmanoğlu
Co-supervisor, **Dept. of Comp. Eng., Ankara University**

Examining Committee Members:

Prof. Dr. Fatih Sulak
Department of Mathematics, Atılım University

Assoc. Prof. Dr. Oğuz Yayla
Institute of Applied Mathematics, METU

Prof. Dr. Ali Aydın Selçuk
Department of Computer Engineering, TOBB-ETU

Assoc. Prof. Dr. Bülent Tuğrul
Department of Computer Science, Ankara University

Assist. Prof. Dr. Buket Özkaya
Institute of Applied Mathematics, METU

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: UĞUR ŞEN

Signature :

ABSTRACT

COLLUSION-RESISTANT TLS ATTESTATION PROTOCOLS: A VERIFIABLE, MODULAR FRAMEWORK FOR DECENTRALIZED APPLICATIONS

ŞEN, Uğur

Ph.D., Department of Cryptography

Supervisor : Assoc. Prof. Dr. Oğuz Yayla

Co-Supervisor : Assist. Prof. Dr. Murat Osmanoğlu

FEBRUARY 2026, 99 pages

A significant portion of today’s accessible data is stored on centralized servers and is typically accessed through the Transport Layer Security (TLS) protocol, which provides confidentiality and integration guarantees. However, blockchain-based systems cannot natively consume such off-chain data, as TLS was not designed to produce publicly verifiable evidence that can be validated by smart contracts. To address this limitation, a class of protocols commonly referred to as Designed Commitment TLS (DCTLS) or zkTLS has emerged, enabling privacy-preserving attestations derived from TLS sessions without requiring any modification to server-side deployments. Despite their practicality, existing DCTLS constructions rely on designated verifiers, which fundamentally limit public verifiability and introduce vulnerability to prover-verifier collusion. In such settings, a malicious prover and verifier can jointly deviate from the protocol to produce fraudulent attestations that remain indistinguishable from honest executions. Prior attempts to mitigate this issue, including trusted execution environments (TEEs), decentralized oracle networks (DONs), and blind-signature-based approaches, either impose strong trust assumptions, introduce high prover-side complexity, or incur significant scalability and efficiency costs. In this thesis, we develop a modular framework for collusion-resistant TLS attestations that generalizes existing DCTLS constructions. We minimize the designated and trusted verifier assumption by multiplying the number of verifiers without compromising ef-

efficiency, achieved through joint randomness via Distributed Verifiable Random Functions (DVRF). We first demonstrate how Distributed Verifiable Random Functions (DVRF) can be integrated with DECO to enable decentralized storage-based attestation protocol. By generating joint randomness via DVRF, each verifier can bind itself to the session and independently reason about the correctness of the execution, thereby reducing reliance on a single trusted party. To address the scalability challenges of decentralized verification, we then refine this construction by replacing decentralized storage with Threshold Signature Schemes (TSS), thereby rendering attestations compact and efficient. Although the literature contains concrete instances such as DECO and Distefano, it lacks a generalized construction that characterizes DCTLS protocols as a unified class. Therefore, we first provide a general formalization of DCTLS protocols. Based on this formalization, we then derive an exportable abstraction, denoted as dx-DCTLS, and show how DECO and Distefano can be transformed into dx-DCTLS by replacing non-verifiable components with verifiable cryptographic counterparts. This abstraction serves as a unifying layer that allows existing DCTLS protocols to be extended without altering the underlying TLS server infrastructure. On top of dx-DCTLS, we present a collusion-minimized attestation framework in which the verifier role is distributed across a configurable set of auxiliary verifiers. Following the previous construction, we integrate the proposed dx-DCTLS with Distributed Verifiable Random Functions (DVRFs) and Threshold Signature Schemes (TSS), yielding a framework that supports *t-out-of-n* consensus. Crucially, the number of auxiliary verifiers is decoupled from the core TLS interaction, ensuring that prover complexity remains $\mathcal{O}(1)$ while collusion resistance scales with the threshold parameter t . We give a game-based formalization of threshold attestation unforgeability that captures adversarial behaviors specific to multi-verifier environments. Under this definition, we provide a game-based security proof under standard cryptographic assumptions. We evaluate practicality through our prototype implementation of the DVRF-TSS layer and a performance analysis of dx-DCTLS, showing that the additional overhead remains modest even at large threshold sizes. Finally, through a realistic TLS attestation use case, we demonstrate that the proposed framework enables privacy-preserving and scalable blockchain applications without relying on trusted hardware, a single authority, or blind-signature mechanisms.

Keywords: zkTLS, DCTLS, TLS, Attestation, Smart Contracts, Collusion

ÖZ

MUVAZAAYA DAYANIKLI TLS TASDİK PROTOKOLLERİ: MERKEZİYETSİZ UYGULAMALAR İÇİN DOĞRULANABİLİR VE MODÜLER BİR YAKLAŞIM

ŞEN, Uğur

Doktora, Kriptografi Bölümü

Tez Yöneticisi : Doç. Dr. Oğuz Yayla

Ortak Tez Yöneticisi : Dr. Öğr. Üyesi Murat Osmanoğlu

Şubat 2026, 99 sayfa

Günümüzde erişilebilir verilerin önemli bir bölümü merkezi sunucularda saklanmakta ve bu verilere genellikle gizlilik ve bütünlük garantileri sağlayan Transport Layer Security (TLS) protokolü aracılığıyla erişilmektedir. Ancak blokzincir tabanlı sistemler, TLS'nin akıllı sözleşmeler tarafından doğrulanabilir kamusal kanıtlar üretmek üzere tasarlanmamış olması nedeniyle bu tür zincir dışı verileri doğrudan tüketememektedir. Bu sınırlamayı aşmak amacıyla, Tasarım Taahhütlü TLS (Designed Commitment TLS, DCTLS) veya zkTLS olarak adlandırılan bir protokol sınıfı ortaya çıkmış; bu protokoller, sunucu tarafında herhangi bir değişiklik gerektirmeden TLS oturumlarından türetilen gizliliği koruyan doğrulanabilir tasdiklerin üretilmesini mümkün kılmıştır. Bununla birlikte, mevcut DCTLS yapıları belirlenmiş doğrulayıcılara dayanmaktadır. Bu durum kamusal doğrulanabilirliği sınırlamakta ve kanıtlayıcı-doğrulayıcı işbirliği (muvazaa) saldırılarına karşı yapısal bir zayıflık oluşturmaktadır. Muvazaa durumunda, kötü niyetli bir kanıtlayıcı ve doğrulayıcı protokolden birlikte saparak, dürüst çalışmalardan ayırt edilemeyen sahte tasdikler üretebilmektedir. Literatürde muvazaanın etkisini azaltmak amacıyla güvenilir yürütme ortamları (Trusted Execution Environments, TEE), merkeziyetsiz oracle ağları (Decentralized Oracle Networks, DON) ve kör imza tabanlı yaklaşımlar kullanılmış olsa da, bu çözümler ya güçlü güven varsayımları getirmekte, ya kanıtlayıcı tarafında yüksek karmaşıklık doğurmakta ya da ölçeklenebilirlik ve verimlilik açısından ciddi maliyetler yaratmak-

tadır. Bu tezde, mevcut DCTLS yapılarını genelleştiren ve belirlenmiş doğrulayıcı varsayımını minimize eden, modüler bir muvazaaya dayanıklı TLS tasdik çerçevesi geliştirilmektedir. Bu kapsamda, doğrulayıcı sayısı artırılarak tekil güven varsayımı minimize edilmekte; bu işlem, Dağıtık Doğrulanabilir Rastgele Fonksiyonlar (Distributed Verifiable Random Functions, DVRF) aracılığıyla üretilen ortak rastgelelik sayesinde, verimlilikten ödün vermeden gerçekleştirilmektedir. İlk olarak, DVRF'nin DECO protokolü ile nasıl bütünleştirilebileceği gösterilerek, merkeziyetsiz depolama tabanlı TLS tasdik protokolü önerilmiştir. Ortak rastgelelik üretimi sayesinde her bir doğrulayıcı, oturuma kriptografik olarak bağlanabilmekte ve yürütmenin doğruluğu hakkında bağımsız bir güvence elde edebilmektedir; böylece tekil güvenilir taraflara olan bağımlılık azaltılmaktadır. Merkeziyetsiz doğrulamanın ölçeklenebilirlik sorunlarını ele almak amacıyla, bu yapı daha sonra merkeziyetsiz depolama yerine Eşik İmza Şemaları (Threshold Signature Schemes, TSS) kullanılarak kompakt hale getirilmiştir; bu sayede tasdikler daha kompakt ve verimli hale getirilmektedir. Devamında, literatürde DECO ve Distefano gibi somut örnekler bulunmasına rağmen, DCTLS protokollerini formel olarak tanımlayan genelleştirilmiş bir yapı mevcut değildir. Bu nedenle, öncelikle DCTLS protokollerinin genel bir formalizasyonu sunulmaktadır. Bu formal tanım üzerine inşa edilen ve dx-DCTLS olarak adlandırılan dışa aktarılabilir bir soyutlama türetilmektedir. dx-DCTLS, doğrulanabilir olmayan kriptografik bileşenlerin doğrulanabilir karşılıklarıyla değiştirilmesi yoluyla TLS tasdiklerinin üçüncü taraflarca doğrulanabilmesini mümkün kılmakta ve mevcut DCTLS protokollerinin temel TLS sunucu altyapısına müdahale edilmeden genişletilmesine olanak sağlamaktadır. dx-DCTLS üzerine inşa edilen muvazaaya dayanıklı çerçevede, doğrulayıcı rolü yapılandırılabilir sayıda yardımcı doğrulayıcıya dağıtılmaktadır. Önceki önerilerde kullanıldığı üzere, dx-DCTLS önerisi, Dağıtık Doğrulanabilir Rastgele Fonksiyonlar (DVRF) ve Eşik İmza Şemaları (TSS) ile entegre edilerek t -out-of- n uzlaşmasını destekleyen bir çerçeve elde edilmektedir. Ayrıca, yardımcı doğrulayıcı sayısı, çekirdek TLS etkileşiminden bağımsızdır; bu sayede kanıtlayıcı tarafındaki karmaşıklık $\mathcal{O}(1)$ düzeyinde kalırken, muvazaaya karşı dayanıklılık eşik parametresi t ile ölçeklenmektedir. Önerilen çerçevenin güvenliği, çoklu doğrulayıcı ortamlarına özgü saldırgan davranışları kapsayan oyun tabanlı bir eşik tasdik sahteciliği tanımı ile formal olarak tanımlanmıştır. Bu tanım altında, standart kriptografik varsayımlar kullanılarak oyun tabanlı bir güvenlik ispatı sunulmaktadır. Önerilen çerçevenin pratik uygulanabilirliğini değerlendirmek amacıyla, DVRF-TSS katmanının bir prototip uygulaması ve dx-DCTLS için performans analizi gerçekleştirilmektedir. Elde edilen sonuçlar, ek yükün yüksek eşik boyutlarında dahi sınırlı kaldığını göstermektedir. Son olarak, gerçekçi bir TLS tasdik kullanım senaryosu, önerilen çerçevenin güvenilir donanıma, tekil bir otoriteye ya da kör imza mekanizmalarına ihtiyaç duymaksızın gizliliği koruyan ve ölçeklenebilir blokzincir uygulamalarına olanak sağladığı ortaya konmaktadır.

Anahtar Kelimeler: zkTLS, DCTLs, TLS, Tasdikleme, Akıllı Kontraktlar, Muvazaa

To the pursuit of truth

ACKNOWLEDGMENTS

I would like to sincerely thank my supervisor, Assoc. Prof. Dr. Oğuz Yayla, for his valuable guidance and constructive feedback throughout this doctoral study. His directions and thoughtful suggestions played an important role in shaping this thesis. I am grateful for his supportive and professional approach during the research process.

I am grateful to my co-advisor, Assist. Prof. Dr. Murat Osmanoğlu for his valuable contributions to this study through our years of in-person discussions and his innovative ideas on blockchain technologies. His insights and perspectives significantly enriched the intellectual development of this thesis.

I would like to express my deepest respect and gratitude to Assoc. Prof. Dr. Ali Doğanaksoy, with whom I began this doctoral journey as my supervisor. Beyond his technical expertise, he taught us the values of integrity, generosity, and humanity in academia. I am sincerely thankful for the intellectual space he created through cryptography meetings and the opportunities he consistently offered to his students. We were deeply saddened by his recent passing, and I will carry his ideas, guidance, and academic spirit forward with great respect.

I would like to thank Prof. Dr. Ali Aydın Selçuk for guiding our meetings together with Assist. Prof. Dr. Murat Osmanoğlu. I am grateful for his valuable guidance during our technical discussions, his unique and insightful feedback on security topics, and his generosity in opening his office to us. The welcoming atmosphere he created, always accompanied by excellent coffee, as well as our joint runs, provided meaningful spaces for discussion, which I found both motivating and intellectually enriching.

I would like to thank the members of my dissertation committee, Prof. Dr. Fatih Sulak, Assoc. Prof. Dr. Bülent Tuğrul, and Assist. Prof. Dr. Buket Özkaya, for their time, careful evaluation, and valuable comments, which contributed to the improvement of this thesis.

I would like to thank the Council of Higher Education of Türkiye (YÖK) for supporting my doctoral studies through the 100/2000 Doctoral Scholarship Program.

I would like to thank Dr. Ahmet Ramazan Ağırtaş, with whom I began METU-IAM Cryptography, and whose expertise in threshold cryptography I trust, for the insightful discussions and key points he shared, which contributed to the development of this study.

I would also like to thank Onur İnanç Doğruyol for the enjoyable and enlightening conversations we shared on the latest developments in blockchain technologies, and for his support and guidance in helping me enter the blockchain field.

My thanks also go to the members of the Oasis Writers working group, where doctoral students come together for structured, timed working sessions and mutual support. This shared space, grounded in the simple reminder that we are not alone, played an important role in sustaining my focus, motivation, and sense of belonging throughout the doctoral process.

I would also like to thank the METU-IAM Cryptography staff for their support, respect, and dedication, and for assisting me with kindness and professionalism.

I would also like to express my sincere gratitude to my company, IFT, and to my colleagues for providing such a supportive and intellectually stimulating environment throughout this journey. Their flexibility, trust, and encouragement have been invaluable. My sincere thanks also go to FAME CRYPT for their support and for fostering a collaborative space.

I also thank my parents and brother for sharing both the joys and challenges of this journey with me, for celebrating each achievement together, and for standing by me with care and understanding throughout this process.

I express my heartfelt gratitude to my beloved wife, Dr. Dilara Özel Şen, for encouraging my scientific work not only through her words but also through her actions, for her thoughtful advice, and for giving me the strength to continue during moments of exhaustion. I am also grateful to our cats, Minnoş and Erdiñ Gauss, for being an essential part of our home and for the calm and comfort they brought, which helped me stay focused and grounded.

TABLE OF CONTENTS

ABSTRACT	vii
ÖZ	ix
ACKNOWLEDGMENTS	xiii
TABLE OF CONTENTS	xv
LIST OF TABLES	xix
LIST OF FIGURES	xxi
LIST OF ABBREVIATIONS	xxiii
CHAPTERS	
1 INTRODUCTION	1
1.1 Motivation	4
1.2 Contribution	6
1.3 Organization	8
2 PRELIMINARIES	11
2.1 Transport Layer Security (TLS)	12
2.1.1 TLS 1.2: Configurable Authenticated Key Exchange	12
2.1.2 TLS 1.3: Streamlined AKE with Forward Secrecy by Default	13

2.2	Zero-Knowledge Proofs (ZKP)	14
2.3	Secure Multiparty Computation (MPC)	15
2.3.1	Two-Party Computation (2PC)	16
2.3.2	Security Definitions of MPC	16
2.4	Designated-Commitment TLS (DCTLS) Protocols	17
2.4.1	DECO Protocol	18
2.4.2	Distefano Protocol	20
2.5	Distributed Verifiable Functions (DVRF)	22
2.6	Threshold Signature Schemes (TSS)	23
2.7	Verifiable 2PCs (v2PCs)	24
2.8	Collaborative SNARKs (Co-SNARKs)	25
3	OPTIMISTIC MULTI-VERIFIERS DECO WITH DECENTRALIZED STORAGE	27
3.1	Building Blocks	29
3.1.1	Optimistic Approach	29
3.1.2	Joint Randomness	30
3.1.3	Decentralized Storage	31
3.2	The Proposed Protocol	32
3.2.1	Entities	32
3.2.2	Construction	33
3.3	Protocol Analysis	36
3.3.1	Security Discussion	36

3.3.2	Efficiency Analysis	37
3.4	Summary	38
4	COMPACT MULTI-VERIFIERS DECO WITH TRESHOLD SIG- NATURE SCHEME	39
4.1	The Proposed System Architecture	41
4.1.1	Construction	41
4.2	System Analysis	43
4.2.1	Security Discussion	43
4.2.2	Efficiency Analysis	45
4.3	A Use-Case Example	47
4.4	Summary	49
5	DESIGNED EXPORTABLE DCTLs (DX-DCTLs)	51
5.1	Building Blocks	53
5.1.1	Game-based security notations	53
5.1.2	DVRF Security Definition	54
5.1.3	TSS Security Definition	55
5.1.4	v2PC Security Definition	56
5.1.5	Co-SNARK Security Definition	57
5.2	Problem Formulation	58
5.3	Syntax	60
5.4	Security Definitions	62
5.4.1	Threshold Attestation Unforgeability	62

5.4.2	Adversarial Oracles	64
5.5	Strawman Protocols	65
5.5.1	nPC-based Strawman Protocol	66
5.5.2	DVRF-Integrated DCTLS Strawman Protocol	67
5.6	Collusion-Minimized TLS Attestation Protocol	69
5.6.1	The dx-DCTLS Abstraction: From DCTLS to dx-DCTLS	69
5.6.2	The Full Protocol $\Pi_{\text{coll-min}}$	71
5.7	Instantiating dx-DCTLS from DECO and Distefano	76
5.7.1	Modified DECO as dx-DCTLS.	77
5.7.2	Modified Distefano as dx-DCTLS.	77
5.8	Performance Evaluations	78
5.9	A Use-Case Examples	84
5.9.1	Use Case 1: Confidential Binary Options	85
5.9.2	Use Case 2: Off-Chain Credit and Income Verification	86
5.10	Summary	88
6	CONCLUSION	89
	REFERENCES	93
	CURRICULUM VITAE	99

LIST OF TABLES

Table 4.1 Comparison of Signature Schemes from [46], Where t Denotes the Threshold Parameter Used in DVRF and TSS Constructions	46
Table 5.1 Comparative Analysis of TLS Attestation Architectures [46]	83

LIST OF FIGURES

Figure 1.1 Overview of a Multi-Verifier Attestation Workflow Designed to Mitigate Prover-Verifier Collusion by Distributing Verification Across Auxiliary Verifiers	3
Figure 2.1 An Overview of the DECO [56] Three-Party Handshake	19
Figure 3.1 Overview of the Proposed Three-Party Handshake [45]	34
Figure 4.1 Overview of the Integration of DECO, DVRF, and TSS, Adapted from [46]	41
Figure 4.2 Overview of the Proposed Scheme in an Example Flight Delay Insurance Use Case [46]	48
Figure 5.1 DVRF Uniqueness Experiment adapted from [49]	55
Figure 5.2 Overview of the DCTLS-Based Workflow for Smart Contracts [49]	59
Figure 5.3 Threshold Attestation Unforgeability Experiment [49]	64
Figure 5.4 General Overview of DCTLS Protocols with Server, Prover, and Verifier	66
Figure 5.5 nPC-Based Strawman Protocol Extending to an n -Verifier Setting [49]	67
Figure 5.6 DVRF-Integrated Strawman Protocol Extending to an n -Verifier Setting [49]	68
Figure 5.7 Unforgeability Experiment of dx-DCTLS [49]	70
Figure 5.8 The Proposed Collusion-Minimized Protocol $\Pi_{\text{coll-min}}$ Construction [49]	72
Figure 5.9 Building DVRF Uniqueness Adversary \mathcal{B}_D [49]	74
Figure 5.10 Building dx-DCTLS Unforgeability Adversary \mathcal{B}_X [49]	76

Figure 5.11 LAN Performance Comparison Showing the Execution Time of the DVRF-TSS Extension, Including the Initial DKG Phase Followed by DVRF-Based Randomness Generation and Threshold Signature (TSS) Creation, for Varying t -out-of- n Auxiliary Verifier Configurations [49] . . .	80
Figure 5.12 Network Communication Cost of DVRF-TSS, Including DKG [49]	80
Figure 5.13 Network Communication Cost of DVRF-TSS, Excluding DKG [49]	81
Figure 5.14 Execution Time of the DVRF-TSS under Two WAN Profiles (WAN1 and WAN2), Where Color Represents the WAN Profile and Line Style Distinguishes Executions With and Without the DKG Phase, for Varying t -out-of- n Auxiliary Verifier Configurations [49]	81

LIST OF ABBREVIATIONS

ZKP	Zero-Knowledge Proof
SC	Smart Contract
TLS	Transport Layer Security
dApp	Decentralized Application
DCTLS	Designated Commitment TLS
dx-DCTLS	Designated-Exportable DCTLS
zkTLS	Zero-Knowledge TLS
DON	Decentralized Oracle Network
TEE	Trusted Execution Environment
DVRF	Distributed Verifiable Random Function
DKG	Distributed Key Generation
TSS	Threshold Signature Scheme
MPC	Multi-Party Computation
2PC	Two-Party Computation
nPC	N-Party Computation
PPT	Probabilistic Polynomial Time
FROST	Flexible Round-Optimized Schnorr Threshold
ECDSA	Elliptic Curve Digital Signature Algorithm
PRF	Pseudo Random Function
MAC	Message Authentication Code
IPFS	Interplanetary File System
RTT	Round Trip Time

CHAPTER 1

INTRODUCTION

Blockchains operate as self-contained computation environments, inherently isolated from external data sources. This architectural limitation poses a challenge for decentralized applications (dApps) that must respond to real-world events. To address this gap, *oracles* have been developed as trusted intermediaries that retrieve data from off-chain sources and deliver it to smart contracts, enabling dApps to interact with the external world in meaningful ways. For example, lending or access control applications may require age verification or proof of solvency derived from authenticated off-chain records before executing on-chain logic. Without secure oracles, systems rely on on-chain identities or trusted notaries, increasing trust assumptions and compliance risks.

Oracles can be realized through a variety of design paradigms, including voting-based mechanisms, dispute resolution frameworks, replication-based trust models, and cryptographic attestation schemes. Examples of non-cryptographic oracle designs include *Astraea* [1], which resolves factual claims through decentralized voting and incentive mechanisms, and systems such as *Decentagram* [57], which focus on data availability and dissemination rather than cryptographic attestation of authenticated web data.

Among the diverse landscape of oracle designs, *TLS-based oracles* have emerged as a particularly promising approach. These systems leverage the ubiquity and security guarantees of the Transport Layer Security (TLS) protocol [14, 39] to access authenticated data from web servers. By relying on the widespread deployment of HTTPS, TLS-based oracles provide a practical and secure bridge between traditional web in-

frastructure and blockchain systems.

However, the TLS protocol was not originally designed to produce verifiable transcripts. Naive attempts to circumvent this, such as third-party sharing of session keys, undermine privacy guarantees. Early solutions [9, 22, 40] sought to extract attestations from TLS sessions without revealing sensitive information, but these approaches typically required modifications to the TLS stack, limiting their adoption in practice.

Recent research has led to the development of protocols collectively referred to as *zkTLS* or *Designed Commitment TLS (DCTLS)* [11, 16, 34, 38, 56]. These protocols integrate TLS with cryptographic tools such as zero-knowledge proofs (ZKPs) and two-party computation (2PC) to ensure privacy while minimizing changes to existing TLS infrastructure. Typically, a DCTLS protocol involves three roles: the server, the prover, and the verifier. The prover and verifier jointly establish a TLS session using 2PC. Zero-knowledge proofs allow the prover to selectively disclose authenticated data to the verifier, who can then attest to its correctness without gaining access to the full session transcript.

DCTLS protocols aim to ensure both correctness and privacy-preserving in their attestations by providing two core security guarantees. First, unforgeability ensures that a prover cannot generate attestations for data that was never committed during the TLS session. Second, privacy guarantees that the verifier learns no more information than what the prover explicitly and selectively discloses through zero-knowledge proofs.

Despite these advances, a significant limitation remains in DCTLS protocols, namely the lack of public verifiability. Therefore, only a designated party can confirm correct execution, which limits the use of DCTLS proofs in smart contracts. In general, the designed verifier is effective, as it enables operations to be denied under specific circumstances in a controlled manner. However, in public verifiability settings, current DCTLS designs rely on a trusted verifier to participate in the session and forward attestations to the blockchain. This reliance introduces a potential single point of failure. In particular, a malicious prover and verifier may collude to fabricate attestations, giving rise to what is commonly referred to as the *collusion problem*. For example,

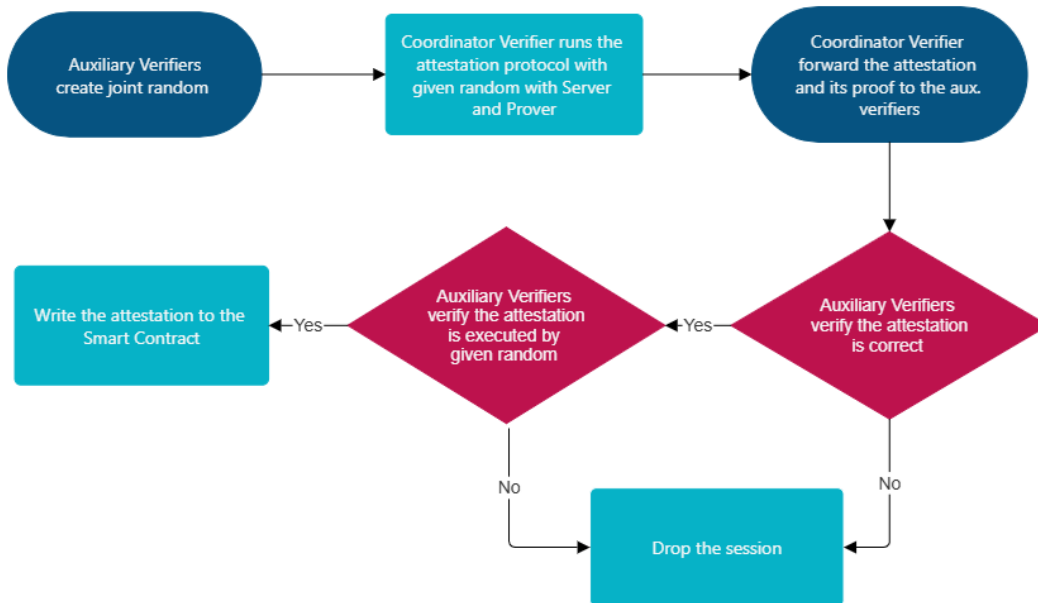


Figure 1.1: Overview of a Multi-Verifier Attestation Workflow Designed to Mitigate Prover-Verifier Collusion by Distributing Verification Across Auxiliary Verifiers

in a decentralized insurance application, a dishonest prover could bribe the verifier to falsely confirm a fabricated event, resulting in an illegitimate payout. ¹

Existing approaches to mitigating prover-verifier collusion in oracle systems can be broadly grouped into cryptographic, replication-based, and hardware-assisted designs. Cryptographic approaches, exemplified by TLSNotary [38], employ techniques such as blind signatures to prevent verifiers from linking attestations to specific protocol executions, thereby limiting collusion opportunities. However, these schemes require repeated blind-signature operations across multiple notaries, which introduces high computational cost, coordination overhead, and scalability limitations in decentralized settings, while also relying on protocol components that are not fully specified or formally analyzed. A second class of solutions reduces reliance on individual verifiers through replication, as seen in decentralized oracle networks such as Chainlink [8], Band Protocol [4], and API3 [5]. In TLS-based attestation settings, this model forces the prover to engage in independent protocol executions with each oracle instance, resulting in linear prover complexity and additional challenges when attestations are performed over time-sensitive or rapidly changing data. A third line of work relies

¹ Acknowledging this risk, most existing DCTLS protocols explicitly assume that the prover and verifier do not collude.

on Trusted Execution Environments (TEEs) [41, 27, 55, 57] to confine oracle logic within hardware-protected enclaves, with systems such as Town Crier [55] combining authenticated TLS communication with enclave-based execution and public verifiability. While effective in controlled deployments, TEE-based approaches depend on hardware-specific trust assumptions, are vulnerable to implementation-level attacks, and require sensitive data to be processed inside enclaves, unlike DCTLS constructions, where plaintext data remain exclusively with the prover. Consequently, although existing techniques reduce collusion risk through different mechanisms, they do so by trading off scalability, efficiency, openness, or privacy, highlighting the need for TLS-based oracle designs that provide cryptographic collusion resistance while preserving data confidentiality and supporting large verifier sets.

To overcome this limitation, this thesis proposes an efficient alternative that replaces the single designated verifier assumption in DCTLS protocols with a multi-verifier setting, as illustrated in Figure 1.1. In this setting, a set of auxiliary verifiers jointly generate randomness and delegate the interactive attestation protocol to a coordinator verifier. The coordinator executes the attestation with the server and the prover, then forwards the resulting attestation and proofs to the auxiliary verifiers. These verifiers independently validate correctness and collectively decide whether the attestation is accepted and submitted to the smart contract or the session is aborted. The central goal is to enable attestations that remain privacy-preserving while being jointly verifiable and suitable for consumption by smart contracts, without introducing excessive overhead or impractical trust assumptions.

1.1 Motivation

A fundamental vulnerability in existing TLS-based oracle designs stems from their reliance on a trusted verifier to attest and relay off-chain data to the blockchain. In current DCTLS protocols, verifier honesty is assumed by design, creating a single point of failure in adversarial or permissionless settings, where verifier identities are inexpensive, and collusion is economically rational. As a result, a malicious prover can collude with a verifier to produce fraudulent attestations that remain indistinguishable from honest executions, severely limiting the practical deployment of such

protocols.

Although prior work has explored alternative mitigation strategies against collusion, such as decentralized oracle networks, repeated attestations, trusted hardware, and notary-based approaches, these solutions tend to impose strong trust assumptions, increase prover-side complexity, or depend on components that are non-portable and difficult to audit. More fundamentally, existing approaches fail to address the structural limitation that DCTLS protocols tightly couple attestation execution with verification, thereby preventing independent third-party validation of correctness. In addition, although several DCTLS constructions have been proposed in the literature, they remain defined on a per-protocol basis, lacking a unified abstraction that captures their shared structure and security goals. This fragmentation requires separate protocol designs and security analyses for each construction, limiting generality and reuse.

This thesis is motivated by the absence of a cryptographically sound and efficient framework that minimizes prover-verifier collusion while preserving the privacy and deployability advantages of TLS-based attestations. A motivation is to unify existing DCTLS constructions under a common abstract formulation, enabling security guarantees to be defined and proven at the level of a general DCTLS abstraction rather than on a per-protocol basis. Our aim is to enable off-chain data to be incorporated into smart contracts with correctness guarantees, thereby enhancing the functionality of decentralized applications that depend on authenticated external information in adversarial environments.

Beyond cryptographic soundness, a key motivation of this thesis is to ensure that collusion-resistant TLS attestations remain practically deployable in real-world decentralized applications. In particular, we aim to demonstrate that minimizing trust and collusion does not come at the cost of prohibitive performance overheads or impractical system complexity. Finally, this thesis is also motivated by the need to demonstrate that the proposed framework can be applied in representative real-world scenarios, beyond its theoretical formulation.

1.2 Contribution

In this thesis, we make the following contributions to the design and analysis of collusion-resistant TLS attestation protocols.

First, we introduce a conceptual framework that integrates DVRFs into the DECO protocol under an optimistic verification model. In this design, the single designated verifier assumption is replaced with a distributed verifier model, where multiple verifiers jointly generate randomness and participate in the attestation process, thereby minimizing prover-verifier collusion risk. Commitments are generated and stored via decentralized storage by the participating parties, and the attestation is initially accepted under the assumption of honest behavior. In the presence of a dispute, each participant is required to open its commitment to enable dispute resolution. While this model provides a clear and intuitive security baseline, it inherently relies on trusted and tamper-proof storage to preserve commitments until disputes arise. As a result, despite its conceptual soundness, the optimistic model proves impractical for many real-world deployments, where such storage guarantees cannot be reliably enforced.

Second, we develop an improved and more practical architecture that eliminates the need for trusted storage and optimistic assumptions by integrating threshold signature schemes (TSS) directly into the protocol design. This compact and self-contained model achieves verifiability without relying on privileged intermediaries, minimizes prover-verifier collusion risk, and is particularly well-suited for resource-constrained decentralized applications. By avoiding heavy coordination layers and excessive on-chain overhead, the design enables efficient deployment in smart-contract-based environments. We further evaluate the on-chain verification gas costs and network communication requirements for three different threshold signature schemes. This comparative analysis enables selecting the most suitable signature construction depending on application-level requirements and the chosen threshold parameters. The results demonstrate that TSS-based attestations remain efficient and feasible for decentralized environments, even when deployed at scale. For example, while TLSS-BLS offers strong aggregation properties at the cost of higher on-chain gas consumption, FROST incurs substantially lower on-chain verification costs despite requiring higher network communication overhead.

Third, and most importantly, we introduce a generalized unique Designed Commitment TLS (DCTLS) construction from existing (DCTLS) constructions under a unified security model. We also demonstrate that standard DCTLS protocols are insufficient to achieve jointly verifiable and collusion-resistant attestations in multi-verifier settings. Building on this insight, we introduce designed exportable DCTLS (dx-DCTLS), an exportable and verifiable abstraction derived from DCTLS, which replaces non-verifiable two-party components with verifiable cryptographic counterparts. This abstraction enables third-party verification of TLS attestations and serves as a unifying layer over existing protocols such as DECO and Distefano, without requiring changes to the underlying TLS server infrastructure. On top of the dx-DCTLS abstraction, we construct a final collusion-minimized TLS attestation framework that integrates Distributed Verifiable Random Functions (DVRFs) and Threshold Signature Schemes (TSS).

As a foundation for this development, we also introduce a strawman protocol that generalizes any two-party computation DCTLS scheme to an n -party setting by extending its two-party components to full n -party computation. By instantiating and analyzing multiple concrete protocols, including extensions of DECO and Distefano, we show that dx-DCTLS-based constructions preserve the security guarantees of the strawman design while reducing prover complexity from $\mathcal{O}(n)$ to $\mathcal{O}(1)$ and eliminating the need for infeasible full n -party computation.

Finally, the practicality of the proposed framework is evaluated through a prototype implementation and a comprehensive performance analysis. We benchmark the DVRF-TSS layer under varying threshold sizes and report both computation time under LAN and WAN settings and network communication overhead. Our measurements indicate that the dominant overhead stems from the DKG phase; consequently, when a pre-existing DKG is reused, the protocol remains practical even under comparatively adverse WAN conditions, scaling up to threshold sizes of 50 in terms of both communication and computation time. Based on existing implementations, we then derive concrete overhead bounds for dx-DCTLS, showing that even under an accessible link bandwidth of 64 Mb/s, the resulting computational cost remains comparable to a standard DECO execution, thereby confirming the practical feasibility of our dx-DCTLS construction. Lastly, we present a comparative evaluation against the

closest related protocols. The results show that our framework achieves lower prover-side complexity while enabling higher scalability through the DVRF-TSS layer, supporting larger thresholds and minimizing collusion risk. Together, these findings confirm that the proposed protocols are secure and suitable for decentralized applications.

1.3 Organization

The structure of this thesis is outlined below.

Chapter 2 introduces the foundational concepts and cryptographic primitives used in the proposed protocols, including Distributed Verifiable Random Functions (DVRFs), zero-knowledge TLS (zkTLS), verifiable 2PC (v2PC), and collaborative succinct non-interactive arguments of knowledge (co-SNARKs).

Chapter 3 introduces the first baseline construction, in which collusion resistance is achieved by replacing the designated verifier assumption with a distributed verifier model based on Distributed Verifiable Random Functions (DVRFs). Rather than allowing a single verifier to generate randomness and initialize the protocol, a group of verifiers jointly produces randomness via a DVRF and assigns its output to a coordinator verifier, V_{coord} . This construction prevents any single verifier from controlling protocol initialization and mitigates prover-verifier collusion. The protocol further leverages decentralized storage to commit protocol state at each step, following an optimistic security model in which disputes are resolved by opening committed information.

Chapter 4 introduces a more compact variant of the baseline protocol by incorporating Threshold Signature Schemes (TSS), leveraging the same Distributed Key Generation (DKG) mechanism already required for Distributed Verifiable Random Functions (DVRFs). While the decentralized storage-based construction in Chapter 3 achieves collusion resistance, it is not well-suited for scenarios, where attestations must be consumed directly by smart contracts due to storage and verification overheads. Observing that DVRFs already require a DKG phase, this chapter addresses the question of how the same setup can be reused to obtain compact, on-chain verifiable attestations. To this end, Threshold Signature Schemes (TSS) are integrated to replace

decentralized storage with succinct threshold signatures, significantly reducing on-chain verification costs. In addition, this chapter presents a detailed analysis of gas costs for different TSS verification variants and demonstrates the practicality of the construction through representative real-world deployment scenarios.

Chapter 5 introduces dx-DCTLS, a design-exportable abstraction that unifies existing DCTLS constructions, including both DECO and Distefano for TLS 1.3. After formalizing DCTLS as a general construction, the chapter identifies why naive designs are incompatible with DVRF integration and derives the structural requirements for such integration. Based on these insights, dx-DCTLS is proposed and combined with the DVRF-TSS framework from earlier chapters to obtain the final protocol. The security of the resulting construction is established through game-based proofs under standard cryptographic assumptions, and its practicality is demonstrated through empirical evaluation in both LAN and WAN settings, where we report computation time and network communication costs for the DVRF-TSS layer and derive concrete upper bounds for the dx-DCTLS component.

Finally, Chapter 6 concludes the thesis with a synthesis of the main contributions and results. It summarizes the evolution of the proposed collusion-resistant TLS attestation framework, the formal security analysis, and the practical evaluation, and discusses limitations and avenues for future work.

CHAPTER 2

PRELIMINARIES

This chapter introduces the cryptographic preliminaries used throughout the thesis and clarifies how each component supports a collusion-minimized TLS attestation framework. We first review the structure of TLS 1.2 and TLS 1.3, emphasizing their key schedules and security guarantees, and explain why deriving publicly verifiable yet confidentiality-preserving evidence from standard TLS traffic is non-trivial. We then present the foundations of zero-knowledge proofs (ZKPs) and secure multiparty computation (MPC), with particular focus on two-party computation (2PC), which underpins DCTLS-style three-party handshake protocols. Building on these primitives, we describe Designated-Commitment TLS (DCTLS) protocols and summarize representative constructions such as DECO and Distefano, highlighting their roles, targeted TLS versions, and core guarantees such as unforgeability and privacy. Finally, we introduce Distributed Verifiable Random Functions (DVRFs) and Threshold Signature Schemes (TSS), both of which rely on distributed key generation (DKG). These mechanisms are used in later chapters to mitigate prover-verifier collusion and enable threshold agreement. They also support the construction of compact attestations suitable for on-chain verification. We conclude with additional preliminaries on verifiable 2PC and collaborative SNARKs, which provide further building blocks for public verifiability and multi-party proving in adversarial settings.

2.1 Transport Layer Security (TLS)

The *Transport Layer Security* (TLS) protocol [14, 39] is the standard cryptographic mechanism for securing communication over the Internet. TLS provides encrypted and authenticated channels, typically between clients and servers. These channels routinely carry sensitive and trust such as personally identifiable information, proof-of-age assertions, social security status, and purchase authorizations. While such information is critical to service delivery, its release beyond the immediate session context poses substantial privacy risks [19, 21]. Extracting verifiable yet privacy-preserving credentials from TLS-protected streams is cryptographically challenging, as the protocol guarantees confidentiality and endpoint authentication by design.

In response to increasing demands from both legislation, such as the GDPR [19] and standards bodies such as the W3C [21], there is a growing interest in enabling controlled, zero-knowledge-compatible data export from TLS sessions. This section provides an overview of the cryptographic structure of TLS 1.2 and TLS 1.3, with a focus on their primitives, security guarantees, and implications for privacy-preserving protocols such as DECO[56], and Distefano [11].

2.1.1 TLS 1.2: Configurable Authenticated Key Exchange

TLS 1.2, specified in RFC 5246 [14], is characterized by a highly flexible handshake protocol that negotiates key exchange algorithms, cipher suites, and MACs. The protocol supports both RSA-based key transport (deprecated) and ephemeral Diffie-Hellman key exchange (DHE or ECDHE). After key agreement, a shared *Pre-master secret* is transformed into a *Pre-master secret* using the TLS-PRF:

$$\text{PRF}(\textit{secret}, \textit{label}, \textit{seed}) = P_{\text{hash}}(\textit{secret}, \textit{label} \parallel \textit{seed}),$$

where P_{hash} is recursively defined as:

$$P_{\text{hash}}(\textit{secret}, \textit{seed}) = \text{HMAC}_{\textit{secret}}(A_1 \parallel \textit{seed}) \parallel \text{HMAC}_{\textit{secret}}(A_2 \parallel \textit{seed}) \parallel \dots$$
$$A_1 = \text{HMAC}_{\textit{secret}}(\textit{seed}), \quad A_i = \text{HMAC}_{\textit{secret}}(A_{i-1}).$$

TLS 1.2 allows both stream ciphers (e.g., RC4, now deprecated) and block ciphers in CBC mode (e.g., AES-CBC), with integrity protection provided by HMAC. However,

this flexibility comes at the cost of complexity and fragility: numerous attacks such as BEAST, Lucky13 [24], and POODLE [36] exploit weaknesses in cipher negotiation, padding, or MAC-then-encrypt constructions. Forward secrecy is not enforced by default and depends on the selected key exchange mechanism. RSA-based key transport lacks forward secrecy entirely, making past sessions vulnerable upon long-term key compromise.

2.1.2 TLS 1.3: Streamlined AKE with Forward Secrecy by Default

TLS 1.3, defined in RFC 8446 [39], is a cryptographic overhaul of its predecessor, offering reduced complexity and stronger security guarantees. In contrast to TLS 1.2, it enforces forward secrecy by default by supporting only ephemeral ECDHE key exchange, eliminates separate MAC constructions by mandating authenticated encryption with associated data (AEAD) ciphers such as AES-GCM and ChaCha20-Poly1305, and adopts a simplified key schedule based entirely on HKDF [33], enabling modular analysis and provable security in the random oracle model.

The handshake derives secrets through the following key schedule:

```
early_secret = HKDF-Extract(0, PSK)
handshake_secret = HKDF-Extract(early_secret, ECDHE)
master_secret = HKDF-Extract(handshake_secret, "derived")
client_app_key = HKDF-Expand(master_secret, "c ap traffic",
                             transcript_hash)
server_app_key = HKDF-Expand(master_secret, "s ap traffic",
                             transcript_hash)
```

TLS 1.3 binds the session transcript hash into every key derivation step, ensuring downgrade resilience and session uniqueness. Formal models such as ACCE (Authenticated and Confidential Channel Establishment), as shown in [25], have been used to analyze its security, showing it to be a robust, composable foundation for building higher-level privacy-preserving cryptographic protocols. Overall, TLS 1.3 represents a shift toward simplicity and rigor, with architectural features that make it significantly more compatible with modern cryptographic extensions and formal proofs.

2.2 Zero-Knowledge Proofs (ZKP)

Zero-Knowledge Proofs (ZKPs) are interactive protocols that take two inputs, (x, w) , where x is the public statement and w is the private witness, such that $(x, w) \in \mathcal{R}$ for some NP relation \mathcal{R} . The goal is to generate a proof that convinces a verifier of the truth of x without revealing w . These interactive protocols can typically be transformed into non-interactive zero-knowledge proofs (NIZK) using the Fiat-Shamir heuristic [17], which replaces interaction with a cryptographic hash function in the random oracle model.

Definition 1. (*Zero-Knowledge Proofs (ZKP)*). *The ZKP scheme consists of three algorithms, denoted as $ZKP = (\text{Setup}, \text{Prove}, \text{Verify})$, given by*

- $(pk, vk) \leftarrow \text{ZKP.Setup}(1^\lambda, \mathcal{R})$: The Setup function takes 1^λ as a security parameter, outputs proving and verification keys, and \mathcal{R} is a binary relation for (x, w) pair.
- $\pi \leftarrow \text{ZKP.Prove}(pk, x, w)$: Prover uses proving key pk , statement x and witness w , where $\mathcal{R}(x, w) = 1$.
- $1/0 \leftarrow \text{ZKP.Verify}(vk, \pi, x)$: Verifier verifies the proof with verification key without witness w .

The ZKP scheme satisfies the soundness, completeness, and zero-knowledge properties.

Definition 2 (Completeness). *A zero-knowledge proof scheme ZKP satisfies completeness if, for all valid statement-witness pairs (x, w) such that $\mathcal{R}(x, w) = 1$, an honest prover can always convince an honest verifier to accept the proof.*

$$\Pr \left[\begin{array}{l} (pk, vk) \leftarrow \text{ZKP.Setup}(1^\lambda) \\ \pi \leftarrow \text{ZKP.Prove}(pk, x, w) \\ \text{ZKP.Verify}(vk, \pi, x) = 1 \end{array} \right] = 1$$

Definition 3 (Soundness). *A zero-knowledge proof scheme ZKP satisfies soundness if for any probabilistic polynomial-time (PPT) adversary \mathcal{A} that produces an accepting proof for a statement x , there exists a PPT extractor $\mathcal{X}_{\mathcal{A}}$ that can extract a witness w such that $\mathcal{R}(x, w) = 1$, except with negligible probability.*

$$\Pr \left[\begin{array}{l} (pk, vk) \leftarrow \text{ZKP.Setup}(1^\lambda) \\ (\pi, x) \leftarrow \mathcal{X}_{\mathcal{A}}(pk, x, w) \\ \text{ZKP.Verify}(vk, \pi, x) = 1 \\ \wedge \mathcal{R}(x, w) = 0. \end{array} \right] \leq \text{negl}(\lambda)$$

Definition 4 (Zero-Knowledge). *A zero-knowledge proof scheme ZKP satisfies zero-knowledge if for any probabilistic polynomial-time (PPT) adversary \mathcal{A} , there exists a PPT simulator $\mathcal{S}_{\mathcal{A}}$ such that the adversary cannot distinguish between a real proof and a simulated proof generated without access to the witness, except with negligible probability.*

$$\Pr \left[\begin{array}{l} (pk, vk) \leftarrow \text{ZKP.Setup}(1^\lambda) \\ x \leftarrow \mathcal{A}(pk), \exists w : \mathcal{R}(x, w) = 1 \\ \pi^* \leftarrow \mathcal{S}_{\mathcal{A}}(pk, x) \\ \text{ZKP.Verify}(vk, \pi^*, x) = 1 \end{array} \right] \leq 1 - \text{negl}(\lambda)$$

2.3 Secure Multiparty Computation (MPC)

Secure Multiparty Computation (MPC) is a cryptographic primitive that enables a set of mutually distrustful parties to jointly compute a function over their private inputs without revealing anything beyond the output. The classical goal is to simulate the ideal functionality, where a trusted third party receives all inputs, computes the function, and returns the output, without any actual trusted party being involved.

Definition 5. (*Secure Multi-Party Computation (MPC) Realization*). *Consider a function $f(x_1, x_2, \dots, x_n)$ that is jointly evaluated by n parties, where each party P_i holds a private input x_i . A protocol Π is a secure realization of f if it satisfies the following security features even in the presence of adversarial parties: correctness, privacy, and indistinguishability.*

Secure MPC protocols can be instantiated under various threat models, including semi-honest (passive) and malicious (active) adversaries, and the achievable guarantees vary accordingly.

2.3.1 Two-Party Computation (2PC)

A special case of MPC is *two-party computation (2PC)*, where only two parties P_1 and P_2 wish to compute a joint function $f(x_1, x_2)$ over their private inputs. This setting is fundamental both for its practical applications and its theoretical importance in establishing general feasibility results for MPC.

Several cryptographic techniques have been developed for 2PC, including:

- **Yao's Garbled Circuits** [53, 54]: One party encodes a Boolean circuit in a "garbled" form and the other evaluates it obliviously using oblivious transfer (OT).
- **Secret Sharing-Based Protocols** [20, 31]: Parties share their inputs using additive or Shamir-style secret sharing and perform computation on shares using linear operations and interaction for nonlinear gates.
- **Homomorphic Encryption-Based Methods** [13]: Inputs are encrypted under homomorphic encryption schemes, and computation is performed directly on ciphertexts.

2.3.2 Security Definitions of MPC

An MPC protocol satisfies the following security properties.

- **Correctness**: Honest parties receive the correct output $f(x_1, x_2, \dots, x_n)$.
- **Input Privacy**: No party learns anything about other parties' inputs, beyond what is revealed by the output.
- **Independence of Inputs**: Parties choose their inputs independently of one another.
- **Fairness (Optional)**: All parties receive the output, or none do. This is hard to achieve in 2PC without trusted setup.

- **Guaranteed Output Delivery:** Honest parties eventually receive the output even if other parties abort (achievable in some settings).
- **Robustness and Verifiability:** The protocol ensures output integrity even in the presence of actively malicious adversaries.

Secure multi-party computations have emerged as a central primitive in privacy-preserving cryptographic systems, enabling applications such as private bidding, secure voting, privacy-preserving machine learning, and more. In practice, protocol efficiency often depends on assumptions about the adversarial model (semi-honest vs. malicious), the number of parties, and the type of network (synchronous vs. asynchronous).

2.4 Designated-Commitment TLS (DCTLS) Protocols

Designated-Commitment TLS (DCTLS) protocols are a class of MPC-based zero-knowledge TLS protocols, where the *prover* and the *verifier* collaboratively act as a single logical TLS client to attest from the *server*. The literature contains a broad range of concrete instantiations of this approach. For example, DECO [56] provides one of the early privacy-preserving TLS 1.2 attestation constructions, TLSNotary [38] highlights practical implementations and deployments, and a TEE-based design [55] enables publicly verifiable attestations in trusted execution environments. Beyond these representative examples, several other constructions and systems explore related design points, including alternative proof techniques, protocol optimizations, and deployment-oriented variants [11, 16, 34, 52]. The main purpose of these protocols is to enable exporting certain verifiable claims from a secure TLS session to a designated verifier, without compromising the confidentiality or integrity of the client's private data.

DCTLS protocols achieve this functionality by modifying the standard TLS handshake and record-layer processing. Specifically, they perform the TLS handshake through secure two-party computation (2PC), where the client and the verifier hold secret shares of the session's ephemeral state (e.g., random nonces, premaster secrets,

keys). Once the handshake is complete, subsequent TLS records (application data) are also processed using shared state and 2PC protocols.

We present a generic DCTLS construction in Chapter 5, which is instantiated to capture both DECO [56] and Distefano [11] as concrete realizations of the DCTLS paradigm. While these two systems serve as representative and well-studied examples, the proposed framework is not limited to them. We argue that other DCTLS protocols appearing in the literature can also be expressed within this construction by appropriately instantiating the underlying MPC, commitment, and verification components. In this sense, our formulation provides a unifying abstraction that accommodates existing designs while remaining flexible enough to incorporate future DCTLS proposals.

2.4.1 DECO Protocol

DECO [56] is an oracle protocol consisting of three key roles: Prover P , who is the data owner, the website server S who stores the prover’s data and the verifier V also known as Oracle, tasked with validating the accuracy of the data on server S by leveraging the integrity guarantees provided by TLS. The protocol ensures privacy by enabling the verifier, functioning as the oracle, to confirm the data’s origin on the server without accessing the full details of the query. This is achieved through the application of a zero-knowledge protocol (ZKP). In the settings of the CBC-HMAC cipher suite for TLS 1.2, the DECO scheme denoted as $\text{DECO} = (\text{Handshake Phase}, \text{Query Phase}, \text{Proof Generation Phase})$ algorithms:

Handshake Phase (HSP): Handshake phase is three-party handshake that the verifier V and the prover P collaborate as a single TLS client. This phase starts with the prover P by choosing a random r_c , then sends it to the server in the *ClientHello* message. Server responds by picking random (r_s, s_s) , then sends $y_s = s_s * G$ and Certificate to the prover P in *ServerHello* message. Prover P validates the Certificate and forwards y_s and the certificate to the verifier V . The verifier V checks the *Certificate* and pick a random number s_v and replies back to the prover P with $y_v = s_v * G$. Prover P picks s_p randomly and calculates y_p as $y_p = s_p * G + y_v$ then forwards it to the server S . Now, the server S can calculate premaster secret Z as $Z = s_s * y_p$ while

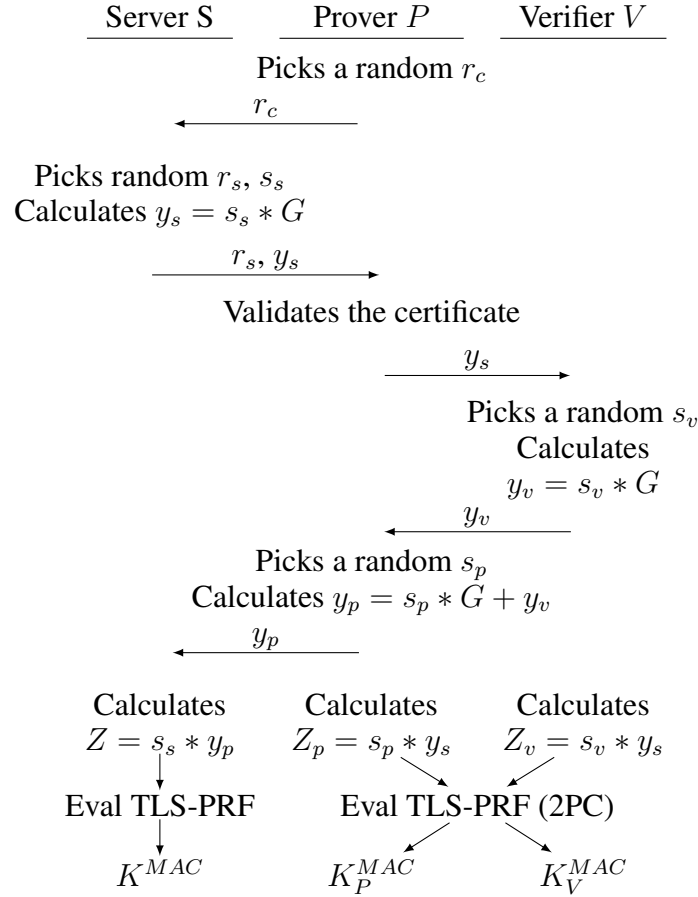


Figure 2.1: An Overview of the DECO [56] Three-Party Handshake

the prover P and verifier V can calculate their additive share of premaster secret Z as Z_p and Z_v , where $Z_p = s_p * y_s$ and $Z_v = s_v * y_s$. The phase is ended with, the server S obtains K^{MAC} and K^{ENC} while prover P obtains K_P^{MAC} and K^{ENC} and verifier V obtains only K^{MAC} by the help of 2PC. Now, the peers are ready for the next step, named Query Phase (QP). DECO HSP phase is shown in Figure 2.1.

Query Phase (QP): The query phase is executed through a sequence of protocol steps. The prover P creates a query Q with statement b that verifier shows and encrypt it by using K^{ENC} namely \hat{Q} . However, P cannot create the authentication tag τ because lacks of the full MAC key, or other share of K_V^{MAC} . To achieve this, P and V collaborate through a 2-party computation (2PC) to jointly generate the tag τ . P appends the τ to the Q , encrypted $(Q||\tau)$ to \hat{Q} and send it to the server S , then receives encrypted response \hat{R} . The prover P can decrypt and read the \hat{R} , but it cannot verify the authenticity of the response. Without validating authenticity, P commits (\hat{Q}, \hat{R}) with K_P^{MAC} to V , and V replies with K_V^{MAC} . Finally, P reconstructs the full MAC

key K^{MAC} , enabling it to authenticate the server's response.

Proof Generation Phase (PGP): In proof generation phase, the prover P constructs a zero-knowledge proof (ZKP) called as DECO Proof to demonstrate that the private values $x = (Q, R, K^{ENC}, \theta_s)$ were derived correctly from the public values $w = (\hat{Q}, \hat{R}, K^{MAC}, b)$. Here, θ_s represents the sensitive information being queried, which must remain confidential, and b denotes the statement to be proven. Additionally, (Q, R) represent the plaintext query and response, while (\hat{Q}, \hat{R}) are their corresponding ciphertexts.

After receiving the ZKP proof, the verifier V verifies the proof with respect to the public inputs and checks that $K^{MAC} = K_V^{MAC} + K_P^{MAC}$; by the completeness property of the zero-knowledge proof system as defined in Definition 2, the verifier concludes that the attestation is valid.

DECO ensures the integrity of both the prover and verifier, meaning that attestation cannot proceed if either party acts maliciously. In particular, DECO is secure against forgery of false proofs.

2.4.2 Distefano Protocol

Similar to other DCTLS protocols such as DECO, Distefano [11] consists of three roles Prover P who is the data owner, the website server S who stores the prover's data and the verifier V also known as Oracle, tasked with validating the accuracy of the data on server S by leveraging the integrity guarantees provided by TLS. In the settings of the AES-GCM cipher suite for TLS 1.3, the Distefano scheme, denoted as Distefano = (Handshake Phase, Query Phase, Commit Phase) algorithms:

Handshake Phase (HSP): Three-party handshake protocol that takes security input 1^λ and returns the TLS 1.3 handshake key and its shares among Server S , Prover P , and Verifier V , where the original client is shared into two components, prover and verifier. Prover P and verifier V pick two random numbers, respectively s_v and s_p then both exchange Z_v and Z_p , where $Z_v = s_v * G$ and $Z_p = s_p * G$. Both obtain ephemeral key share (SSK), where $SSK = Z_p + Z_v$. The prover sends the SSK to the server S in the ClientHello message, then server S replies with the ServerHello

message including ServerKeyShare (SKS), namely Y_s , where $Y_s = y_s * G$ calculated by picking a random y_s . Prover P forwards the SKS to the verifier V , thus the verifier calculates its share of $SSK_v = Y_s * s_v$, and the prover calculates its share of $SSK_p = Y_s * s_p$. The rest of the key derivation processes of TLS 1.3 are done by multiple 2-party computations (2PCs) to simulate a single client. Finally, prover P and verifier V obtain (s_p, s_v) . The complete set of keys is given by (pp, sp_p, sp_s, sp_v) , where pp denotes the public parameter, and sp_p , sp_s , and sp_v denote the secret keys obtained by the prover, the server, and the verifier, respectively.

Query Phase (QP): In this phase, again prover P and verifier V creates a valid TLS query by using key parameters (pp, sp_p, sp_s, sp_v) created in previous phase. The prover and verifier act as a single TLS 1.3 client by using the 2PC and creating a query Q . The server responds to this query with a response R to the Prover. Also the query phase outputs the encrypted query and response (\hat{Q}, \hat{R}) respectively. The prover P gets the encrypted response from server S , but cannot decrypt it since its key share is not enough to decrypt it.

Commit Phase (CP): In the commitment phase, the prover commits to the encrypted query and response ciphertexts (\hat{Q}, \hat{R}) using a commitment scheme $\Gamma = (\mathbf{Commit}, \mathbf{Challenge}, \mathbf{Open})$. The scheme provides perfectly hiding and computationally binding security guarantees. Upon receiving the commitment, the verifier releases its key share sp_v , enabling the prover to decrypt the response. The prover then opens the commitment to the verifier, selectively revealing the committed values by partially redacting them over an index range (i, j) . After the verifier receives this ZKP, the verifier outputs 1 if the verification is done; otherwise, the verifier outputs 0.

Although the Distefano protocol provides *session privacy* and *authentication security* features too, we are interested in unforgeability as follows: Distefano ensures the *prover's integrity*, meaning attestation cannot be accomplished if either party acts maliciously.

2.5 Distributed Verifiable Functions (DVRF)

A Distributed Verifiable Random Function (DVRF) is a type of MPC, where all participants collaborate to generate a random number that can also be independently validated by others afterward with examples [2, 15, 18]. Generally, a Distributed Verifiable Random Function (DVRF) functions in a t -out-of- n threshold model, where any t participants from a total of n can jointly generate and validate the random number. The DVRF, denoted as **DVRF** = (**DKG**, **PartialEval**, **Combine**, **Verify**) is typically composed of the following algorithms:

- **DKG**($1^\lambda, t, n$) takes as input a security parameter 1^λ , the total number of participants n , and the threshold parameter t then outputs a public key pk a set of qualified nodes named QUAL, a partial verification key list $VK = \{vk_1, vk_2, \dots, vk_n\}$ for the nodes, and a partial secret key list $SK = \{sk_1, sk_2, \dots, sk_n\}$.
- **PartialEval**(α, sk_i, vk_i) is the partial evaluation algorithm that takes as inputs a plaintext α and the partial secret and verification keys (sk_i, vk_i) for a node to generate a partial evaluation γ_i along with a corresponding non-interactive zero-knowledge (NIZK) proof π_i . Specifically, $\gamma_i = (i, v_i, \pi_i)$, where v_i represents the i -th evaluation share.
- **Combine**(pk, VK, α, E) is the combination algorithm. It takes as inputs the global public key pk , the set of verification keys VK , a plaintext $\alpha \in \text{Dom}$, and a set $E = \{\gamma_{i_1}^\alpha, \dots, \gamma_{i_{|E|}}^\alpha\}$, consisting of partial function evaluations from at least $|E| \geq t$ distinct nodes. The algorithm outputs either a pair (v, π) , where $v \in \text{Rand}$ is the pseudorandom function value and π is its correctness proof.
- **Verify**(pk, VK, α, v, π) is the verification algorithm. It takes as inputs the public key pk , a set of verification keys VK , a plaintext α , a value $v \in \text{Rand}$, and a proof π . The algorithm outputs either 1, indicating that the proof is valid, or 0 otherwise.

A DVRF is required to satisfy consistency, robustness, and uniqueness.

- **Consistency:** All honest verifiers who participate in the DVRF evaluation on

the same input will derive the same output. This ensures agreement among distributed parties and prevents conflicting values for a given input.

- **Robustness:** The system tolerates a bounded number of faulty or malicious verifiers. As long as a threshold number of honest parties participate, the final output remains correct and verifiable.
- **Uniqueness:** For any given input and public key, there exists exactly one valid output that can be generated. This prevents equivocation and ensures that no adversary can produce multiple valid outputs for the same input.

2.6 Threshold Signature Schemes (TSS)

The threshold signature scheme provides a cryptographic mechanism that enables a group of n participants to jointly generate a digital signature such that any subset of size at least t can cooperate to produce a valid signature, while fewer than t participants cannot. This t -out-of- n signing capability ensures both fault tolerance and distributed trust, which are essential in decentralized systems [7, 32, 42].

Formally, a threshold signature scheme consists of the following algorithms: **TSS** = (**DKG**, **Sign**, **Verify**).

- **DKG**($1^\lambda, t, n$): Given a security parameter 1^λ , a threshold value t , and the total number of participants n , the algorithm outputs a public key pk and a set of secret key shares $SK = \{sk_1, sk_2, \dots, sk_n\}$, where sk_i is privately held by participant i . This step is ideally performed in a distributed manner to avoid a single point of trust.
- **Sign**(α, sk_i): Each participant i uses their secret share sk_i to generate a partial signature σ_i on the message α . Once at least t partial signatures are collected, they can be combined (often via Lagrange interpolation) to produce a unique and compact threshold signature σ .
- **Verify**(pk, α, σ): Given the public key pk , the message α , and the aggregated signature σ , it returns 1 if the signature is valid, and 0 otherwise.

Threshold signature schemes are widely used in distributed cryptographic systems such as multi-party consensus, decentralized identity management, and blockchain oracle protocols. Their non-interactive verification and robustness to faulty participants make them highly suitable for secure decentralized deployments.

2.7 Verifiable 2PCs (v2PCs)

The publicly verifiable two-party computation scheme is inspired by the Publicly Verifiable Covert (PVC) model [58]. It enables two mutually distrusting parties to jointly compute a deterministic function while ensuring that any deviation from the protocol can be detected and externally verified, without revealing the honest party’s private input. This construction is particularly relevant in adversarial or economically incentivized environments, where rational adversaries may be deterred by public accountability and auditable punishment mechanisms. v2PCs denoted as **v2PC** = (**Execute**, **Verify**):

- **Execute**(a_{in}, b_{in}) takes the inputs a_{in} and b_{in} held by parties A and B , respectively, and computes the deterministic function $f(a_{in}, b_{in})$. It outputs a_{out} to A and b_{out} to B , together with a proof π . The proof π is generated only in the event of suspected misbehavior or upon explicit invocation and serves as a public certificate of the computation’s correctness. Importantly, it enables a third party to verify the integrity of the protocol execution without requiring access to both private inputs, thereby preserving data confidentiality.
- **Verify**(π, a_{in}, a_{out}) takes as input a proof π together with an input-output pair (a_{in}, a_{out}) (or, analogously, (b_{in}, b_{out})) and checks their consistency under the function f . The verification algorithm outputs 1 if the proof π validates the correctness of the computation with respect to the provided input-output pair, and outputs 0 otherwise. This formulation preserves the full functionality of two-party computation while enabling third-party verification of correct execution using only one party’s input and the corresponding output, without revealing the other party’s private input.

Such verifiability is particularly valuable in real-world scenarios involving rational

adversaries. For example, if one party attempts to cheat, the honest party can publish π alongside their input-output pair to trigger a publicly auditable dispute resolution process. In on-chain deployments, this may involve invoking a smart contract that automatically verifies π and, upon detection of misbehavior, enforces penalties such as forfeiture of a security deposit. This framework makes the protocol practical for applications like secure auctions, decentralized gaming, and fair data exchange, where both integrity and privacy must be preserved under adversarial incentives.

2.8 Collaborative SNARKs (Co-SNARKs)

Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARKs) are among the most widely used zero-knowledge proof systems, allowing a prover to demonstrate the validity of a computation without disclosing the underlying witness. However, a notable limitation of traditional zk-SNARKs is that a single prover must hold the entire witness. To address this constraint, recent proposals [35, 37] introduces collaborative zk-SNARKs, in which multiple provers jointly participate in the proving process while preserving the confidentiality of their witness shares.

A co-SNARK scheme consists of the following algorithms: $\text{co-SNARK} = (\text{Execute}, \text{Verify})$.

- **Execute**($x, \{w_i\}_{i=1}^n$): Given a public statement x and a set of private witnesses $\{w_i\}_{i=1}^n$ held by n collaborating provers, this algorithm performs a joint computation to evaluate the statement. It outputs a public result y together with a succinct proof π attesting to the correctness of the computation. Each witness w_i is provided by a distinct party, and the proof π certifies the correct computation of y from x and the joint witnesses.
- **Verify**(π, x, y): For a given proof π , the public statement x , and the claimed output y . It outputs 1 if the proof verifies that y was correctly computed with respect to x and some valid hidden witnesses $\{w_i\}$, and outputs 0 otherwise. This verification procedure is identical to the **Verify** algorithm of the zero-knowledge proof defined in Definition 1.

While a co-SNARK scheme satisfies the security properties of *completeness*, *sound-*

ness, and *t-zero knowledge*, each property plays a distinct role in the collaborative setting. *Completeness* ensures that if all participating provers follow the protocol honestly and hold valid witness shares, the jointly generated proof will be accepted by the verifier. *Soundness* guarantees that no coalition of malicious provers can convince the verifier of an incorrect statement, except with negligible probability. *t-zero knowledge* further ensures that even if up to t provers collude, no additional information about the honest parties' witness shares is revealed beyond what is implied by the public output. Although all three properties are essential for the security of collaborative proving systems, our analysis in this thesis focuses primarily on the *soundness* property.

CHAPTER 3

OPTIMISTIC MULTI-VERIFIERS DECO WITH DECENTRALIZED STORAGE

This chapter introduces the first efficient protocol construction that addresses the oracle collusion problem in the context of using DECO-based TLS attestations within smart contracts. Building on the cryptographic foundations presented in Chapter 2, we consider a setting in which the designated verifier assumption of DECO [56] is relaxed by distributing the verifier role across multiple auxiliary parties. To the best of our knowledge, this chapter presents the first application of Distributed Verifiable Random Functions (DVRFs) in the context of DCTLs protocols to coordinate verifier-side randomness and prevent unilateral control of session initialization. The proposed construction follows an optimistic design, where honest executions proceed without additional overhead, and relies on decentralized storage to enable dispute resolution only in the presence of misbehavior. Overall, the construction demonstrates how collusion resistance can be achieved without increasing prover-side complexity, while preserving the privacy and correctness guarantees of the underlying TLS session.

The primary motivation of this chapter is to mitigate prover-verifier collusion in DECO protocol without introducing trusted hardware, blind signatures, or excessive prover-side overhead. Since DECO relies on a designated verifier, this role must be examined more carefully. In particular, we observe that the designated verifier effectively controls the randomness used to initialize the protocol, which creates a natural point of designed feature then consequently collusion. This observation motivates the question of how the protocol would behave if, instead of a single verifier,

the initialization randomness were jointly generated by a set of parties. Our answer to this question is to employ a Distributed Verifiable Random Function (DVRF) to coordinate verifier-side randomness in a decentralized manner. Distributing the verifier role, however, raises a further question regarding how disagreements or misbehavior should be handled once the protocol is executed. In the absence of a single trusted verifier, dispute resolution becomes a central design challenge. This chapter addresses this issue by adopting an optimistic protocol design and relying on decentralized storage to resolve disputes only when misbehavior occurs. Overall, this chapter is motivated by whether collusion resistance can be achieved through verifier decentralization while keeping the prover logic unchanged and maintaining practical efficiency.

To this aim, we present an optimistic multi-verifier extension of DECO that replaces the designated verifier with a distributed verifier model based on Distributed Verifiable Random Functions (DVRFs). We begin by analyzing the DECO handshake phase and identifying the points, where verifier-controlled randomness influences protocol initialization. Based on this analysis, we introduce additional commitment steps that enable the integration of DVRF-generated randomness into the handshake without modifying the prover logic. The protocol ensures that no single verifier can control session initialization or randomness generation, thereby preventing unilateral manipulation and reducing the risk of prover-verifier collusion. Disputes are resolved through an optimistic mechanism that relies on publicly verifiable commitments stored in decentralized storage. We further provide a detailed security discussion of the proposed construction and analyze the storage requirements incurred by decentralized dispute resolution. Overall, the construction preserves $O(1)$ prover complexity while achieving collusion resistance that scales with the number of verifiers. The protocol and its analysis were presented at the International Conference on Information Security (ISC Turkey) 2024 [45].

The remainder of this chapter is structured into the sections described below. Section 3.1 introduces the cryptographic and system-level building blocks required for the proposed construction, including the optimistic verification model, joint randomness generation via DVRFs, and decentralized storage assumptions. Section 3.2 describes the protocol in detail, specifying the participating entities and the step-by-step

construction. Finally, Section 3.3 provides a comprehensive analysis of the protocol, discussing both its security guarantees under collusion and its efficiency in terms of communication, computation, and storage overhead.

3.1 Building Blocks

This section introduces the additional foundational components that will be used in the design of the proposed protocol. Specifically, it covers the decentralized storage infrastructure, which ensures integrity and availability without relying on a centralized party, and the optimistic execution model, which assumes honest behavior by default but provides mechanisms for dispute resolution in the presence of misbehavior. These components form the basis for constructing an applicable system architecture in the following sections.

3.1.1 Optimistic Approach

The optimistic approach implemented in this protocol operates under the assumption that all participating parties behave honestly during the execution phase. This assumption allows the system to avoid unnecessary computational and communication overhead as long as no disputes arise. Each party records a trace of their execution, including relevant inputs, intermediate states, and outputs. Rather than revealing this trace immediately, each party generates a cryptographic commitment to the trace and publishes this commitment to a decentralized storage system. The storage system must be tamper-resistant and publicly accessible, ensuring that once a commitment is made, it cannot be altered or removed by any party.

Decentralized storage systems, such as distributed ledgers or content-addressed networks, serve to anchor these commitments in a manner that is verifiable by all participants. At this stage, privacy is preserved because the actual execution traces are not revealed. Only their committed representations are visible, relying on the binding and collision-resistant properties of the underlying commitment scheme.

If one of the parties detects potential misbehavior or inconsistencies in the outputs, the

protocol enters a dispute phase. During this phase, each involved party must reveal the corresponding portion of their trace that was previously committed. The revealed data is then verified against the original commitment stored in the decentralized storage. Verification ensures that the opening matches the commitment without exposing unrelated private information.

An external verifier, such as a smart contract or a decentralized adjudication mechanism, can independently check the validity of these openings. The ability to verify disputes without relying on a trusted third party strengthens the protocol’s robustness in adversarial environments. By combining efficient normal-case execution with strong guarantees in the presence of disputes, the optimistic approach provides a scalable, secure, and auditable mechanism for decentralized computation.

3.1.2 Joint Randomness

Designated verifiers limit verifiability to a specific party, which makes them effective against coercion and blackmail. However, this selectivity inherently prevents public verifiability. Since only the designated verifier can assess protocol compliance, external observers cannot detect deviations, enabling undetectable collusion.

In standard constructions, becoming a designated verifier begins with controlling a piece of randomness that initializes the protocol. This raises a structural question: who generates and holds the initial randomness?

Prior work, including [26], distributes this role among multiple designated verifiers to reduce the risk of collusion. However, these methods follow an n -out-of- n model: the absence of even a single auxiliary verifier invalidates the protocol.

This work proposes replacing the strict unanimity requirement with threshold-based randomness generation using distributed verifiable random functions (DVRFs). Unlike prior models, DVRFs allow t -out-of- n generation and verification of randomness, enabling the protocol to remain verifiable even in the presence of partial participation or faults.

3.1.3 Decentralized Storage

In order to support the optimistic execution model, the protocol requires a tamper-resistant and publicly accessible storage layer, where commitments can be reliably published and retrieved. This role is fulfilled by decentralized storage systems, which eliminate reliance on centralized servers and ensure that data remains verifiable, persistent, and censorship-resistant.

Examples of such decentralized storage networks include IPFS [6], Codex [12], and Arweave [51]. These systems are built on content-addressable architectures, where data is identified by its cryptographic hash rather than by its physical location. As a result, any modification to the committed data would lead to a different address, making tampering immediately detectable. This immutability property is essential for guaranteeing that the published commitments remain valid and verifiable over time.

In the context of this protocol, each party generates a cryptographic commitment to their local execution trace and stores it in the decentralized storage network. The commitments are made publicly accessible to ensure that any third party, including verifiers or smart contracts, can retrieve and validate them as needed. Because cryptographic commitments are computationally hiding, publishing them poses no risk to the privacy of the underlying data. The hiding property ensures that no information about the original input or computation trace can be inferred from the commitment alone.

By leveraging decentralized storage, the protocol guarantees that all commitments are recorded in a verifiable and permanent manner. This public availability enables transparent and trustless dispute resolution in the event of protocol deviation. It also aligns with the decentralized philosophy of the broader system, removing reliance on any single point of failure or trusted intermediary.

In the context of the protocol, we assume access to an abstract decentralized storage interface that provides two fundamental operations:

- **Put:** This operation stores a data object in the network. Given an input data \in

$\{0, 1\}^*$, the interface computes a content identifier as a cryptographic hash function H and returns: $\text{cid} \leftarrow \mathbf{Put}(\text{data})$, where $\text{cid} = H(\text{data})$ uniquely identifies the data and guarantees content addressability and immutability.

- **Get:** This operation retrieves a data object from the network based on its content identifier. Given cid , the interface returns: $\text{data} \leftarrow \mathbf{Get}(\text{cid})$. Correctness ensures that if cid was previously generated by a valid **Put** operation, then the output will be the original data object. If the data is unavailable or tampered with, retrieval fails or the verification fails.

These systems provide persistent, verifiable, and censorship-resistant storage, which is critical for ensuring that protocol commitments remain publicly accessible and tamper-proof throughout the dispute resolution process.

3.2 The Proposed Protocol

The proposed protocol combines DECO, DVRF, and decentralized storage within an optimistic execution model for mitigating collusion risk. This section presents the construction of the protocol and describes its main phases, including the DVRF, the DECO attestation, and the verification.

3.2.1 Entities

The proposed protocol involves the following entities, each with a specific role in the system:

- **Server S:** The TLS server S holds the data owned by the prover P and serves this data over a standard TLS connection in response to valid queries. The server functions identically to the server in the DECO protocol, regardless of whether the data is public or private.
- **Prover P:** The prover P that initiates a TLS session with S and seeks to attest to a statement b derived from the server response.

- **Coordinator Verifier** V_{coord} : This verifier initiates the verification request and is the entity that must be convinced about the origin and correctness of the data. Unlike in the original DECO model, where the verifier must be fully trusted, the proposed protocol is designed so that a malicious V_{coord} cannot collude with the prover to produce a false proof. In this setting, V_{coord} is a transparent verifier under the assumption that the majority of auxiliary verifiers are honest.
- **Auxiliary Verifiers** $\{V_i\}$: These are additional verifiers responsible for generating joint randomness via a DVRF protocol and for assisting in the verification process. Their coordination with V_{coord} strengthens the trust model by decentralizing the verification responsibility. With the V_{coord} , the set of verifiers is as follows: $V = \{V_{\text{coord}}, V_2, \dots, V_n\}$, where $|V| = n$
- **A smart contract**: It is an immutable execution platform called SC that consumes attestations on-chain by verifying threshold approvals and enforcing application-specific logic.
- **Decentralized Storage**: This is an immutable and publicly accessible storage layer that all parties can interact with. Both the prover P and verifiers are able to publish and retrieve commitments from this storage, ensuring tamper-resistance and transparency in the protocol.

3.2.2 Construction

The proposed protocol is structured according to the following architectural framework, as shown in Figure 3.1:

1. The verifier set $V = \{V_{\text{coord}}, V_2, \dots, V_n\}$ comes together to create a joint verifiable randomness by starting Distributed Key Generation (DKG) as the first step of DVRF.
2. After running DKG, each verifier V_i has its own secret key sk_i and runs the Eval function that creates the randomness s_v obtained by V_{coord} .
3. Server S , Prover P , and V_{coord} execute the DECO, specifically, V_{coord} uses s_v unlike the original DECO, where it picks randomly.

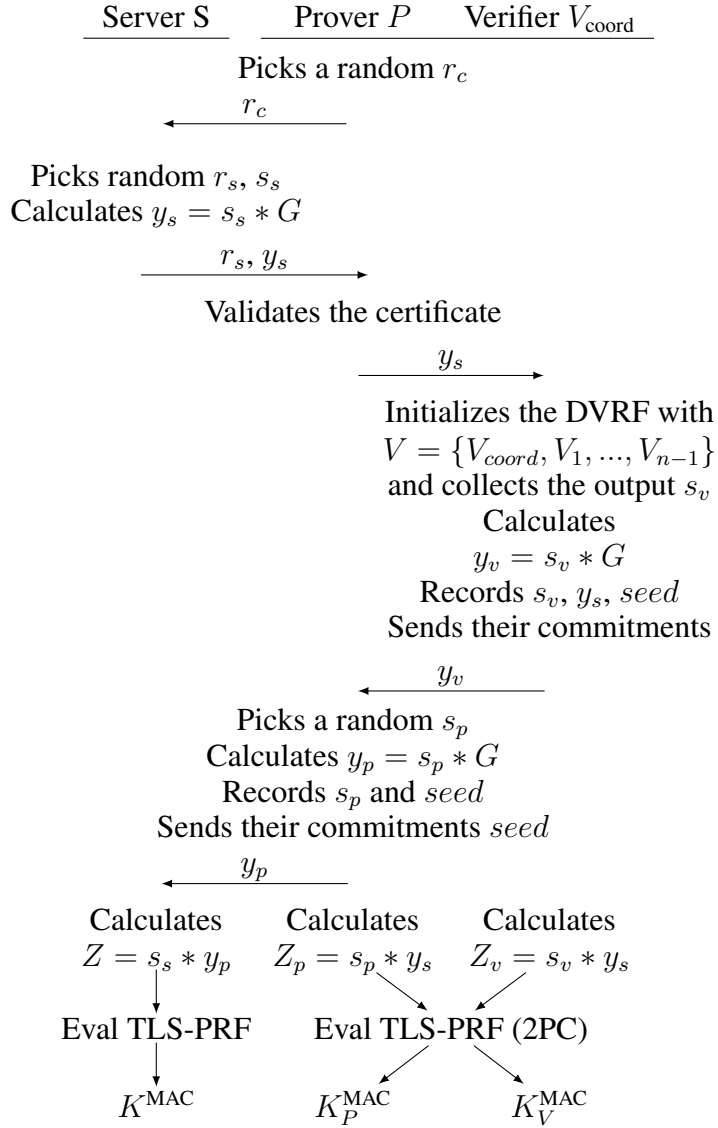


Figure 3.1: Overview of the Proposed Three-Party Handshake [45]

4. Prover P initializes the TLS protocol by sampling a random r_c and forwarding it to the server S in the *ClientHello* message.
5. Server S responds by picking two random numbers r_s, s_s , calculates $y_s = s_s * G$, where G is the generator of the elliptic group, forwarding r_s in *ServerHello* message and y_s to the prover P with a valid *Certificate*.
6. Prover P validates the *Certificate* and forwards y_s and the *Certificate* to the verifier V_{coord} .
7. V_{coord} forwards the *Certificate* to auxiliary verifiers in set V .

8. Each verifier V_i validates the *Certificate* and contributes the Distributed Verifier Random Function (DVRF) and V_{coord} outputs DVRF as s_v , then calculates $y_s = s_v * G$, and forwards it to Prover P . Lastly, verifier V_{coord} records s_v and y_s and Put their hashes as commitments to the decentralized storage and receives the related cid.
9. Prover P chooses another random s_p and calculates $y_p = s_p * G + y_v$, then forwards it to the server S . Next, Prover P records the s_p and Put its hash as commitment and gets related cid.
10. All three parties calculate their *Pre-master secret* keys as follows: the server S , $Z = s_s * y_p$, Prover P , $Z_p = s_p * y_s$ and verifier V , $Z_v = s_v * y_s$, where $Z = Z_p + Z_v$. In addition, V_{coord} and Prover P record the *seed* value similarly before.
11. All three parties derive the MAC keys from the *Pre-master secret* by evaluating TLS-PRF as in DECO.
12. Then, the query phase and proving phase are conducted as in DECO. Finally Prover P creates a valid ZKP that proves the private values are generated from the public values as described below: public $x = (Q, R, K^{\text{ENC}}, \theta_s)$ and private $w = (\hat{Q}, \hat{R}, K^{\text{MAC}}, b)$, where θ_s is the sensitive information that requires to be kept secret and b is the statement, (Q, R) is plaintext, (\hat{Q}, \hat{R}) ciphertext of query and corresponding response.
13. Each auxiliary verifier V_i performs the verification steps as specified in DECO by validating the zero-knowledge proof according to Definition 1 and checking that the MAC key computed locally matches the MAC key obtained from the prover.

Unless there is a dispute from auxiliary verifier V_i , the protocol ends the verification in the last step. Otherwise, under the assumption that one of the verifiers V_i requests a verification, both the coordinator V_{coord} and the prover P need to submit the pre-commitment values s_v , s_p , y_s , and the *seed*. In the verification phase, verifier V_i verifies the following conditions.

- Peers retrieve the required data from decentralized storage by invoking

the **Get** operation, as described in Section 3.1.3, using the corresponding content identifiers (**cids**).

- Calculates the commitments and then compares them with the commitments presented by verifier V_{coord} and prover P .
- Calculates $Z = (s_p + s_v) * y_s$.
- Evaluates the TLS-PRF with Z and $seed$ to obtain K^{MAC^*} .
- Compares K^{MAC^*} and K^{MAC} , then outputs either 0 or 1 according to whether the values.
- An optional reward and slashing mechanism can be applied during the verification process.

3.3 Protocol Analysis

Protocol analysis examines the security and performance characteristics of the proposed protocol. In particular, we first discuss its security properties under malicious and collusion-based threat models, and then analyze the additional communication, computation, and storage overhead introduced by the integration of DVRF and distributed verification mechanisms.

3.3.1 Security Discussion

Under the security definitions of DVRF and DECO, we define the security of the proposed protocol. The proposed protocol inherits prover and verifier integrity from DECO without modification.

The proposed protocol satisfies the t collusion-freeness property that: If V_{coord} colludes with the prover, the session is aborted by a verification request from any $V_i \in V$. Another saying for the collusion is that prover P requires to collude every verifier in the set V_{coord} , otherwise any verifier V_i who doesn't collude can break the process with the help of freshly generated s_v as the result of DVRF. In this scenario, after the request of V_i , V_{coord} needs to reveal the precommitment values. When the V_i calculates

the commitments again, prover P can realize the commitments cannot match since the V_{coord} uses a different s'_v from the distributed created one.

Consequently, from V_{coord} integrity, to attack with collusion, prover P needs to collude with all the members in the set of V , which is getting more infeasible when $|V|$ is increasing.

3.3.2 Efficiency Analysis

Compared to the original DECO protocol, the proposed protocol introduces additional communication, storage, and computation overhead due to the integration of DVRF, distributed coordination, and decentralized storage. We analyze the costs as follows.

- **Distributed Key Generation (DKG):** In the initialization phase of the DVRF, each of the n verifiers participates in a broadcast round to establish shared keys. Assuming a standard DKG protocol, each verifier sends messages to the remaining $n - 1$ verifiers. Thus, the total number of DKG messages is: $\text{Cost}_{\text{DKG}} = n(n - 1)$
- **DVRF Evaluation:** Each auxiliary verifier $V_i \in \{V_{\text{coord}}, V_2, \dots, V_n\}$ sends a partial evaluation to the coordinator verifier V_{coord} . The total number of messages in this phase is: $\text{Cost}_{\text{Eval}} = n - 1$
- **DVRF Combine:** After receiving all partial evaluations, the coordinator aggregates and broadcasts the final randomness to the other verifiers. This step involves: $\text{Cost}_{\text{Combine}} = n - 1$

In the event of a dispute, both the prover P and the coordinator verifier V_{coord} are required to reveal and submit their pre-committed values. Let c denote the number of committed values per party (e.g., $c = 4$ for s_v , s_p , y_s , and the *seed*), and assume that each commitment has a fixed length of λ bits (e.g., 256 bits). Under these assumptions, the total storage overhead incurred during dispute resolution is $\text{Storage}_{\text{dispute}} = 2c \cdot \lambda$, which remains modest and is incurred only in exceptional cases.

The proposed protocol introduces an additional communication cost of: $\text{Total}_{\text{comm}} = n(n - 1) + 2(n - 1)$ and an additional dispute-based storage cost of $2c \cdot \lambda$ bits. These overheads are justified by the added benefits of public verifiability, decentralized trust, and collusion resistance.

3.4 Summary

Taken together, this chapter shows that oracle collusion in DECO-based TLS attestations is not an inherent limitation of the protocol itself, but rather a consequence of the designated verifier assumption and its control over session initialization randomness. By re-examining this design choice, we demonstrate that collusion resistance can be achieved through verifier decentralization without altering the core prover logic or weakening the underlying TLS security guarantees. The integration of DVRFs enables a principled redistribution of trust, ensuring that no single verifier can unilaterally influence protocol execution.

At the same time, the chapter highlights the importance of combining cryptographic mechanisms with system-level design choices. The optimistic execution model and the use of decentralized storage ensure that the protocol remains efficient in the common case, while still providing guarantees in adversarial settings. As a result, the proposed construction offers a scalable path toward deploying DECO-based TLS attestations in permissionless and adversarial environments, such as smart contract platforms, where verifier trust assumptions must be minimized.

CHAPTER 4

COMPACT MULTI-VERIFIERS DECO WITH TRESHOLD SIGNATURE SCHEME

This chapter addresses the previous chapter’s practical limitation by presenting a deterministic and collusion-resistant TLS attestation protocol. The previous approach relied on trusted storage and optimistic execution to preserve attestation integrity, where protocol transcripts were stored off-chain and accessed only in the event of a dispute. While conceptually appealing, such designs suffer from limited robustness and practical deployability in adversarial and permissionless environments. In contrast, this chapter introduces a compact construction based on Threshold Signature Schemes (TSS) [7, 32, 42], enabling deterministic verification and improved usability for real-world decentralized applications.

The primary motivation of this chapter is to improve the practicality and deployability of collusion-resistant TLS attestation protocols by eliminating reliance on optimistic execution and trusted storage. While the optimistic design in Chapter 3 minimizes overhead in the common case, it still requires decentralized storage and dispute resolution mechanisms that complicate integration with on-chain systems. This observation motivates the search for a more compact and deterministic alternative. Replacing decentralized storage with Threshold Signature Schemes (TSS) introduces new design considerations. While TSS enables constant-size attestations and direct on-chain verification, it may also introduce additional network communication, coordination overhead, and on-chain verification costs. Understanding these trade-offs and identifying suitable deployment scenarios, where the benefits of compactness outweigh the introduced overhead become key motivating factor. Consequently, this chapter is also

motivated by the need to evaluate the practical costs and applicability of TSS-based designs across different decentralized use cases. This chapter is motivated by the question of whether a deterministic, collusion-resistant construction can be achieved while carefully managing the overhead introduced by TSS and reusing cryptographic components where possible.

In this chapter, we present a deterministic multi-verifier extension of DECO that replaces optimistic dispute resolution with threshold-based attestation using TSS. The proposed protocol leverages threshold signatures to achieve compact, constant-size attestations that are directly verifiable on-chain. To address the overhead introduced by TSS, we conduct a detailed analysis of both network-level communication costs and on-chain verification (gas) costs, summarizing the trade-offs across different threshold signature instantiations. A key contribution of this chapter is the reuse of Distributed Key Generation (DKG) as a shared primitive between DVRF and TSS, eliminating redundant setup phases and avoiding a major performance bottleneck. In addition, we present a mediator-free peer-to-peer decentralized application scenario to demonstrate the applicability of the proposed protocol in fully decentralized environments. The protocol design, performance analysis, and use-case evaluation were presented at the IEEE International Conference on Cyber Security and Resilience (CSR) 2025 [46].

This chapter proceeds with the following organization. Section 4.1 introduces the threshold signature background and discusses candidate TSS constructions relevant to our setting. Section 4.2 presents the deterministic protocol construction, detailing how TSS is integrated into the DECO handshake and attestation workflow. Section 4.3 analyzes the protocol's performance, including network communication overhead and on-chain verification costs. Finally, Section 4.4 presents a peer-to-peer decentralized application use case and discusses the security and deployability implications of the proposed design.

4.1 The Proposed System Architecture

In this section, we propose a protocol that combines DVRF and TSS to make the DECO, first, more resistant against the collusion attack according to the single verifier DECO mechanism and second, more efficient than the architecture that requires multiple DECO executions as shown in Figure 4.1. In this figure, black arrows denote joint randomness generation, red arrows denote the DECO Proof flow to verifiers, and blue arrows denote the TSS process.

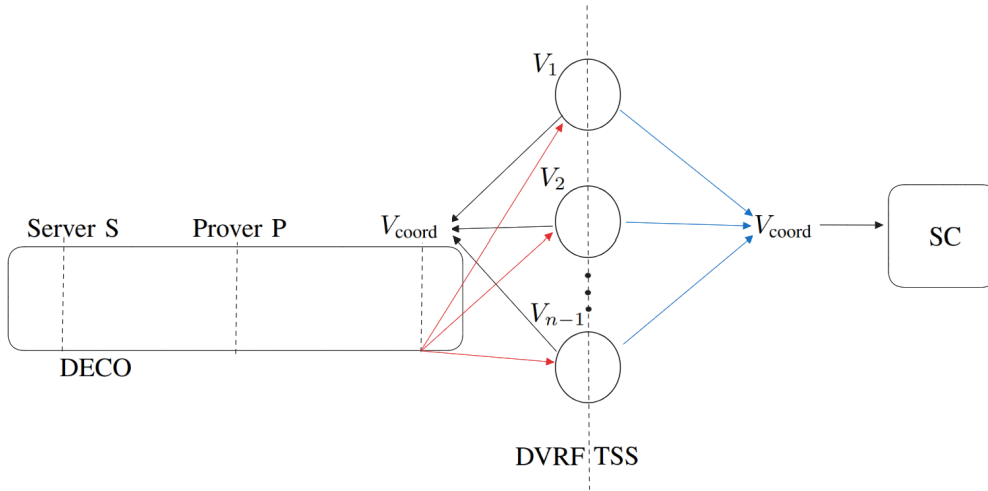


Figure 4.1: Overview of the Integration of DECO, DVRF, and TSS, Adapted from [46]

4.1.1 Construction

In the proposed protocol, the attestation process is deterministic and terminates with on-chain verification by the smart contract, in contrast to the approach of [45]. The protocol is defined over the same entities introduced in Section 3.2.1 and consists of the following four components: Protocol = (Handshake, Verification, Signing, Verification).

- **Handshake:** The proposed 3-party handshake protocol involves three roles: the server S , the prover P , and the set of verifiers $V = \{V_{\text{coord}}, V_2, \dots, V_n\}$, where $|V| = n$. The first verifier, V_{coord} , acts as the coordinator and is responsible for

facilitating interaction between the prover and the rest of the verifiers transparently. Basically, server S , prover P and verifier V_{coord} execute the DECO as it is only the difference V_{coord} is a proxy node and it uses the joint randomness coming from the DVRF instead of picking randomly as described in [45]. At the end of the handshake, prover P obtains the K_P^{MAC} while each verifier gets K_V^{MAC} as detailed in [45, 56]. We define V_{coord} without loss of generality, allowing it to be randomly selected from the verifier set as a safeguard against denial-of-service (DoS) attacks.

- **Verification:** With the previously determined keys, DECO is executed among server S , prover P and verifier V_{coord} . Firstly V_{coord} obtains the commitment of query and response from the prover P with K_P^{MAC} as in [56], then V_{coord} broadcasts the commitment to the nodes with K_P^{MAC} . Finally, V_{coord} obtains the DECO Proof, then broadcasts again to each verifier V_i . Upon receiving the DECO Proof, each verifier V_i checks: (i) K^{MAC} is equal $K_V^{MAC} + K_P^{MAC}$, (ii) ZKP proof is valid with its public input commitment of query and K^{MAC} , lastly (iii) K^{MAC} is created by the output of DVRF. Here, the first two checks come with the DECO inherently, and the last check is for proving the DECO execution to the set of verifiers that the DECO is executed with randomness, which is the output of the DVRF.
- **Signing:** In this phase, after each verifier V_i ensures the validity of the attestation by the DECO, then they start the signing phase with the same DKG in the DVRF phase. We expand the type and efficiency of various signatures in Section IV. Eventually, the signed attestation is written to the smart contract by V_{coord} .
- **Verification:** Smart contract validates the signature coming from the V_{coord} and initiates further transactions according to the design of the application. One might question whether DVRF output verification needs to be performed on-chain. In our design, this verification is handled by the verifiers off-chain. However, we note that an alternative design could be implemented, where the smart contract also verifies the DVRF output. This approach would shift some of the verification burden onto the blockchain, potentially increasing transparency but also introducing additional computational and gas costs. The choice between

these designs depends on the specific security and efficiency requirements of the application.

4.2 System Analysis

In this section, we evaluate the proposed scheme with respect to its efficiency and security.

4.2.1 Security Discussion

Since DECO already provides security against standalone malicious provers and verifiers, we focus exclusively on the threat model in which collusion is present in scenarios, where the prover and a subset of verifiers act maliciously in a coordinated manner. In the absence of collusion, the integrity guarantees of DECO are sufficient to ensure security against a single malicious prover or verifier. Beyond this baseline, the proposed scheme tolerates collusion between the prover P and up to $t - 1$ corrupted verifiers in the verifier set V .

More precisely, for a collusion attempt to succeed and result in the export of a valid attestation signature, the prover must collude with at least t verifiers. Collusion involving fewer than t verifiers is insufficient to violate the security of the protocol, as such coalitions cannot overcome the combined protections provided by DECO integrity, the uniqueness and threshold security of the DVRF, and the unforgeability guarantees of the threshold signature scheme. The following discussion elaborates on why collusion below this threshold cannot lead to a successful attack.

We now discuss the security of the proposed protocol under collusion between the prover and a subset of verifiers. In particular, we consider the adversarial setting in which the prover colludes with at most $t - 1$ verifiers from the verifier set V , where $|V| = n$ and $n/2 < t$. The threshold parameter t is shared by both the distributed verifiable random function (DVRF) and the threshold signature scheme (TSS).

The security guarantees of the protocol arise from the combined properties of its underlying components: verifier and prover integrity of DECO, uniqueness and thresh-

old security of the DVRF, and the unforgeability of the TSS. Together, these properties ensure that collusion between the prover and fewer than t verifiers is insufficient to produce a valid attestation signature.

We consider the strongest relevant adversarial strategy in which the prover first colludes with the coordinator verifier V_{coord} . If V_{coord} is honest, the DECO component enforces verifier integrity and prevents any deviation at this stage. Consequently, a successful collusion attempt necessarily assumes that V_{coord} is among the corrupted verifiers.

Under this assumption, the prover may attempt to submit an invalid attestation. Such an attestation can take the form of a verifiable DECO Proof generated using randomness that is inconsistent with the output of the DVRF, or a malformed (non-verifiable) DECO Proof. The colluding coordinator then broadcasts this invalid attestation to the verifier set V .

Honest verifiers independently validate the attestation by checking consistency between the DECO Proof and the DVRF output. Due to the uniqueness property of the DVRF, the derived MAC key K^{MAC} is uniquely determined and cannot be substituted or recomputed using alternative randomness. As a result, any attestation that does not correctly bind to the DVRF output is rejected by all honest verifiers.

While the $t - 1$ colluding verifiers may locally accept the invalid attestation, they are collectively insufficient to produce a valid threshold signature. The unforgeability property of the TSS ensures that fewer than t signature shares cannot be combined into a valid attestation signature. Consequently, the protocol prevents the export of a valid attestation even under coordinated collusion between the prover and up to $t - 1$ verifiers.

Overall, this discussion illustrates that the protocol maintains security against collusion attacks as long as the number of corrupted verifiers remains below the threshold. The interaction between DECO integrity, DVRF uniqueness, and threshold signature unforgeability ensures that invalid attestations cannot be escalated into valid protocol outputs.

4.2.2 Efficiency Analysis

The efficiency of the proposed scheme can be analyzed in three categories: computational cost, network cost, and gas cost evaluation. It is important to note that while the DECO and DVRF components remain unchanged, we evaluate three different signature scheme alternatives.

- **Computational cost:** The proposed scheme comprises the execution of a Distributed Key Generation (DKG), a DECO protocol, joint randomness generation using DKG, and a single signing process. We analyze the choice of signature schemes, including Multi-Sig ECDSA, BLS-TSS, and FROST, as presented in Table 4.1, while acknowledging that our approach is not limited to these options. Here, Multi-Sig is defined as a straightforward implementation of multiple ECDSA [29] without requiring a key distribution process. Each verifier independently generates their own public-private key pair and signs the attestation before sending it to the smart contract.
- **Network cost:** In addition to the messaging required by DECO, the proposed scheme necessitates additional network communication between V_{coord} and the verifier set V . In total, V_{coord} performs three broadcast messages, while the set V undergoes two peer-to-peer communication phases, one for randomness generation using DVRF and another for threshold signing. Notably, if the Multi-Sig option is utilized, it does not require any rounds for signing; however, this setting requires a *n-out-of-n* model and each verifier to have the wallet account.
- **Gas cost:** The gas cost is the signature verification cost on-chain. It varies depending on the chosen scheme and the underlying elliptic curve. For example, ECDSA and FROST do not require pairing check operation in verification; they are performed in the *secp256k1* curve [44]. However, threshold BLS (TSS-BLS) is required to be operated in pairing-friendly curves such as *altbn128* as in [10]. Multi-Sig ECDSA has a special precompiled contract named *ecRecover* so it takes only 3,000 gas cost to verify a single ECDSA signature with 2 elliptic curve multiplication (ECMUL) and 1 elliptic curve addition (ECADD), and one hash execution. Since this is a multi-signature the total cost will be $t \times 3,000$, where t is the number of the threshold. Secondly, FROST signature

verification takes 4,200 gas on the *secp256k1* curve according to an experimental study [43]. The same verification costs about 12,000 gas on the *altbn128* curve as shown in [10]. The last option is threshold BLS (TSS-BLS), but its verification incurs a high cost of approximately 115,000 gas due to two pairing checks operation in the *altbn128* curve [10]. Note that, FROST [32] offers efficiency in terms of gas costs but requires two communication rounds when using a transparent aggregator. Alternatively, it supports a single-round aggregation; however, this approach relies on a trusted aggregator, which is incompatible with our architecture’s trust assumptions. All the gas cost examination here only covers the verification of the signature; the total gas cost will be higher with additional transaction costs.

Table 4.1: Comparison of Signature Schemes from [46], Where t Denotes the Threshold Parameter Used in DVRF and TSS Constructions

Signature Scheme	Rounds	Verification Cost	Gas Cost <i>secp256k1</i> - <i>altbn128</i>
Multi-Sig ECDSA [29]	0	$2t$ ECMUL $2t$ ECADD t Hash to \mathbb{G}	$3,000t$ - N/A
TSS-BLS [7]	1	2 Pairing check 1 Hash to \mathbb{G}	N/A - 115,000
FROST [32]	2	2 ECMUL 2 ECADD 1 Hash to \mathbb{G}	4,200 - 12,000

The overall comparison of representative signature schemes used for attestation signing is summarized in Table 4.1, adapted from [46]. The comparison highlights differences in signature size, verification cost, and suitability for threshold deployment, where the threshold parameter t denotes the minimum number of participants required to produce a valid signature. These properties motivate the choice of threshold signature schemes in the proposed architecture, as they enable compact attestations, efficient on-chain verification, and robustness against verifier failures and collusion. In practice, the choice of a concrete signature scheme depends on deployment constraints. For small threshold values t , Multi-Sig ECDSA schemes that do not require aggregation can be sufficient. When network bandwidth is not a primary concern,

schemes such as FROST provide efficient threshold signing with low verification overhead. For larger threshold values and settings with stricter bandwidth requirements, TSS-BLS signatures can be preferred despite higher computational cost, as they offer constant-size signatures and efficient verification. The proposed scheme also ensures availability by leveraging the t -out-of- n threshold, allowing the system to tolerate up to $n - t$ offline verifiers while still maintaining functionality.

4.3 A Use-Case Example

Although there are many decentralized applications (dApps) in use that do not require any website attestation, such as decentralized exchanges (DEXs), some dApps require website attestation for expanded functionality. Insurance is counted as one of these areas because, upon purchasing the insurance using cryptocurrency, the dApps require incident proof for reimbursement, which can be provided from trusted websites.

Two studies on decentralized insurance applications, [3, 23], exhibit limited functionality since both require insurance providers and insurance notaries to be part of the blockchain network, which is impractical in real-world scenarios, where external entities may not have direct blockchain participation.

We illustrate the application of the proposed scheme with the example of a flight delay insurance decentralized application (Dapp) without requiring any intermediate notary. The example scheme consists of **setup**, **pay-in**, and **reimbursement** sections as shown in Figure 4.2.

Setup: We assume that the flight delay smart contract (SC) is already deployed with two functions, where the pay-in function is for purchasing the insurance and the pay-out function that validates the attestation and initializes a reimbursement transaction. Also, let the verifier set $V = \{V_{\text{coord}}, V_2, \dots, V_n\}$, and each verifier obtains the same public key pk and corresponding shared secret keys sk_i , where $i \in (0, t)$ by using DKG as described in the DRVF section.

Pay-in: Assume that Alice wants to buy flight delay insurance. Firstly, Alice commits her flight information as PNR number, timestamp, and threshold $C = (\text{PNRnumber}_c,$

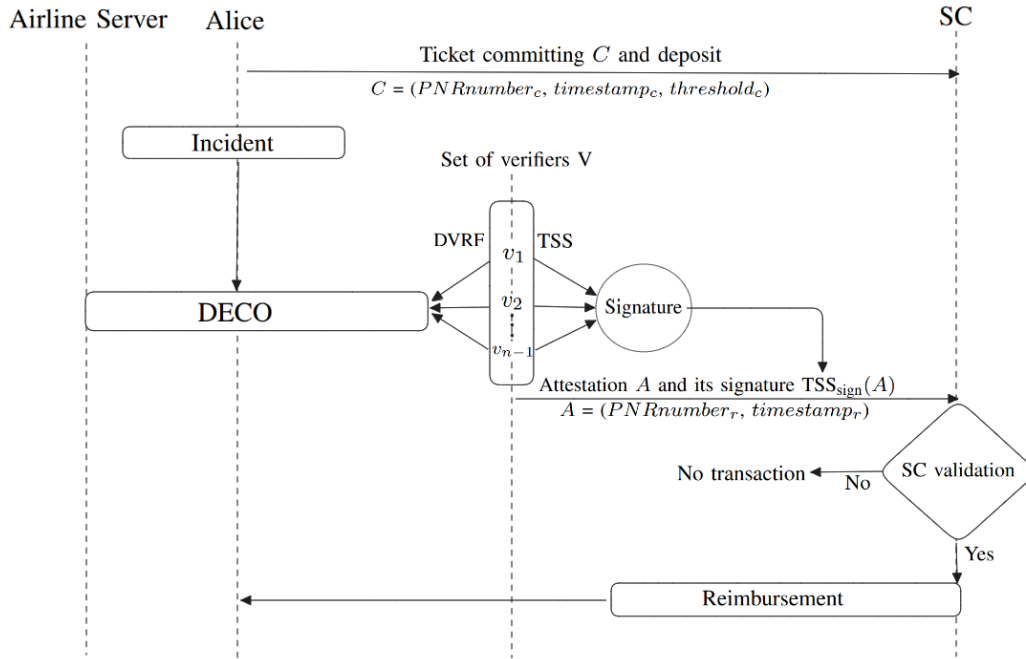


Figure 4.2: Overview of the Proposed Scheme in an Example Flight Delay Insurance Use Case [46]

$timestamp_c, threshold_c$), and then a binary calculates the insurance premium. Then, Alice buys the insurance by sending C and the corresponding premium fee to the SC.

Reimbursement: Suppose that Alice’s flight is delayed more than the threshold. To payment, Alice initializes the DECO component of the proposed scheme with the server S as the airline’s website, the prover P is Alice, and the verifier set is V_{coord} in V . After the execution of the DECO component successfully, each verifier in the set V ensures the attestation from the airline website such as $A = (PNRnumber_r, timestamp_r)$. Then, the verifiers jointly sign the attestation A by using the same DKG and then send $TSS_{sign}(A)$ to the SC. The SC verifies the signature and initiates a payout transaction to Alice if $timestamp_r - timestamp_c < threshold_c$. Note that, the rest of the information about the ticket, such as metadata or seat number, can be kept secret in the proving phase of the DECO part of the proposed scheme.

In conclusion, the proposed scheme minimizes the collusion risk because even if Alice and V_{coord} have a secret agreement, Alice needs to collude with at least $t - 1$ more verifiers. By mitigating the risk of collusion, individuals can develop fully functional dApps without the need for intermediaries.

4.4 Summary

This chapter shows that eliminating the optimistic approach and decentralized storage does not weaken the collusion resistance of DECO-based TLS attestation protocols. By adopting a deterministic design based on threshold signatures, the proposed construction removes dispute-driven execution paths and enables compact, directly verifiable attestations suitable for on-chain environments. As a result, the protocol improves deployability and operational simplicity while preserving the core security guarantees required for adversarial and permissionless settings.

The chapter further demonstrates that the additional coordination and verification overhead introduced by Threshold Signature Schemes can be systematically analyzed and effectively managed. Through detailed evaluations of network communication and on-chain verification costs, as well as the reuse of Distributed Key Generation (DKG) across cryptographic primitives, the construction achieves practical efficiency in fully decentralized deployments. Moreover, the joint use of DVRF and TSS in this chapter establishes a combined DVRF-TSS primitive, which serves as a foundational building block reused and extended in the subsequent chapter. Together, these results position threshold-based TLS attestation as a viable and scalable alternative for decentralized applications.

CHAPTER 5

DESIGNED EXPORTABLE DCTLS (DX-DCTLS)

This chapter introduces designed exportable DCTLS (*dx-DCTLS*), a generic abstraction for collusion-minimized TLS attestations with verifiability beyond designated verifiers. Unlike a standalone end-to-end protocol, *dx-DCTLS* is not intended as a complete construction by itself; rather, it serves as a verifiable execution component that can be composed with distributed validation mechanisms. Building on the DVRF-TSS construction developed in the previous chapter, we revisit DCTLS protocols from a foundational perspective and explicitly separate protocol execution from verification. Prior chapters focused on concrete constructions instantiated over DECO and did not aim to provide a unified security definition applicable across different TLS versions. In contrast, this chapter establishes a common security foundation by jointly considering DECO [56] and the construction of Distefano [11], thereby extending the analysis to both TLS 1.2 and TLS 1.3. Through this comparative analysis, we identify that existing DCTLS constructions lack an explicit notion of exportable verification: while correctness may be locally checked by a participating verifier, the full protocol execution is not externally verifiable by third parties. To address this limitation, we first define a unified DCTLS execution model and formalize its security requirements. We then observe that this model alone is insufficient to support externally verifiable correctness across the entire protocol execution. This insight motivates the introduction of *dx-DCTLS*, an execution model in which protocol correctness remains verifiable throughout all interaction phases, from initialization to completion, under a fixed public input. By construction, *dx-DCTLS* enables the export of verifiable execution traces that can be validated independently of protocol participation. When composed with the DVRF-TSS primitive introduced earlier, this abstraction yields a

concrete collusion-minimized TLS attestation protocol with formally defined security guarantees.

The motivation of this chapter is to provide a clear and formal basis for reasoning about collusion-minimized TLS attestations. Existing DCTLS protocols tightly couple protocol execution and verification to a participating verifier, which limits correctness checks to local execution contexts. While this design is sufficient for concrete implementations, it makes it difficult to define security properties that support external verification and to reason about correctness and unforgeability once verification is delegated to third parties. In addition, prior DCTLS constructions are developed as standalone designs, tied to specific protocol instantiations and TLS versions, and are typically evaluated only at the implementation level. As a result, there is no unified execution model that enables formal security definitions, systematic comparison across constructions, or principled composition with distributed validation mechanisms. This chapter is motivated by the need to close this gap by introducing explicit security definitions that capture collusion resistance, and by showing how these definitions can be satisfied by concrete constructions. First, we aim to formalize the execution and verification properties required to minimize collusion in DCTLS protocols. Second, we show that the proposed *dx-DCTLS* abstraction, when composed with the DVRF-TSS construction introduced in the previous chapter, provides integrity and external verifiability under these formal definitions. Finally, we demonstrate that the resulting construction is feasible in practice by evaluating its implementation overhead and comparing it against closely related DCTLS protocols, thereby motivating its applicability beyond a purely theoretical setting.

As a contribution of this chapter, we formalize *dx-DCTLS* as a generic abstraction for core components of verifiable TLS attestation. We demonstrate how representative DCTLS constructions spanning different design points can be systematically transformed into this abstraction. In particular, we analyze DECO [56], which targets TLS 1.2, and the construction of Distefano et al. [11], which supports TLS 1.3, and identify common verification structures that can be decoupled from protocol execution. Building on this abstraction, we instantiate *dx-DCTLS* using the DVRF-TSS primitive developed in the previous chapter, yielding a concrete collusion-minimized TLS attestation protocol $\Pi_{\text{coll-min}}$ with constant prover complexity and threshold-

based verification. We gave the security properties of $\Pi_{\text{coll-min}}$ in a game-based model and proved that the threshold attestation unforgeability feature is satisfied under standard cryptographic assumptions of DCTLS and DVRF. We show that the DVRF-TSS primitive of $\Pi_{\text{coll-min}}$ can be implemented using a precomputed Distributed Key Generation (DKG) and scaled to deployments of up to 50 nodes. In addition, we analyze the additional overhead introduced by *dx-DCTLS* and derive an upper bound on the extra cost incurred over standard DCTLS executions. Finally, we present representative use cases, including confidential binary options and off-chain credit and income verification, demonstrating how *dx-DCTLS* enables externally verifiable attestations in realistic decentralized application settings. The results presented in this chapter have been published in the Cryptology ePrint Archive as [49].

The remainder of this chapter is organized as described below. Section 5.1 formalizes the *dx-DCTLS* abstraction and introduces the exportable verification model. Section 5.2 analyzes existing DCTLS constructions and shows how DECO and Distanzo et al. can be transformed into the *dx-DCTLS* framework. Section 5.3 presents the instantiation of *dx-DCTLS* using the DVRF-TSS primitive and defines the protocol $\Pi_{\text{coll-min}}$. Finally, Section 5.4 provides a formal security analysis and proves the unforgeability of threshold attestations.

5.1 Building Blocks

In this section, we introduce the formal security definitions and notation used throughout the remainder of this chapter. The underlying primitives were introduced earlier in Chapter 2 and are extended here with their corresponding security definitions.

5.1.1 Game-based security notations

For the security game G , which is a forgery used in the cryptographic scheme Δ , we denote the advantage of an \mathcal{A} in Δ by $\text{Adv}_{\mathcal{A},\Delta}^G(\lambda)$, where

$$\text{Adv}_{\mathcal{A},\Delta}^G(\lambda) = \Pr[\text{Exp}_{\mathcal{A},\Delta}^G(\lambda) = 1].$$

By definition, Δ is secure with respect to the game G iff $\text{Adv}_{\mathcal{A},\Delta}^G(\lambda) \leq \text{negl}(\lambda)$ for some function $\text{negl}(\cdot)$ and the security parameter λ .

5.1.2 DVRF Security Definition

We rely on a Distributed Verifiable Random Function (DVRF) construction, as introduced in Chapter 2. A DVRF consists of the algorithms `DKG`, `PartialEval`, `Combine`, and `Verify`, and enables a set of distributed parties to jointly compute a publicly verifiable random value.

A DVRF provides the standard security properties of consistency, robustness, and uniqueness. Consistency ensures that all honest parties derive the same output for a given input, robustness guarantees that the output can be computed despite a bounded number of faulty or malicious participants, and uniqueness ensures that for any fixed input there exists at most one valid output that verifies under the public key.

Figure 5.1 illustrates the uniqueness experiment for the Distributed Verifiable Random Function (DVRF). The experiment formalizes the requirement that, for a fixed public key and input, there exists exactly one valid output that can be verified successfully. This property is essential in the proposed framework because the jointly generated randomness is later used to bind multiple verifiers to a single TLS session execution. If uniqueness were violated, an adversary could produce inconsistent randomness values and induce divergent verification outcomes among verifiers. Therefore, the experiment supports the correctness and consistency guarantees required by multi-verifier attestation.

DVRF Uniqueness Game. For a given (t, n) -threshold DVRF scheme $\text{DVRF} = (\text{DKG}, \text{PartialEval}, \text{Combine}, \text{Verify})$ and a PPT adversary \mathcal{A} , we define the uniqueness experiment as in Figure 5.1:

- The adversary \mathcal{A} selects a set of corrupted nodes C such that $|C| \leq t$. The challenger controls the remaining honest nodes. All parties execute $\text{DKG}(1^\lambda, t, n)$ to generate the public parameters pk and the verification keys VK .
- The adversary \mathcal{A} can request partial evaluations (`PartialEval`) on any input α

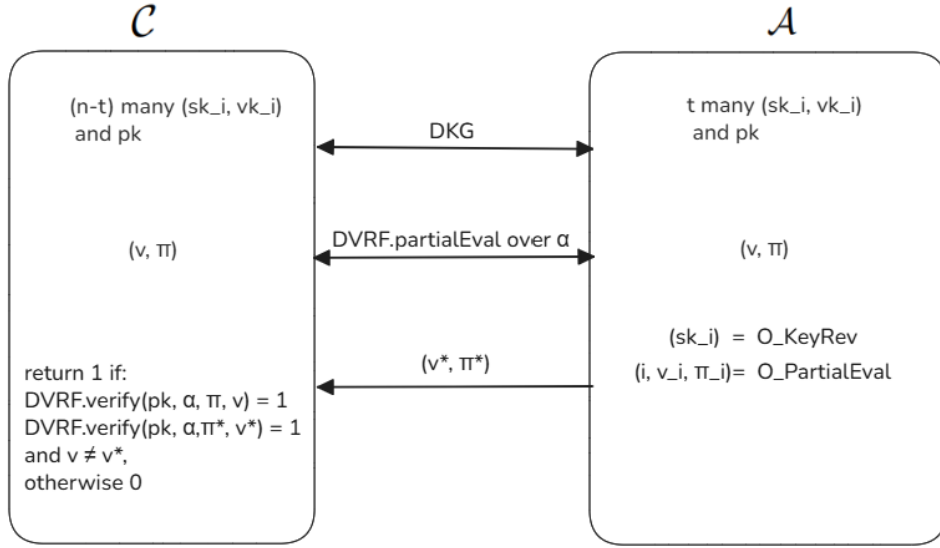


Figure 5.1: DVRF Uniqueness Experiment adapted from [49]

from $\mathcal{O}^{\text{partial-eval-reveal}}$, or reveal the secret key (KeyRev) of specific honest nodes from $\mathcal{O}^{\text{key-reveal}}$.

- \mathcal{A} and \mathcal{C} run DKG.PartialEval on α to calculate and then exchange partial evaluations, finally both calculate final randomness and proof, namely (v, π) .
- Adversary \mathcal{A} forges output (v^*, π^*) .
- \mathcal{C} outputs 1, if $\text{DVRF.Verify}(pk, \alpha, \pi_{\text{DVRF}}^*, v^*) = \text{DVRF.Verify}(pk, \alpha, \pi_{\text{DVRF}}, v) = 1$, otherwise outputs 0.

$$\text{Adv}_{\mathcal{A}, \text{DVRF}}^{\text{Uniq}}(\lambda) = \Pr[\text{Exp}_{\mathcal{A}, \text{DVRF}}^{\text{Uniq}}(\lambda) = 1] \leq \text{negl}(\lambda)$$

5.1.3 TSS Security Definition

We rely on a threshold signature scheme (TSS), as defined in Chapter 2. A TSS enables a set of n participants to jointly produce a valid signature on a message, such that any subset of at least t participants can generate a signature that verifies under a common public key.

A threshold signature scheme provides threshold existential unforgeability, which guarantees that no adversary controlling fewer than t participants can forge a valid signature on a new message. This property is essential for our construction, as it

ensures that attestations cannot be approved unless a sufficient number of auxiliary verifiers jointly authorize the result.

Since our security analysis relies solely on the unforgeability guarantee of the threshold signature scheme, we do not restate the underlying algorithms or correctness properties here and instead treat TSS as a cryptographic primitive with standard security guarantees.

Threshold Existential Unforgeability Game. A TSS provides the *threshold existential unforgeability* (TEU) as a security property.

Let $\text{TSS} = (\text{DKG}, \text{Signing}, \text{Verify})$ be a t -out-of- n threshold signature scheme. We say TSS is *existentially unforgeable* under chosen message attacks if the following experiment outputs 1 with negligible advantage $\text{negl}(\lambda)$ for all PPT adversary \mathcal{A} :

- \mathcal{A} chooses a corruption set $C \subseteq \{1, 2, \dots, n\}$ with $|C| \leq t$ while \mathcal{C} controls the honest nodes in $V \setminus C$.
- \mathcal{C} and \mathcal{A} run $\text{DKG}(1^\lambda, t, n)$. Honest nodes receive their respective secret shares sk_i for $i \in V \setminus C$, while \mathcal{A} learns the shares sk_i for $i \in C$ and the public key pk .
- \mathcal{A} may query $\mathcal{O}^{\text{signing-oracle}}$ for signing messages α of its choice. For each query on α , the \mathcal{A} returns partial signatures $\sigma_i = \text{Sign}(\alpha, sk_i)$ from any subset of honest nodes, where M is the set of queried messages.
- Eventually, \mathcal{A} outputs and sends (α^*, σ^*) to \mathcal{C} .
- \mathcal{C} outputs 1 if $\text{Verify}(pk, \alpha^*, \sigma^*) = 1$ and $\alpha^* \notin M$. Otherwise, it outputs 0.

$$\text{Adv}_{\mathcal{A}, \text{TSS}}^{\text{TEU}}(\lambda) = \Pr[\text{Exp}_{\mathcal{A}, \text{TSS}}^{\text{TEU}}(\lambda) = 1] \leq \text{negl}(\lambda)$$

5.1.4 v2PC Security Definition

We adopt a verifiable two-party computation (v2PC) primitive, as introduced in Chapter 2 and inspired by the Publicly Verifiable Covert (PVC) model of [58]. A v2PC protocol enables two parties to jointly evaluate a deterministic function while

producing publicly verifiable evidence that the computation was executed correctly, without revealing either party’s private input.

A $v2PC$ scheme provides a soundness guarantee, ensuring that any deviation from the agreed-upon computation by a malicious participant can be detected and proven to an external verifier. This property is essential for our construction, as it allows the outcome of a two-party computation to be exported and validated by auxiliary verifiers.

Within our framework, $v2PC$ replaces non-verifiable two-party computation components in existing $DCTLS$ protocols. This substitution enables auxiliary verifiers to independently validate that critical protocol steps were executed correctly and bound to the intended session state. By relying on verifiable execution rather than designated trust, $v2PC$ prevents transcript reuse and manipulation by a malicious prover–verifier pair. Since the security analysis depends only on the soundness and public verifiability guarantees of $v2PC$, we omit the underlying algorithmic details here and treat $v2PC$ as a cryptographic primitive with standard security properties.

5.1.5 Co-SNARK Security Definition

We make use of collaborative zero-knowledge succinct non-interactive arguments of knowledge (co -SNARKs), building on the construction introduced in [37]. Unlike standard zk -SNARKs, which require a single prover to hold the entire witness, co -SNARKs enable multiple provers to jointly generate a succinct proof while keeping their individual witness shares private. This makes them well-suited for settings in which sensitive inputs are inherently distributed.

Within our framework, co -SNARKs are employed to replace non-verifiable two-party computation components in selected phases of $DCTLS$ protocols. This allows protocol participants to jointly attest to the correct execution of cryptographic computations without reconstructing the full witness at any single party. As a result, TLS -derived secrets remain distributed while the resulting proof can be publicly verified by auxiliary verifiers.

A co -SNARK scheme provides the standard security properties of completeness, suc-

cinctness, zero-knowledge, and knowledge soundness. For the purposes of this thesis, however, the analysis relies exclusively on the soundness property, which guarantees that no adversary can produce a valid proof for an incorrect statement. We therefore omit the algorithmic specification of **co-SNARKs** here and treat them as a cryptographic primitive with standard soundness guarantees.

5.2 Problem Formulation

This section focuses on the problem formulation specific to **dx-DCTLs**. The general challenges of on-chain TLS attestations and the limitations of existing approaches have been discussed in earlier chapters. Here, we directly formalize the setting and assumptions relevant to collusion-minimized and jointly verifiable TLS attestations.

On-chain TLS Attestation with DCTLs. In an on-chain TLS attestation setting based on **DCTLs**, four parties are involved: a server S , a prover P , a verifier V , and a smart contract **SC**. The prover P is the data owner who wishes to convince the smart contract that a statement b , such as age eligibility or proof of solvency, originates from an authentic TLS-protected interaction with the server S , while selectively hiding sensitive parts of the underlying data. Since the smart contract cannot participate in interactive protocols, the verifier V validates the TLS attestation and submits the result to **SC**, acting as an oracle, as shown in Figure 5.2. In this figure, the prover P proves statements about data held by the server S to a verifier V , who subsequently records the verified result on the smart contract.

To enable this functionality, the prover and verifier jointly emulate a single TLS client and interact with the server through a standard TLS session. The verifier is expected to be convinced that the attested statement b was obtained with integrity from a genuine session with the server before submitting it to the smart contract. This design relies on the assumption that the verifier behaves honestly and does not collude with the prover.

In decentralized settings, this designated verifier assumption is insufficient. A colluding prover and verifier can fabricate attestations that appear valid to the smart contract, thereby undermining the correctness of on-chain applications. Existing approaches

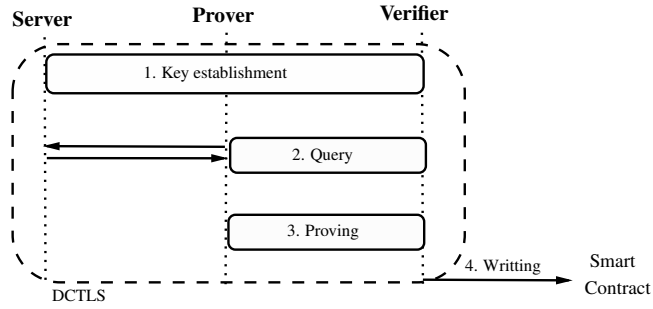


Figure 5.2: Overview of the DCTLs-Based Workflow for Smart Contracts [49]

mitigate this risk through verifier replication, trusted hardware, or additional trust assumptions, which introduce trade-offs in prover scalability, deployment complexity, or trust.

dx-DCTLs Setting. We consider a setting in which the verifier role is distributed across a set of verifiers V with distinct responsibilities. The protocol must allow the prover to participate in only a single DCTLs execution in order to keep prover complexity low. At the same time, multiple auxiliary verifiers should be able to validate the correctness of this execution.

Only one verifier interacts directly with the prover during the DCTLs protocol. The remaining verifiers do not participate in the interactive TLS session. Instead, they contribute to the generation of shared randomness and verify that the attestation execution is correctly bound to this jointly agreed randomness. If the auxiliary verifiers can confirm that the attestation was executed with the agreed randomness, a single DCTLs execution suffices to convince the entire verifier set.

Under this formulation, successful forgery no longer depends on compromising a single verifier. Instead, a prover must collude with multiple verifiers to produce an attestation that is accepted by the smart contract. The remaining challenge is to express this collective validation in a form that can be efficiently verified and accepted on-chain.

Without loss of generality, we assume that the smart contract accepts attestations only from whitelisted website servers. Attacks involving a malicious server or a compromised smart contract are outside the scope of this work, as the focus is on minimizing verifier-side trust assumptions.

5.3 Syntax

This section defines the syntax of the proposed primitive for collusion-minimized TLS attestations. The syntax specifies the entities, cryptographic components, and algorithmic interfaces that underlie the constructions and security definitions presented in subsequent sections. The focus here is on formalizing inputs, outputs, and roles, rather than protocol instantiation details.

Entities. We adopt the system entities from Section 3.2.1. Rather than redefining the entities, we briefly recall their roles as they appear in the context of the protocol construction presented in this chapter.

Algorithms. The primitive is defined through four algorithms: **Setup**, **Randomness Creation**, **Attestation**, and **Signing**. This section fixes the syntax of each algorithm by specifying its inputs and outputs.

- **Setup**(1^λ): Generates public parameters pp and system metadata for a verifier set of size n with threshold t .
- **Randomness Creation**(pp): Randomness creation is realized via a distributed verifiable random function. The distributed verifiable random function (DVRF) follows the standard construction defined in the preliminaries and consists of the following components: (i) **DKG**, which establishes threshold key material among the verifier set and outputs a public key, per-verifier verification keys, secret key shares, and a qualified set of participants; (ii) **PartialEval**, which allows each verifier to compute a partial evaluation for an input string α together with an associated correctness proof; (iii) **Combine**, which aggregates at least t valid partial evaluations into a unique randomness value and a public verification proof; and (iv) **Verify**, which checks that the derived randomness is a valid DVRF output for the given input.
- **Attestation**($rand$): Defines an interactive protocol, denoted **dx-DCTLS**, executed among the coordinator verifier V_{coord} , the prover P , and the server S . The function takes as input the agreed randomness $rand$ and outputs an attested statement b together with an exportable attestation proof π_a . The proof

π_a contains the information required by auxiliary verifiers to validate that the attestation was executed correctly under $rand$. The proof is broadcast to all $V_i \in V \setminus \{V_{\text{coord}}\}$. The dx-DCTLS protocol consists of the following subprotocols.

- $\text{HSP}(1^\lambda; rand) \rightarrow (sps, sp_p, sp_v, \pi_{\text{HSP}})$: A handshake subprotocol that derives TLS key material while binding the session to the designated randomness $rand$. The server S obtains sps , while the prover and the coordinator verifier obtain additive shares (sp_p, sp_v) . The proof π_{HSP} attests that the handshake was executed with the designated randomness $rand$, which distinguishes dx-DCTLS from prior DCTLS constructions.
- $\text{QP}(sp_s, sp_p, sp_v) \rightarrow (Q, R, \hat{Q}, \hat{R})$: A query subprotocol jointly executed by the prover and the coordinator verifier. Since both parties hold additive shares of the TLS session keys, neither can perform the query independently. To obtain a valid server response, the prover embeds a valid session secret θ_s into the query. The prover obtains the plaintext query and response (Q, R) , while the coordinator verifier obtains the committed encrypted values (\hat{Q}, \hat{R}) .
- $\text{PGP}(x, w) \rightarrow \pi_{\text{pgp}}$: A proof generation subprotocol that outputs a zero-knowledge proof $\pi_{\text{pgp}} \leftarrow \text{ZKP.Prove}(x, w)$. The public input is $w = (\hat{Q}, \hat{R}, sp, b)$, and the witness is $x = (Q, R, \theta_s)$. The proof attests that the statement b is correctly derived from (Q, R) and is consistent with the committed values (\hat{Q}, \hat{R}) . The value sp is derived from the designated randomness $rand$ during the handshake phase.
- $\text{Signing}(rand, \pi_{\text{pgp}}, \pi_a, b)$: Defines the validation and approval phase executed by auxiliary verifiers. Each auxiliary verifier checks the correctness of the attestation proofs under the agreed randomness $rand$. If verification succeeds, the verifiers engage in a threshold signing protocol to produce a signature consumable by the smart contract.
- $\text{DKG}(\cdot) \rightarrow (sk_i, pk)$: Reloads the DVRF key material without recomputing distributed key generation.

- $\text{Sign}(b, sk_i) \rightarrow \sigma$: Each auxiliary verifier signs the statement b using its secret key share sk_i . The resulting threshold signature σ and the statement b are submitted to the smart contract for on-chain verification.

5.4 Security Definitions

This section presents a formal treatment of the security requirements for the proposed collusion-minimized DCTLS-based attestation framework. The definitions presented here build on the system model and syntax introduced in previous chapters and focus on capturing adversarial behavior in decentralized, on-chain verification settings.

We consider a strong adversarial model in which verifiers in the set V are not assumed to be honest or semi-honest. An adversary may corrupt verifiers adaptively, cause them to deviate arbitrarily from the protocol, and allow them to collude with each other and with the prover in an attempt to forge or bias TLS attestations. Corruption may occur at any point during protocol execution, and the adversary obtains the internal state of corrupted parties.

We assume that the TLS server correctly executes the standard TLS protocol and does not behave maliciously or collude with the prover or verifiers. Similarly, the smart contract is assumed to be correctly deployed and to execute deterministically according to its code. Attacks involving a malicious server or a compromised smart contract are outside the scope of this work. Under these assumptions, the primary security objective of the framework is captured through the notion of *threshold attestation unforgeability*.

5.4.1 Threshold Attestation Unforgeability

We consider an attestation setting in which verification is carried out by a set of verifiers $V = \{V_1, V_2, \dots, V_n\}$. During an execution of the protocol, an adversary \mathcal{A} may corrupt the prover and an arbitrary subset of verifiers, with the restriction that at least one verifier remains honest throughout the protocol execution. Corrupted parties are fully controlled by \mathcal{A} , while honest verifiers follow the protocol specification.

The adversary interacts adaptively with honest verifiers and eventually attempts to produce a forged attestation proof in a collusion setting, denoted by the bad event B_{coll} . The security requirement is that the probability of this event occurring is negligible for any probabilistic polynomial-time adversary.

Figure 5.3 presents the threshold attestation unforgeability experiment, which captures adversarial behavior in a multi-verifier setting. The experiment models an adversary that interacts with threshold signing and verification oracles and attempts to produce a valid attestation without collecting a sufficient number of verifier contributions. This definition ensures that attestations cannot be forged unless the adversary controls at least a threshold number of auxiliary verifiers. Therefore, this shows that collusion risk is mitigated. The experiment is central to the security of the proposed framework, as it formally links collusion resistance to the threshold parameter.

Definition 6. (*Threshold Attestation Unforgeability Experiment*). *Let the protocol $\Pi_{\text{coll-min}} = (\text{DVRF}, \text{dx-DCTLs}, \text{TSS})$ defined in the previous sections. We say that $\Pi_{\text{coll-min}}$ satisfies threshold attestation unforgeability if no probabilistic polynomial-time adversary \mathcal{A}_{TAU} can succeed in the following experiment with more than negligible probability.*

- \mathcal{A}_{TAU} corrupts the prover P , the coordinator verifier V_{coord} , and a subset of auxiliary verifiers indexed by a corruption set $C \subseteq \{0, 1, \dots, n-1\}$ with $|C| \leq t-1$. The challenger \mathcal{C}_{TAU} controls the honest auxiliary verifiers in $V \setminus C$.
- \mathcal{A}_{TAU} and \mathcal{C}_{TAU} jointly execute the distributed key generation and DVRF partial evaluation protocols on input α , resulting in randomness s_v , a corresponding public key pk , and a verification proof π_{DVRF} .
- \mathcal{A}_{TAU} executes the dx-DCTLs handshake subprotocol HSP using s_p and obtains (sp_s, sp_p, sp_v) together with a handshake proof π_{HSP} .
- \mathcal{A}_{TAU} queries the server oracle $\mathcal{O}^{\text{Server}}$ with (sp_s, sp_p, sp_v) and receives the query and response tuples (Q, R, \hat{Q}, \hat{R}) .
- \mathcal{A}_{TAU} may query the key reveal oracle $\mathcal{O}^{\text{key-reveal}}$ to obtain secret key shares, or the partial evaluation reveal oracle $\mathcal{O}^{\text{partial-eval-reveal}}$ to obtain DVRF partial evaluations, subject to the restrictions of the experiment.

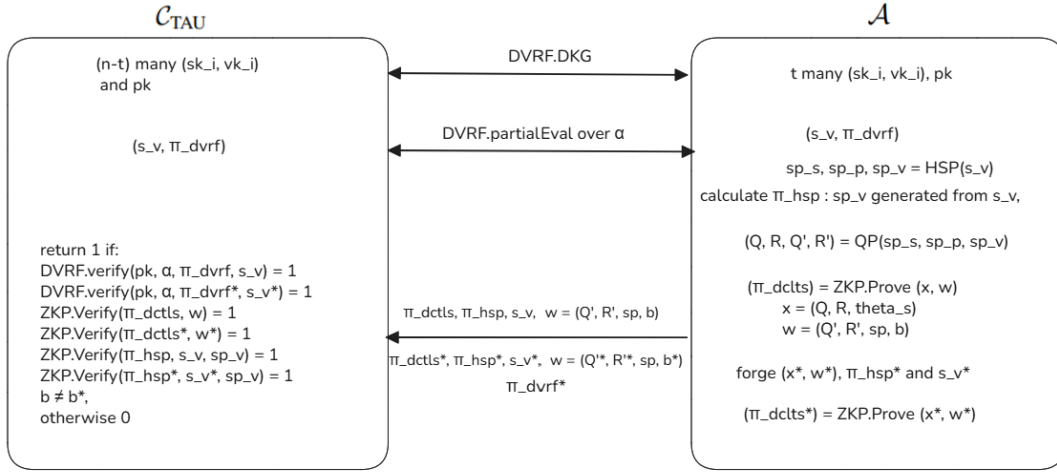


Figure 5.3: Threshold Attestation Unforgeability Experiment [49]

- Using the transcript $x = (\hat{Q}, \hat{R}, sp, b)$, $w = (Q, R, \theta_s)$, along with s_v , π_{DVERF} , and π_{HSP} , the adversary computes a proof $\pi_{\text{dx-DCTLS}} \leftarrow \text{ZKP.Prove}(x, w)$.
- The adversary forges an alternative transcript (x^*, w^*) with corresponding randomness s_v^* , handshake proof π_{HSP}^* , and DVERF proof π_{DVERF}^* , and computes $\pi_{\text{dx-DCTLS}}^* \leftarrow \text{ZKP.Prove}(x^*, w^*)$. Both transcripts are submitted to the challenger.
- The experiment outputs 1 if all verification checks succeed for both transcripts, while the derived statements satisfy $b \neq b^*$. Otherwise, the experiment outputs 0.

Definition 7. (Threshold Attestation Unforgeability). A protocol Π provides threshold attestation unforgeability if for all probabilistic polynomial-time adversaries \mathcal{A} and security parameters λ ,

$$\Pr [Adv_{\Pi}^{\text{TAU}}(\mathcal{A}, \lambda) = 1] \leq \text{negl}(\lambda).$$

5.4.2 Adversarial Oracles

The adversary \mathcal{A} is granted access to the following oracles, which model its adaptive interactions with honest parties in the threshold attestation unforgeability experiment and the subsequent security definitions. These oracles allow \mathcal{A} to query protocol com-

ponents, obtain partial information, and simulate realistic attack capabilities under the specified threat model.

- $\mathcal{O}^{\text{key-reveal}}$: Provides secret key material associated with threshold components of the protocol, subject to experiment restrictions.
- $\mathcal{O}^{\text{partial-eval-reveal}}$: Returns secret partial evaluations from the DVRF session.
- $\mathcal{O}^{\text{Server}}$: Simulates the TLS server by deterministically responding to encrypted queries \hat{Q} with encrypted responses \hat{R} .
- \mathcal{O}^{ro} : Models a random oracle. Upon receiving a query x , the oracle returns a consistent value, sampling uniformly at random if x has not been queried before.

The protocol preserves privacy by construction, inherited from DCTLS. The verifier never obtains the prover’s decryption key and therefore cannot learn the plaintext query response pair (Q, R) as private input, as shown in Definition 4. The security model explicitly captures malicious behavior by both the prover and verifiers, including attempts to forge attestations without a valid session secret θ_s or to deviate by revealing inconsistent key material after commitment. Even in the case where the prover and the verifier act maliciously in concert, the probability of producing an invalid proof remains negligible due to the soundness property of the zero-knowledge proof system according to the Definition 3.

5.5 Strawman Protocols

This section examines two preliminary strawman protocol designs that provide intuitive but insufficient approaches to collusion resistance in DCTLS-based TLS attestations. These designs are presented to clarify the design space and to motivate the architectural choices made in dx-DCTLS. Each strawman is shown to fail either in efficiency, verifiability, or robustness under adversarial behavior.

5.5.1 nPC-based Strawman Protocol

Existing DCTLS constructions rely on two-party computation, where the prover and a single verifier jointly emulate a TLS client. A natural but naive extension attempts to replace this two-party interaction with an n -party computation involving multiple verifiers, with the goal of reducing collusion risk. The resulting strawman protocol generalizes the DCTLS interaction to an n -verifier setting, as illustrated in Figure 5.5.

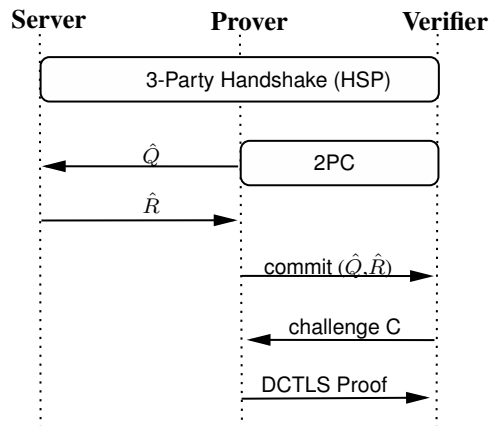


Figure 5.4: General Overview of DCTLS Protocols with Server, Prover, and Verifier

The protocol proceeds in three phases.

- **Handshake Phase.** The server S , the prover P , and n verifiers jointly execute a multi-party handshake. The prover and the verifiers collectively act as a single TLS client and jointly maintain the session key material.
- **Query Phase.** The prover and the n verifiers jointly construct a TLS query using n -party computation. The prover receives the encrypted server response and commits it to the verifiers.
- **Proving Phase.** The prover generates a proof that selectively reveals relevant parts of the server response while hiding sensitive information, allowing the verifiers to validate the attested statement.

Limitations. While this construction reduces collusion risk by requiring agreement among multiple verifiers, it introduces substantial inefficiencies. Replacing two-party computation with n -party computation significantly increases communication complexity and coordination overhead. Moreover, the design lacks modularity, as all

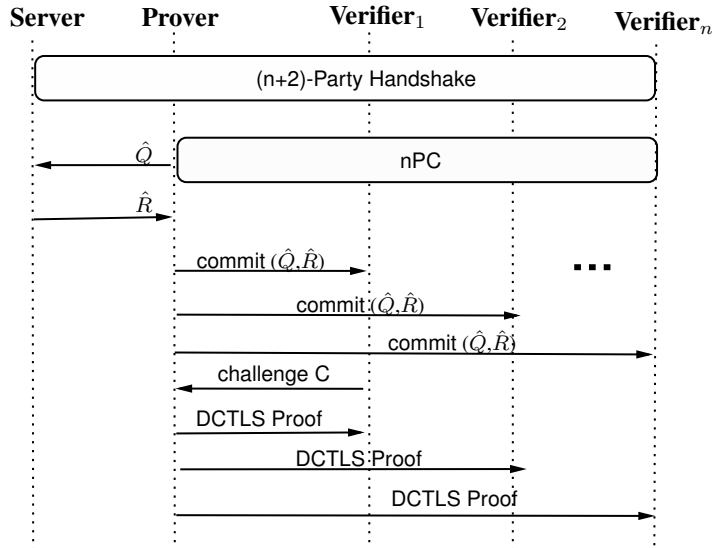


Figure 5.5: nPC-Based Strawman Protocol Extending to an n -Verifier Setting [49]

verifiers must remain online and synchronized throughout the protocol execution. These properties make the construction impractical for deployment in decentralized and resource-constrained environments.

More importantly, the same security objective, namely collusion-resistant and publicly verifiable TLS attestations, can be achieved without incurring the full cost of n -party computation by combining threshold cryptographic primitives with a single interactive DCTLS execution. This observation motivates the separation of interaction and verification roles adopted in dx-DCTLS.

5.5.2 DVRF-Integrated DCTLS Strawman Protocol

The second strawman protocol combines a distributed verifiable random function, a standard DCTLS execution, and a threshold signature scheme. The intended goal is to bind the TLS attestation to jointly generated randomness while allowing multiple verifiers to approve the result.

The protocol is abstractly described as $\Pi = (\text{DVRF}, \text{DCTLS}, \text{Signing})$ and described below.

- **DVRF Phase:** All verifiers jointly agree on an input α and execute the distributed key generation protocol. Each verifier computes a partial evaluation,

and the coordinator verifier combines these evaluations to obtain the final randomness output s_v .

- **DCTLS Phase:** The prover, server, and coordinator verifier execute a standard DCTLS protocol. The coordinator verifier uses the DVRF-derived randomness s_v in place of locally generated randomness and forwards protocol messages to auxiliary verifiers for transparency.
- **Signing Phase:** Each auxiliary verifier checks whether the DCTLS execution appears consistent with the shared randomness s_v . If satisfied, the verifier participates in a threshold signing protocol to approve the attestation.

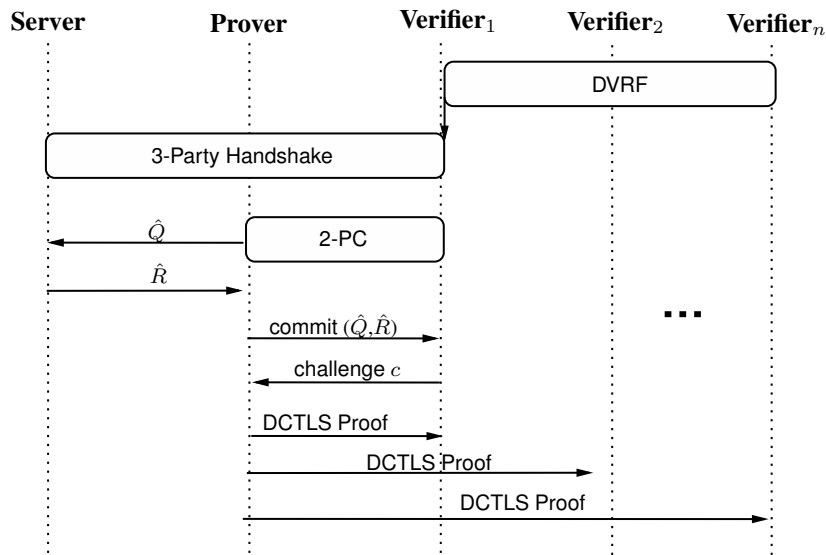


Figure 5.6: DVRF-Integrated Strawman Protocol Extending to an n -Verifier Setting [49]

Limitations. This construction fails to provide the required unforgeability guarantees due to the lack of exportable and verifiable handshake proofs in standard DCTLS. Although auxiliary verifiers can reconstruct the expected pre-master secret using the shared randomness, they cannot independently verify that the session key shares used by the coordinator verifier were correctly derived from this value. The two-party computation used during key derivation does not provide verifiability to third parties.

As a result, an auxiliary verifier cannot reliably determine whether the attestation is bound to the current session or to a different but valid DCTLS execution. An adversary can therefore substitute a distinct, well-formed DCTLS transcript originating

from another session while still producing a proof that verifies successfully. This breaks the intended binding between the attestation and the jointly generated randomness and violates the unforgeability requirement.

These limitations demonstrate that simply combining DVRF with standard DCTLS is insufficient. Explicit verifiability of the handshake phase is required in order to securely bind attestations to shared randomness, which directly motivates the design of dx-DCTLS.

5.6 Collusion-Minimized TLS Attestation Protocol

This section presents the collusion-minimized TLS attestation protocol named as $\Pi_{\text{coll-min}}$, which is designed to remain secure in the presence of collusion between the prover and verifiers. The overall structure of the protocol is illustrated in Figure 5.8. Building on the abstractions and security requirements introduced in earlier chapters, this section first formalizes the dx-DCTLS abstraction and then presents the full protocol construction obtained by combining distributed randomness generation, exportable TLS attestations, and threshold signatures. Finally, the section establishes the security of the construction under the threshold attestation unforgeability notion.

5.6.1 The dx-DCTLS Abstraction: From DCTLS to dx-DCTLS

At a high level, the protocol $\Pi_{\text{coll-min}}$ relies on a variant of DCTLS that enables external and third-party verifiability of the interaction between the prover P and the coordinator verifier V_{coord} . Existing DCTLS constructions, including DECO and Distefano, achieve privacy-preserving TLS attestations but rely on the non-exportability of protocol transcripts. As a consequence, only the designated verifier participating in the protocol can validate the correctness of an attestation, which fundamentally prevents joint verification by a set of verifiers.

To address this limitation, we define a designed exportable variant of DCTLS, referred to as dx-DCTLS. The core idea is to replace non-verifiable two-party computation steps with verifiable mechanisms, such as verifiable two-party computation

or co-SNARK-based proofs, that allow the verifier to demonstrate that the DCTLS execution was carried out using designated randomness rather than locally sampled values. The dx-DCTLS abstraction retains the same interaction structure as standard DCTLS and consists of the subprotocols HSP, QP, and PGP, involving the server S, the prover P, and a verifier V, as specified in Section 5.3. This abstraction by defining the syntax of the Setup, Randomness Creation, Attestation, and Signing algorithms, together specifying how distributed randomness is generated, how dx-DCTLS produces an exportable attestation. The Setup algorithm initializes public parameters and threshold-related metadata for the verifier set. The Randomness Creation algorithm generates a unique and publicly verifiable randomness value using a distributed verifiable random function. The Attestation algorithm executes the dx-DCTLS protocol under the agreed randomness and outputs an attested statement together with an exportable proof. Finally, the Signing algorithm aggregates verifier approvals into a threshold signature that can be validated by a smart contract.

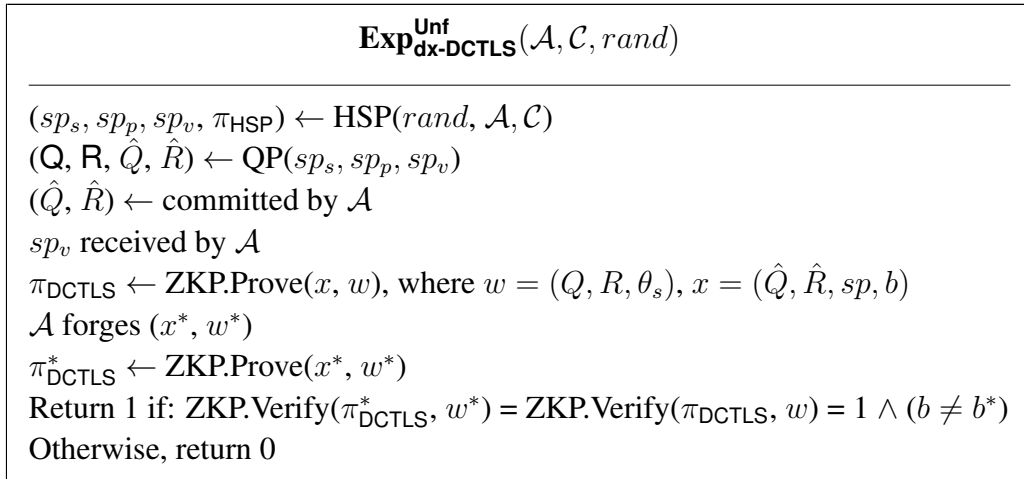


Figure 5.7: Unforgeability Experiment of dx-DCTLS [49]

As in standard DCTLS, the verifier in dx-DCTLS is convinced that the attested statement b originates from a valid TLS session with the server. In addition, dx-DCTLS ensures that this conviction is obtained in an exportable form. In particular, the verifier can present verifiable evidence that the attestation was executed with designated randomness $rand$ by revealing $rand$ together with the handshake proof π_{HSP} . This enables third parties to independently validate the correctness of the execution with integrity guarantees. The corresponding unforgeability experiment for dx-DCTLS is formalized in Figure 5.7.

5.6.2 The Full Protocol $\Pi_{\text{coll-min}}$

The complete collusion-minimized TLS attestation protocol is obtained by combining distributed randomness generation, the dx-DCTLS abstraction, and a threshold signature scheme. The resulting protocol is denoted by $\Pi_{\text{coll-min}} = (\text{DVRF}, \text{dx-DCTLS}, \text{TSS})$ and involves a server S , a prover P , a smart contract SC , a coordinator verifier V_{coord} , and a set of auxiliary verifiers $\{V_i\}$. The overall interaction pattern is shown in Figure 5.8, which provides a high-level overview of the complete interaction pattern of $\Pi_{\text{coll-min}}$, where the orange-colored text is included as explanatory comments for clarity. The protocol proceeds in three main phases. First, the coordinator verifier and auxiliary verifiers jointly generate public randomness using a DVRF through a distributed key generation and combination process. This randomness is then used to parameterize the attestation phase, in which the server, prover, and coordinator verifier execute the dx-DCTLS protocol and produce zero-knowledge proofs attesting to the correctness of the TLS-derived statement. Finally, auxiliary verifiers validate the proofs and collectively generate a threshold signature over the attested statement, which is verified by the smart contract before being accepted by the decentralized application.

The unforgeability experiment shown in Figure 5.7 formalizes the security requirement that no PPT adversary can produce a valid attestation for a statement that was not derived from a correctly executed TLS session. This experiment extends classical DCTLS unforgeability to settings in which verification is distributed across multiple parties. It therefore provides the foundation for proving that dx-DCTLS preserves correctness while enabling public and joint verification.

The central security objective of this construction is threshold attestation unforgeability (TAU), which captures resistance against collusion between the prover and up to $t - 1$ verifiers. The corresponding experiment is defined in Figure 5.3.

Threshold Attestation Unforgeability (TAU) experiment:

Let's assume the protocol $\Pi_{\text{coll-min}} = (\text{DVRF}, \text{dx-DCTLS}, \text{TSS})$ defined above. We call that $\Pi_{\text{coll-min}}$ satisfies threshold attestation unforgeability if, for all probabilistic polynomial-time adversaries \mathcal{A}_{TAU} , the adversary wins the experiment shown in Fig-

$$\Pi_{\text{coll-min}}$$

- **Setup**(1^λ) \rightarrow (pp) $\backslash\backslash$ pp includes security parameters and defining plaintext α for the creating randomness for all parties.
- **Random Creation Phase** $\backslash\backslash$ Among V_{coord} and V_i s
 - $(sk_i, vk_i, pk) \leftarrow \text{DKG}(pp, t, n)$
 - $(i, v_i, \pi_i^{\text{DVRF}}) \leftarrow \text{PartialEval}(\alpha, sk_i, vk_i)$
 - $(rand, \pi^{\text{DVRF}}) \leftarrow \text{Combine}(pk, VK, \alpha, E)$
- **Attestation Phase** $\backslash\backslash$ **dx-DCTLS** runs among server **S**, prover **P** and coordinator verifier V_{coord}
 - (**S**, **P**, V_{coord}) gets $(sp_s, sp_p, sp_v) \leftarrow \text{HSP}(pp, rand)$
 V_{coord} gets $\pi_{\text{HSP}} \leftarrow \text{ZKP.Prove}(sp_v, rand)$
 - **P** gets $(Q, P, \hat{Q}, \hat{R}) \leftarrow \text{QP}(sp_s, sp_p, sp_v)$
 - **P** calculates $\pi_{\text{dx-DCTLS}} \leftarrow \text{ZKP.Prove}(x, w)$: private input $x = (Q, R, \theta_s)$, public input $w = (\hat{Q}, \hat{R}, sp_v, b)$, then sends it to the V_{coord} .
 - V_{coord} broadcasts the $\pi_{\text{dx-DCTLS}}$ with w and pre-calculated π_{HSP} to the V_i .
- **Signing Phase** $\backslash\backslash$ **TSS** runs among V_{coord} and V_i s
 - $\{0, 1\} \leftarrow \text{ZKP.Verify}(\pi_{\text{DCTLS}}, w)$
 - $\{0, 1\} \leftarrow \text{DVRF.Verify}(pk, \alpha, \pi_{\text{DVRF}}, sp_v)$
 - $\{0, 1\} \leftarrow \text{ZKP.Verify}(\pi_{\text{HSP}}, rand)$, for each V_i
 - If it returns 1, each V_i loads (sk_i, vk_i, pk)
 - $\sigma_i \leftarrow \text{Sign}(b, sk_i)$
 - V_{coord} gets $\sigma \leftarrow \text{Combine}(\sigma_i)$, sends it to **SC**
 - $\backslash\backslash$ **The SC verifies the signature**
 - $\{0, 1\} \leftarrow \text{SC.Verify}(\sigma, pk)$
 - If it returns 1, SC accepts the b for the decentralized application; otherwise aborts the session.

Figure 5.8: The Proposed Collusion-Minimized Protocol $\Pi_{\text{coll-min}}$ Construction [49]

ure 5.3 with only negligible probability.

Theorem 1. (*Collision minimization*). *If the distributed verifiable random function satisfies the uniqueness property and dx-DCTLS satisfies unforgeability, then the protocol $\Pi_{\text{coll-min}}$ achieves threshold attestation unforgeability. In particular,*

$$\text{Adv}_{\mathcal{A}, \Pi_{\text{coll-min}}}^{\text{TAU}} \leq \text{Adv}_{\mathcal{B}_{\mathcal{D}}, \text{DVRF}}^{\text{Uniq}} + \text{Adv}_{\mathcal{B}_{\mathcal{X}}, \text{dx-DCTLS}}^{\text{Unf}}$$

Proof. Let X be the event that \mathcal{A} wins TAU game with the probability $\Pr[X]$ represented as $\text{Adv}_{\mathcal{A}, \Pi_{\text{coll-min}}}^{\text{TAU}}$. In event X , the \mathcal{A} successfully forges another valid dx-DCTLS transcript (x^*, w^*) with corresponding s_v^* and its proofs π_{HSP}^* and π_{DVRF}^* . If \mathcal{A} is able to produce this transcript, then one of the two things must happen:

1. Attacker \mathcal{A} wins the DVRF uniqueness game and uses another valid transcript that does not belong to the original session.
2. DVRF uniqueness property is held.

We define two additional events based on these happenings above:

Let Y denote the event that \mathcal{A} wins the DVRF uniqueness game by forging another valid randomness s_v^* that is $\text{DVRF.Verify}(pk, \pi_{\text{DCTLS}}, \alpha, s_v^*)$.

Let Z denote the event that \mathcal{A} wins TAU unforgeability game by forging another valid transcript without Y occurring, which implies that \mathcal{A} must break the dx-DCTLS game.

$$\begin{aligned} \text{Adv}_{\mathcal{A}, \Pi_{\text{coll-min}}}^{\text{TAU}} &= \Pr[X] \\ &\leq \Pr[X \wedge \neg Y] + \Pr[Y] \\ &= \Pr[Z] + \Pr[Y] \\ &= \text{Adv}_{\mathcal{B}_{\mathcal{D}}, \text{DVRF}}^{\text{Uniq}} + \text{Adv}_{\mathcal{B}_{\mathcal{X}}, \text{dx-DCTLS}}^{\text{Unf}} \end{aligned}$$

To prove the theorem, we construct a DVRF adversary $\mathcal{B}_{\mathcal{D}}$ and a dx-DCTLS adversary $\mathcal{B}_{\mathcal{X}}$.

First, we construct $\mathcal{B}_{\mathcal{D}}$ who plays DVRF uniqueness game with challenger $C_{\mathcal{D}}$ and runs \mathcal{A} as subroutine as shown in Figure 5.9 as follows:

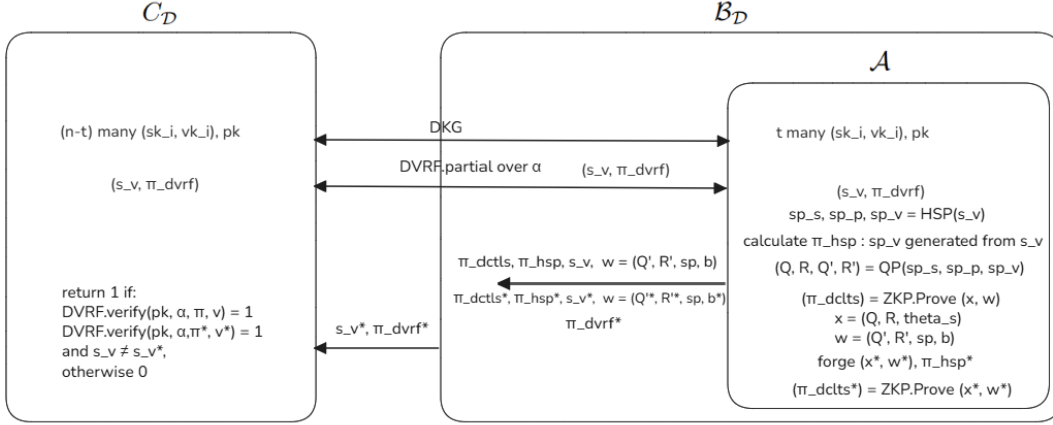


Figure 5.9: Building DVRF Uniqueness Adversary \mathcal{B}_D [49]

- \mathcal{B}_D runs DKG with C_D including \mathcal{A} , they hold t and $(n - t)$ many key shares as (sk_i, vk_i) respectively and a pk .
- \mathcal{B}_D runs DVRF.PartialEval over the α with C_D including \mathcal{A} , and gets (s_v, π_{DVRF}) .
- \mathcal{A} runs dx-DCTLS.HSP with s_v and dx-DCTLS.QP to obtain (Q, P, \hat{Q}, \hat{R}) .
- \mathcal{A} uses the transcript $(x = (\hat{Q}, \hat{R}, sp, b), w = (Q, R, \theta_s), s_v, \pi_{\text{DRVF}}, \pi_{\text{HSP}})$, and calculates $\pi_{\text{dx-DCTLS}}$ which is $\text{ZKP.Prove}(x, w)$.
- \mathcal{A} forges another dx-DCTLS transcript (x^*, w^*) , and s_v^* with its π_{HSP}^* and valid π_{DVRF}^* to calculate $\pi_{\text{dx-DCTLS}}^* = \text{ZKP.Prove}(x^*, w^*)$. Sends both to the \mathcal{B}_D .
- \mathcal{B}_D outputs $(s_v^*, \pi_{\text{DVRF}}^*)$ to the C_D .

$$Pr[Y] = \text{Adv}_{\mathcal{B}_D, \text{DVRF}}^{\text{Uniq}}$$

Finally, we construct \mathcal{B}_X who plays dx-DCTLS unforgeability game with challenger C_X and again runs \mathcal{A} as a subroutine as shown in Figure 5.10:

- \mathcal{B}_X and C_X pick randoms, s_p , and s_v .
- \mathcal{B}_X and C_X run dx-DCTLS.HSP with (s_p, s_v) , then obtain $(sp_s, sp_v, \pi_{\text{HSP}})$, and (sp_p) respectively.
- \mathcal{B}_X runs dx-DCTLS.QP with (sp_s, sp_v, sp_p) and gets (Q, P, \hat{Q}, \hat{R}) .
- \mathcal{B}_X initializes DVRF.DKG with \mathcal{A} and both of them get the secret key shares (sk_i, vk_i) , and DVRF.PartialEval output over α .

- \mathcal{A} receives the identical s_v by querying the \mathcal{O}^{Hash} .
- \mathcal{A} runs dx-DCTLS.HSP with random sampling s_p , which $\mathcal{B}_{\mathcal{X}}$ writes s_p to random tape.
- \mathcal{A} calculates $(sp_s, sp_v, \pi_{\text{HSP}})$, (sp_p) , and gets $(\mathbf{Q}, \mathbf{P}, \hat{Q}, \hat{R})$.
- \mathcal{A} uses the transcript $(x = (\hat{Q}, \hat{R}, sp, b), w = (\mathbf{Q}, \mathbf{R}, \theta_s), s_v, \pi_{\text{DRVF}}, \pi_{\text{HSP}})$, and calculates $\pi_{\text{dx-DCTLS}}$ which is $\text{ZKP.Prove}(x, w)$.
- \mathcal{A} forges another dx-DCTLS transcript (x^*, w^*) and s_v^* with its π_{HSP}^* , and a valid π_{DRVF}^* to calculate $\pi_{\text{dx-DCTLS}}^* = \text{ZKP.Prove}(x^*, w^*)$. Sends both to $\mathcal{B}_{\mathcal{X}}$.
- $\mathcal{B}_{\mathcal{X}}$ outputs $(\pi_{\text{dx-DCTLS}}, w)$ and $(\pi_{\text{dx-DCTLS}}^*, w^*)$ to $C_{\mathcal{X}}$.

$$Pr[Z] = \text{Adv}_{\mathcal{B}_{\mathcal{X}}, \text{dx-DCTLS}}^{\text{Unf}}$$

Finally,

$$\text{Adv}_{\mathcal{A}, \Pi_{\text{coll-min}}}^{\text{TAU}} \leq Pr[Z] + Pr[Y] = \text{Adv}_{\mathcal{B}_{\mathcal{D}}, \text{DVRF}}^{\text{Uniq}} + \text{Adv}_{\mathcal{B}_{\mathcal{X}}, \text{dx-DCTLS}}^{\text{Unf}}.$$

□

The above result of Theorem 1 formalizes threshold attestation unforgeability in multi-verifier settings and captures the adversarial capabilities relevant to the proposed framework as shown in Definition 7. This characterization provides a sound basis for reasoning about collusion resistance and correctness in the constructions that follow. Therefore, even if the prover P colludes with the coordinator verifier V_{coord} and learns the TLS session secret sp , any attestation remains bound to the prover's own TLS session with the claimed server due to the binding property of sp . Knowledge of sp enables the colluding parties to generate internally consistent ciphertexts and corresponding plaintext claims, but it does not allow them to construct attestations for other users' sessions or for interactions with different servers. All attested statements must remain consistent with the specific TLS session established between the prover and the server. As in existing DCTLS constructions such as DECO, commit-then-prove mechanisms ensure that attestations are cryptographically tied to authentic session transcripts, but they do not prevent a prover from choosing which valid request–response pairs within a single TLS session to disclose. Preventing such

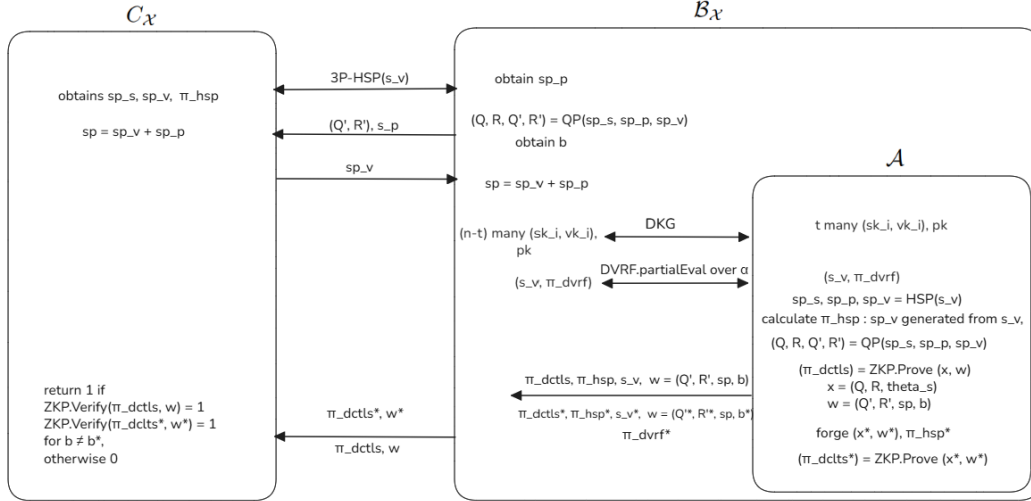


Figure 5.10: Building dx-DCTLS Unforgeability Adversary \mathcal{B}_X [49]

selective disclosure would require additional transcript-locking or commitment mechanisms, which are orthogonal to the design goals of DCTLS protocols and are outside the scope of this thesis.

5.7 Instantiating dx-DCTLS from DECO and Distefano

This subsection explains how the dx-DCTLS abstraction can be instantiated from existing DCTLS protocols, specifically DECO and Distefano. While neither protocol natively satisfies the exportability requirements imposed by dx-DCTLS, both can be adapted by replacing non-verifiable cryptographic components with verifiable counterparts. Despite protocol-specific differences, both instantiations admit a unified view of handshake verifiability.

In both cases, the correctness of the handshake is validated by auxiliary verifiers through a common verification interface:

$$\{0, 1\} \leftarrow \text{ZKP.Verify}(\pi_{\text{HSP}}, rand),$$

where π_{HSP} serves as an exportable certificate binding the TLS session to the distributed randomness $rand$. This abstraction allows the verifiable handshake to be treated as a modular component, independent of the underlying DCTLS instantiation.

5.7.1 Modified DECO as dx-DCTLS.

The DECO protocol closely follows the DCTLS architecture, where the tuple (sp_p, sp_v, sp) corresponds to the TLS 1.2 MAC keys $(K_P^{\text{MAC}}, K_V^{\text{MAC}}, K^{\text{MAC}})$. The required modification affects the handshake subprotocol (HSP), in which the TLS-PRF two-party computation is replaced by a co-SNARK.

In the resulting construction, the prover P provides the MAC key share K_P^{MAC} , while the verifier V provides K_V^{MAC} . The verifiable handshake is formalized as:

$$(K^{\text{MAC}}, \pi_{\text{HSP}}) \leftarrow \text{co-SNARK.Execute}(\{K_P^{\text{MAC}}, K_V^{\text{MAC}}\}, Z_p)$$

where Z_p denotes the pre-master secret known to the auxiliary verifiers. This procedure allows the verifier to reconstruct the MAC key together with a proof π_{HSP} attesting that the derivation was performed using the designated randomness.

To achieve exportability, the verifier distributes the reconstructed MAC key K^{MAC} along with π_{HSP} to the auxiliary verifiers. The proof enables independent verification of the correctness of K^{MAC} without requiring participation in the underlying DECO execution. As a result, the handshake phase becomes both exportable and jointly verifiable, which is essential for the dx-DCTLS abstraction.

Security of this phase follows directly from the security of the underlying cryptographic primitives. Any deviation from a correctly derived K^{MAC} would require either forging a valid proof π_{HSP} , contradicting the soundness of the co-SNARK construction, or producing a valid transcript without a corresponding DECO execution, which violates the unforgeability guarantees of the original protocol.

5.7.2 Modified Distefano as dx-DCTLS.

The Distefano protocol is structurally aligned with DCTLS, where the tuple (sp_p, sp_v, sp) corresponds to the TLS 1.3 traffic secrets. Unlike the TLS 1.2 setting, the use of co-SNARKs is not applicable here, since TLS 1.3 does not separate message authentication from encryption. Because authentication is integrated into the AEAD construction, revealing the jointly derived traffic keys would fundamentally compromise confidentiality.

Consequently, the sum of the key shares (sp_p, sp_v) is never revealed to the coordinator verifier or to auxiliary verifiers in the Distefano construction. While standard two-party computation preserves this confidentiality, its lack of public verifiability prevents it from directly satisfying the dx-DCTLS requirements. To address this limitation, the Distefano instantiation replaces all two-party computations with verifiable two-party computation (v2PC) primitives. The resulting verifiable handshake is formalized as:

$$(sp_p, sp_v, \pi_{\text{HSP}}) \leftarrow \text{v2PC.Execute}(s_p, s_v).$$

For clarity, the tuple (sp_p, sp_v) denotes the aggregate of traffic secret shares derived across the multi-stage Distefano handshake. The associated proof π_{HSP} attests that these shares were correctly derived from the DVRF output, ensuring that the entire process remains bound to the distributed randomness.

As in the DECO-based instantiation, exportability is achieved by distributing the traffic secret shares (sp_p, sp_v) together with the corresponding verifiable handshake proofs to the auxiliary verifiers. These proofs enable independent verification of correctness without disclosing the underlying secrets or requiring participation in the interactive handshake.

Security again follows from reduction to the underlying primitives. Any deviation from a correctly derived traffic secret tuple would require either forging a valid v2PC proof π_{HSP} , contradicting the soundness of the verifiable two-party computation, or generating a valid session transcript without a corresponding Distefano execution, which violates the unforgeability guarantees of the original protocol.

5.8 Performance Evaluations

In this section, we present our prototype implementation and provide a performance evaluation. The results indicate that our modifications do not introduce substantial overhead compared to existing DCTLS protocols.

We describe how a distributed key generation (DKG), a distributed verifiable random function (DVRF), and a threshold signature scheme (TSS) can be combined in practice, with the additional requirement that the final threshold signature must

be verifiable on the Ethereum Virtual Machine (EVM). We first present the design space and possible alternatives, then highlight incompatibilities across some settings, and finally identify two compatible reference instantiations: one based on DDH-style DVRFs and Schnorr/FROST[32] signatures, and another based on Glow-style DVRFs and BLS signatures. The results confirm that increasing the verifier set does not introduce prohibitive overhead, which is essential for effective collusion minimization in large-scale deployments Figure 5.11.

Our performance evaluation proceeds in two phases: the DVRF-TSS extension and the dx-DCTLS component. It is important to note that we do not provide a full end-to-end implementation of dx-DCTLS. Instead, we measure the additional overhead that dx-DCTLS would introduce by calculating its cost relative to DECO.

Our experiments were conducted on a machine equipped with an M3 processor and 16 GB of RAM. The implementation relies on the *frost-secp256k1-vm* and *K256* Rust crates for cryptographic primitives. For distributed key generation, the JF-DKG implementation provided by *frost-secp256k1-vm* was used, while the DDH-based DVRF construction [18] was implemented as part of this work. Performance benchmarks were implemented using the *criterion* crate, which executes multiple iterations of each experiment and reports average execution times.

DVRF-TSS extension. We implement our prototype over secp256k1 together with FROST, although an alternative instantiation based on Glow-style DVRFs and threshold BLS signatures over BN254 would also be feasible. While both instantiation families are algebraically sound, they differ substantially in deployment complexity and verification cost. We select the secp256k1 and FROST stack because the DDH-based DVRF and FROST signatures operate in the same prime-order group and can reuse a single distributed key generation (DKG) instance. This design choice simplifies the overall architecture and reduces the number of required setup phases. In contrast, Glow and other pairing-based DVRFs rely on bilinear groups, require pairing-friendly DKG protocols, and cannot reuse Schnorr or FROST keys. Although the Glow plus BLS alternative is theoretically valid, the secp256k1-based construction better aligns with cryptographic primitives already supported by the EVM and incurs significantly lower on-chain verification overhead. For these reasons, secp256k1 and FROST con-

stitute the more practical choice for an initial implementation.

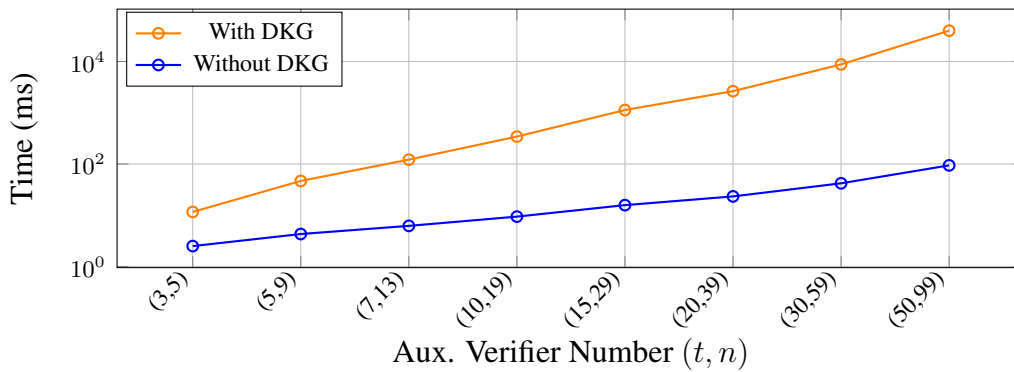


Figure 5.11: LAN Performance Comparison Showing the Execution Time of the DVRF-TSS Extension, Including the Initial DKG Phase Followed by DVRF-Based Randomness Generation and Threshold Signature (TSS) Creation, for Varying t -out-of- n Auxiliary Verifier Configurations [49]

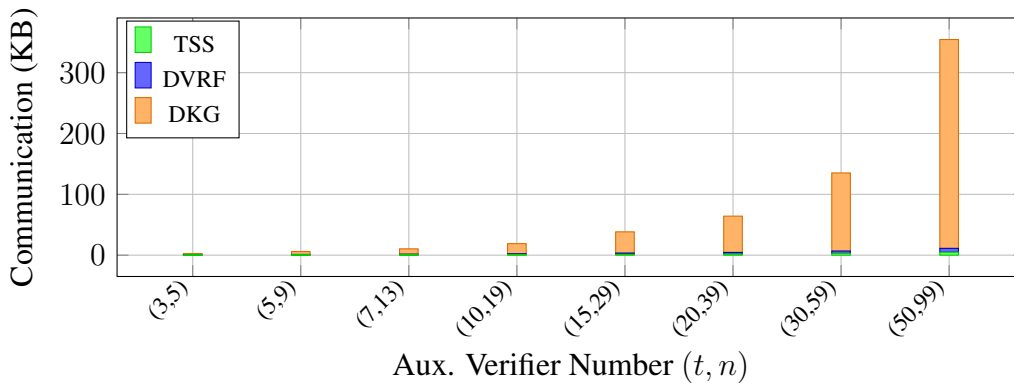


Figure 5.12: Network Communication Cost of DVRF-TSS, Including DKG [49]

The experimental evaluation in [47] is conducted under both LAN and WAN settings for a range of t -out-of- n threshold configurations, with and without the DKG phase. In LAN settings, experiments are performed without simulated network delay, and the resulting execution times are reported in Figure 5.11. These results demonstrate that the DVRF-TSS extension remains feasible even at higher threshold sizes, which is a critical requirement for security-sensitive attestations.

To assess performance under realistic wide-area conditions, additional experiments are conducted under two simulated WAN profiles. The first profile (WAN1) models moderate network conditions with a one-way latency of $40 \text{ ms} \pm 5 \text{ ms}$ ($\text{RTT} \approx 80 \text{ ms}$), bandwidth of 50 Mbps, and packet loss of 0.1%. The second profile (WAN2) represents more adverse conditions with a one-way latency of $75 \text{ ms} \pm 15 \text{ ms}$ ($\text{RTT} \approx 150 \text{ ms}$), bandwidth of 20 Mbps, and packet loss of 0.2%. Packet loss effects are

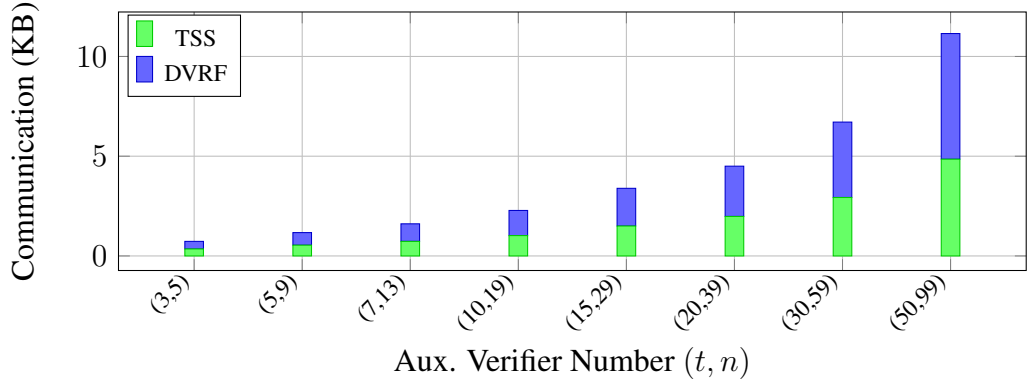


Figure 5.13: Network Communication Cost of DVRF-TSS, Excluding DKG [49]

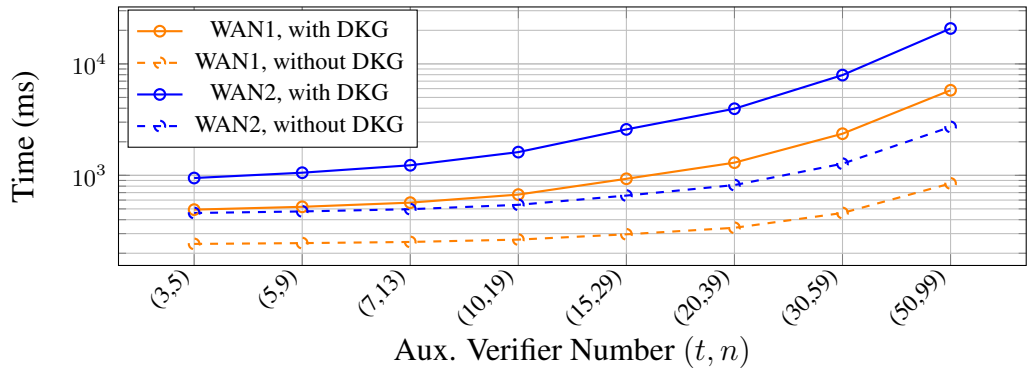


Figure 5.14: Execution Time of the DVRF-TSS under Two WAN Profiles (WAN1 and WAN2), Where Color Represents the WAN Profile and Line Style Distinguishes Executions With and Without the DKG Phase, for Varying t -out-of- n Auxiliary Verifier Configurations [49]

emulated by introducing additional RTT penalties upon loss events. The resulting WAN execution times are summarized in Figure 5.14.

In addition to execution time, we evaluate network communication costs both with and without the DKG phase. The communication overhead with DKG is reported in Figure 5.12, while the no-DKG case is shown in Figure 5.13. As expected, DKG dominates the overall communication cost due to its $\mathcal{O}(n^2)$ complexity. Nevertheless, even under the more demanding WAN2 conditions, the additional overhead introduced by the DVRF-TSS extension remains modest. For example, in the 15-out-of-29 configuration, the protocol incurs approximately one second of additional execution time. Overall, these results indicate that the DVRF-TSS extension scales in a controlled manner and remains practical across realistic network environments, while providing stronger collusion resistance without prohibitive performance costs.

dx-DCTLS. We simulate a DECO-based dx-DCTLS prototype that integrates the DECO protocol with collaborative zk-SNARKs (co-SNARKs). This design choice is practically viable, as prior work shows that co-SNARK constructions significantly reduce the computational cost compared to full n -party computation [37]. In our simulation, we replace only the 2PC-HMAC component in DECO.HSP with a co-SNARK, leaving the remainder of the protocol unchanged. This 2PC-HMAC evaluates the TLS pseudorandom function (TLS-PRF) of TLS 1.2, which requires approximately 60 SHA256 compression operations and translates to 1,719,598 R1CS constraints. According to the experimental results reported in [48], generating a proof for this circuit takes approximately 4.7 seconds.

In addition to proof generation, co-SNARK protocols incur collaborative communication overhead between the participating provers [37]. As a result, the total execution cost depends on the available link capacity between the prover P and the coordinator verifier V_{coord} . Under a high-bandwidth assumption of approximately 3000 Mb/s with two provers, the additional communication overhead is negligible and the co-SNARK computation approaches the cost of a single-prover execution. Under a more conservative setting in which participants operate on resource-constrained devices connected by a 64 Mb/s link, we derive an upper bound of approximately 9.4 seconds for evaluating the TLS-PRF via a co-SNARK.

For comparison, in the original DECO protocol this functionality requires two 2PC-HMAC executions, each taking approximately 5.7 seconds, for a total cost of about 10.4 seconds. In our setting, once the co-SNARK computation completes, the MAC key K^{MAC} becomes visible to both the prover P and the coordinator verifier V_{coord} , which allows the second 2PC-HMAC step in DECO.QP to be safely omitted without weakening security. Consequently, although the co-SNARK component may introduce additional overhead under wide-area network conditions, eliminating one 2PC-HMAC largely compensates for this cost, yielding an end-to-end runtime comparable to that of the original DECO protocol. At this stage, the session and transcript are already cryptographically binding, which further justifies the omission. We note that the high-bandwidth assumption applies only to communication between the prover P and the coordinator verifier V_{coord} , and does not constrain communication among auxiliary verifiers.

A detailed comparison of on-chain verification costs for different threshold signature schemes, including multi-signature, BLS, and the FROST scheme used in our implementation, is provided in our companion work [46]. To avoid redundancy, we omit those tables here and focus instead on off-chain execution costs and the overall feasibility of the proposed design.

While we do not present a full end-to-end implementation of dx-DCTLS, the prototype results and derived upper bounds indicate that the proposed construction is practically feasible. The evaluated overhead remains moderate and scales well with the verifier set size, demonstrating that collusion-minimized TLS attestations can be realized using existing cryptographic primitives and realistic deployment assumptions.

To highlight the architectural advantages of our proposed protocol ($\Pi_{\text{coll-min}}$), we evaluate it against two primary paradigms: the monolithic DECO protocol [56] and a decentralized baseline referred to as **DECO-DON**, the latter being a decentralized oracle network [8] that leverages independent DECO instances for distributed data fetching.

Table 5.1: Comparative Analysis of TLS Attestation Architectures [46]

Feature / Metric	DECO [56]	DECO-DON (Baseline)	$\Pi_{\text{coll-min}}$
Public Verifiability	No	Yes	Yes
Collusion Resistance	No	Yes	Yes
Prover Complexity	$O(1)$	$O(n)$	$O(1)$
Verifier Interaction	Direct	Independent	Collaborative
Auxiliary Node Load	N/A	Heavy	Lightweight
Floating Data Attestation	Yes	No	Yes

The protocol **DECO-DON** is a decentralized baseline architecture, where each verifier independently executes the DECO protocol with the same prover, followed by an off-chain consensus mechanism to finalize the attestation and mitigate collusion risks. As summarized in Table 5.1, while DECO is efficient for single-verifier settings, it lacks public verifiability and is susceptible to prover-verifier collusion. A naive approach to decentralize this, DECO-DON, involves n verifiers executing independent sessions with the prover to reach a consensus. However, this approach imposes a linear computation overhead of $O(n)$ on the prover, making it impractical for large-

scale decentralized networks. In contrast, our construction achieves collusion resistance while maintaining a constant $\mathcal{O}(1)$ prover complexity, regardless of the number of auxiliary verifiers. This is made possible by the exportable nature of **dx-DCTLS**, which allows a single session to be verified by multiple parties. Furthermore, our protocol optimizes the workload of auxiliary verifiers. Unlike DECO-DON, where every participating node must execute the resource-intensive TLS session components, auxiliary nodes in our framework only perform lightweight DVRF and TSS operations. This strategic distribution of computational tasks ensures that our protocol remains scalable and suitable for resource-constrained environments, such as decentralized oracle networks (DONs) and smart contract validators.

The several operational nuances further distinguish $\Pi_{\text{coll-min}}$ from the baselines as summarized in Table 5.1 are as follows:

Auxiliary Node Load (Lightweight vs. Heavy): In DECO-DON, each auxiliary node must bear the full cryptographic cost of a 2PC-based TLS session. In contrast, $\Pi_{\text{coll-min}}$ delegates computationally intensive tasks to a coordinator V_{coord} , enabling auxiliary verifiers to operate using lightweight **DVRF** and **TSS** primitives that are well-suited for resource-constrained environments.

Floating Data Attestation: While the independent sessions in **DECO-DON** struggle to align on highly volatile data due to temporal drift across verifiers, our construction (similar to monolithic DECO) ensures consistency by anchoring the attestation to a single point-in-time TLS session.

5.9 A Use-Case Examples

In this section, we present the proposed protocol $\Pi_{\text{coll-min}}$. The protocol is designed as a general framework for oracle-based decentralized applications that require the following properties: (i) privacy-preserving attestation, (ii) integrity guarantees derived from TLS, and (iii) mechanisms for mitigating collusion among participating parties.

5.9.1 Use Case 1: Confidential Binary Options

Binary options [50] are a fundamental type of financial derivative [30] that allows two parties to speculate on whether a specific condition related to a financial asset will be satisfied at a future time. For instance, the condition may specify whether the price P^* of a stock N on a future date D exceeds a predefined threshold P . In decentralized finance (DeFi), implementing binary options in an on-chain setting is challenging due to the need for trusted and authenticated off-chain data sources, such as asset prices, while simultaneously preserving user privacy. This challenge becomes more pronounced when large financial stakes are involved, as bribery attacks may incentivize collusion, particularly in settings where reliance on a single oracle is insufficient.

Using the proposed protocols, confidential binary options can be executed without disclosing sensitive financial predicates or option parameters to the verifiers. In contrast to prior approaches that rely on a single oracle or trusted execution environments, the proposed design employs a distributed set of verifiers, thereby increasing trustworthiness and resistance to collusion.

The example scheme consists of three main phases: **setup**, **settlement**, and **payout**.

Setup: Alice and Bob agree on the binary option defined by the asset name N , strike price P , and settlement date D . They deploy a smart contract with identifier ID_{SC} that contains commitments to the option parameters C_N , C_P , and C_D , along with their corresponding witnesses r_N , r_P , and r_D . The contract further specifies the agreed verification parameters θ_p , such as the URL from which the asset price is obtained. The verifier set is defined as $V = \{V_{\text{coord}}, V_1, \dots, V_{n-1}\}$, which jointly executes a distributed key generation protocol to derive a shared public key pk and individual signing shares sk_i used for threshold signing.

Settlement: Assuming that Alice wins the bet, she invokes $\Pi_{\text{coll-min}}$, together with the auxiliary verifier set V , in order to securely claim her payout. On the settlement date D , the prover, for example Alice, privately interacts with the price API server S via a TLS session and executes $\Pi_{\text{coll-min}}$ with the verifier set V . The objective is to generate a signed transcript proving that the observed price P^* satisfies the condition $P^* \geq P$

or $P^* < P$, without revealing the exact value of P^* . The signed transcript includes the response $I = (N^*, P^*, D^*)$, and Alice proves in ZKP that

$$P^* \geq P \wedge C_N = \text{com}(N^*) \wedge C_P = \text{com}(P) \wedge C_D = \text{com}(D^*).$$

Upon successful verification, the verifiers jointly produce a threshold signature on the attestation

$$A = \text{hash}(ID_{SC} \parallel \text{Stmt}),$$

where `Stmt` encodes the verified outcome of the binary predicate.

Payout: The signed attestation is submitted to the smart contract. If the attestation is valid, the contract transfers the winning amount to the rightful party, such as Alice. The smart contract verifies the threshold signature using the public key obtained during the distributed key generation phase, ensuring that the attestation has been endorsed by at least t verifiers.

This design provides the following guarantees. First, the verifier set does not learn the exact asset price or other sensitive transaction data. Second, collusion is mitigated, as Alice would need to compromise at least t verifiers, including the coordinating verifier V_{coord} , in order to generate a fraudulent proof. Third, the final payout is determined on chain using an attested binary outcome, without revealing the internal computation or underlying financial data.

In summary, this confidential binary option mechanism demonstrates how the proposed $\Pi_{\text{coll-min}}$ enables the integration of private yet verifiable off-chain data into smart contracts, thereby supporting secure decentralized derivatives with strong privacy and collusion-minimization guarantees.

5.9.2 Use Case 2: Off-Chain Credit and Income Verification

Although most current decentralized finance (DeFi) lending platforms [28], such as Aave, operate exclusively on chain, incorporating off-chain user financial information, including income level or credit score, could substantially expand their lending functionality and enable risk-adjusted lending. This capability is particularly important for users who do not possess significant on-chain collateral but have stable real-world income or verifiable financial standing.

Existing decentralized lending protocols primarily offer over-collateralized loans, which limits participation for underbanked or cash-based users. Integrating $\Pi_{\text{coll-min}}$ enables such protocols to access verified income or credit score information from traditional financial websites in a trust-minimized manner, without requiring users to disclose sensitive personal data. We illustrate the application of the proposed scheme through an income-aware lending decentralized application built on top of Aave. The system enhances lending capabilities without requiring direct integration with banks or centralized credit agencies. It consists of three phases: **setup**, **loan request**, and **repayment**.

Setup: We assume that the Aave smart contract is already deployed with additional logic to verify $\Pi_{\text{coll-min}}$ and to update borrowing parameters accordingly. Let the verifier set be $V = \{V_{\text{coord}}, V_1, \dots, V_{n-1}\}$. Using a distributed key generation protocol as described in the DRVF section, the verifiers jointly derive a public key pk and corresponding secret key shares sk_i , for $i \in \{0, 1, \dots, n-1\}$. A predefined list of acceptable income attestation sources is embedded in the smart contract logic.

Loan request: Suppose that Bob intends to borrow 2000 USDC from Aave with reduced collateral requirements. Bob initiates a zkTLS attestation using the $\Pi_{\text{coll-min}}$ protocol by establishing a TLS session with his payroll provider’s website, acting as the server S . Bob acts as the prover P , while the verifier set includes $V_{\text{coord}} \in V$. Bob privately proves that his monthly income exceeds a required threshold, for example, 3000 USD, based on a TLS response containing the tuple $I = (\text{income}, \text{employment_status}, \text{timestamp})$. Sensitive information, including the exact salary amount or employer identity, remains hidden during the zero-knowledge proving process.

Upon successful completion of the selected protocol, the verifiers jointly produce a threshold signature on the attestation $A = \text{hash}(I)$ using the same distributed key generation configuration. The resulting signature $\text{TSS}_{\text{sign}}(A)$ is submitted to the smart contract, which verifies the signature and dynamically adjusts Bob’s borrowing limit or required collateral ratio.

Repayment: If Bob repays the loan on time, the protocol proceeds identically to the standard Aave repayment process. In the event of default, the smart contract records

the signed attestation on chain. This allows future decentralized applications to reference verified historical income proofs for reputation or trust assessment, without revealing any underlying financial data.

In conclusion, the proposed scheme extends Aave’s lending functionality by incorporating off-chain financial information, such as income or credit scores, in a privacy-preserving and verifiable manner. Since each attestation is verified and co-signed by a threshold set of verifiers, the risk of collusion is minimized. Even if Bob colludes with V_{coord} , he would still need to compromise at least $t - 1$ additional verifiers in order to submit a fraudulent income attestation.

5.10 Summary

This chapter establishes that collusion-minimized TLS attestations can be reasoned about at the level of execution and verification abstractions, rather than being confined to protocol-specific designs. By introducing *dx-DCTLS*, we make explicit the verification logic that was previously embedded within designated verifier executions and show how it can be exported and validated independently. This shift enables a unified treatment of DCTLS protocols across different TLS versions and provides a principled foundation for defining and proving security properties such as external verifiability and unforgeability.

Moreover, the chapter demonstrates that this abstraction is not merely conceptual but can be concretely instantiated and composed with distributed validation mechanisms. By combining *dx-DCTLS* with the DVRF-TSS primitive, we obtain a collusion-minimized TLS attestation protocol with formally defined security guarantees, constant prover complexity, and practical feasibility. The accompanying implementation analysis, comparative evaluation, and representative use cases illustrate that the proposed framework supports both rigorous security reasoning and real-world deployment. Together, these results position *dx-DCTLS* as a reusable and extensible foundation for future work on verifiable TLS attestations in decentralized and adversarial environments.

CHAPTER 6

CONCLUSION

This thesis investigated the problem of producing privacy-preserving, publicly verifiable, and collusion-resistant attestations from TLS sessions for decentralized and smart-contract-based applications. While Designed Commitment TLS (DCTLS) protocols have emerged as a powerful mechanism for extracting authenticated statements from standard TLS connections without modifying server infrastructure, their reliance on a designated verifier fundamentally limits their suitability for adversarial and trust-minimized environments. In particular, the designated verifier assumption creates a structural vulnerability to prover-verifier collusion, undermining public verifiability and the secure on-chain consumption of TLS-based attestations.

The central contribution of this thesis is a systematic rethinking of DCTLS protocols from both a cryptographic and architectural perspective. Rather than treating collusion as an external threat to be mitigated through trusted hardware, oracle networks, or repeated attestations, this work identifies collusion as a consequence of verifier centralization and non-exportable verification. By explicitly targeting this root cause, the proposed framework distributes verification authority and renders correctness externally checkable, while preserving the privacy and integrity guarantees of the underlying TLS session.

The thesis builds on a selection of cryptographic building blocks, including zero-knowledge proofs, secure multiparty computation, Distributed Verifiable Random Functions (DVRFs), Threshold Signature Schemes (TSS), and publicly verifiable computation primitives. These components provide the foundation for constructing collusion-resistant attestations that remain compatible with existing TLS deployments

and standard cryptographic assumptions.

An initial multi-verifier construction based on DVRF-generated joint randomness and decentralized storage demonstrates that collusion resistance can be achieved by binding multiple verifiers to a single protocol execution, while keeping prover-side complexity constant. This design shows how joint randomness prevents any single verifier from unilaterally influencing protocol execution and how optimistic verification enables dispute resolution when deviations occur. At the same time, this construction reveals the practical limitations of decentralized storage and optimistic execution in scenarios where attestations must be consumed directly by smart contracts.

To overcome these limitations, the framework is refined by integrating threshold signature schemes, yielding compact and deterministic attestations suitable for on-chain verification. By reusing the distributed key generation phase already required for DVRFs, the DVRF-TSS construction avoids redundant setup costs and enables efficient aggregation of verifier approval. A detailed evaluation of communication overhead and on-chain verification costs shows that the resulting protocol remains practical even as the number of auxiliary verifiers increases, demonstrating that strong collusion resistance does not require prohibitive performance overhead.

A key conceptual contribution of this thesis is the introduction of Designed Exportable DCTLS (*dx-DCTLS*), a unifying abstraction that separates protocol execution from verification. By replacing non-verifiable components with verifiable cryptographic counterparts, *dx-DCTLS* exposes correctness conditions that can be validated by third parties throughout the protocol lifecycle. This abstraction addresses a fundamental gap in existing DCTLS designs, which ensure local correctness but do not support external verification or unified security definitions across constructions.

Building on this abstraction, a fully specified collusion-minimized TLS attestation protocol, $\Pi_{\text{coll-min}}$, is obtained by composing exportable verification with DVRF-TSS primitives. Its security is formalized through a game-based definition of threshold attestation unforgeability that captures adversarial capabilities unique to decentralized verifier environments. Under standard cryptographic assumptions, rigorous proofs establish that collusion resistance scales with the threshold parameter, while prover-side complexity remains $\mathcal{O}(1)$.

The practical viability of the proposed framework is demonstrated through prototype implementations, performance evaluations, and representative application scenarios. Experimental results obtained under both LAN and WAN settings show that the DVRF-TSS layer scales to deployments with dozens of verifiers, and that the additional overhead introduced by *dx-DCTLS* admits a clear and practical upper bound. In addition, network communication costs are explicitly measured, and the results show that reusing a pre-existing DKG drastically reduces communication overhead compared to configurations that require a fresh DKG execution. Use cases such as confidential binary options and off-chain credit and income verification illustrate how externally verifiable TLS attestations can be safely integrated into decentralized applications without relying on trusted intermediaries.

In summary, this thesis establishes a verifiable, modular, and scalable foundation for collusion-resistant TLS attestations. By unifying existing DCTLS constructions, introducing exportable verification through *dx-DCTLS*, and integrating distributed validation mechanisms via DVRFs and threshold signatures, it advances both the theory and practice of zkTLS-based systems. Compared to decentralized oracle networks, the proposed framework achieves substantially higher prover efficiency by avoiding repeated attestations, while also providing a formally defined and cryptography-based collusion minimization approach that does not rely on trusted execution environments or vendor-specific hardware assumptions. More broadly, this work lays the groundwork for future research on verifiable off-chain data access, decentralized trust reduction, and privacy-preserving interoperability between the web and blockchain ecosystems.

Beyond TLS-based constructions, the techniques developed in this thesis are applicable to other protocols that rely on designated or centralized verifiers. In particular, protocols with similar trust assumptions can be transformed into collusion-minimized designs by distributing verification authority using DVRF-TSS mechanisms and by replacing non-verifiable components with publicly verifiable counterparts, such as collaborative SNARKs or verifiable two-party computation. This suggests that a broader class of designed-verifier protocols can be adapted for secure and trust-minimized smart contract integration using the principles introduced in this work.

REFERENCES

- [1] J. Adler, R. Berryhill, A. Veneris, Z. Poulos, N. Veira, and A. Kastania, *Astraea: A decentralized blockchain oracle*, in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 1145–1152, IEEE, 2018.
- [2] A. R. Ağırtaş, A. B. Özer, Z. Saygı, and O. Yayla, *Distributed verifiable random function with compact proof*, in *International Symposium on Cyber Security, Cryptology, and Machine Learning*, pp. 119–134, Springer, 2024.
- [3] L. Bader, J. C. Bürger, R. Matzutt, and K. Wehrle, *Smart contract-based car insurance policies*, in *2018 IEEE Globecom workshops (GC wkshps)*, pp. 1–7, IEEE, 2018.
- [4] *Band protocol*, <https://www.bandprotocol.com/>, accessed: 2025-02-14, 2025.
- [5] B. Benligiray, S. Milic, and H. Vanttinen, *Decentralized apis for web 3.0*, API3 Foundation Whitepaper, 2020.
- [6] C. Bieri, *An overview into the interplanetary file system (IPFS): use cases, advantages, and drawbacks*, *Communication Systems XIV*, 28, 2021.
- [7] A. Boldyreva, *Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme*, in *International Workshop on Public Key Cryptography*, pp. 31–46, Springer, 2002.
- [8] 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*, Chainlink Labs, 1, pp. 1–136, 2021.
- [9] M. Brown and R. Housley, *Transport layer security (TLS) evidence extensions*, Working Draft, IETF Secretariat, Internet-Draft draft-housley-evidence-extns-01, 2006.
- [10] A. S. Cardozo and Z. Williamson, *EIP-1108: Reduce alt_bn128 precompile gas costs*, *Ethereum Improvement Proposals*, (1108), 2018.
- [11] S. Celi, A. Davidson, H. Haddadi, G. Pestana, and J. Rowell, *Distefano: Decentralized infrastructure for sharing trusted encrypted facts and nothing more*, *Cryptology ePrint Archive*, 2023.

- [12] Codex Storage Team, Codex: A decentralized durability engine: Whitepaper, Technical report, Codex Network, accessed: 2025-07-05, 2024.
- [13] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, Multiparty computation from somewhat homomorphic encryption, in *Annual cryptology conference*, pp. 643–662, Springer, 2012.
- [14] T. Dierks and E. Rescorla, The transport layer security (TLS) protocol version 1.2, Technical report, 2008.
- [15] Y. Dodis, Efficient construction of (distributed) verifiable random functions, in *Public Key Cryptography-PKC 2003: 6th International Workshop on Practice and Theory in Public Key Cryptography Miami, FL, USA, January 6–8, 2003 Proceedings 6*, pp. 1–17, Springer, 2002.
- [16] J. Ernstberger, J. Lauinger, Y. Wu, A. Gervais, and S. Steinhorst, Origo: Proving provenance of sensitive data with constant communication, *Proceedings on Privacy Enhancing Technologies*, 2025.
- [17] A. Fiat and A. Shamir, How to prove yourself: Practical solutions to identification and signature problems, in *Conference on the theory and application of cryptographic techniques*, pp. 186–194, Springer, 1986.
- [18] 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, IEEE, 2021.
- [19] General Data Protection Regulation (GDPR), <https://eur-lex.europa.eu/eli/reg/2016/679/oj>, EU Regulation 2016/679, 2016.
- [20] O. Goldreich, S. Micali, and A. Wigderson, How to play any mental game, or a completeness theorem for protocols with honest majority, in *Providing sound foundations for cryptography: on the work of Shafi Goldwasser and Silvio Micali*, pp. 307–328, 2019.
- [21] W. C. C. Group, Verifiable Credentials Data Model 1.0, <https://www.w3.org/TR/vc-data-model/>, W3C Recommendation, 2019.
- [22] I. Hajjeh and M. Badra, TLS Sign, Internet-Draft draft-hajjeh-tls-sign-04, Internet Engineering Task Force (IETF), Work in Progress. Available at: <https://datatracker.ietf.org/doc/draft-hajjeh-tls-sign/04/>, 2007.
- [23] A. Hassan, M. I. Ali, R. Ahammed, M. M. Khan, N. Alsufyani, and A. Alsufyani, Secured insurance framework using blockchain and smart contract, *Scientific Programming*, 2021(1), p. 6787406, 2021.

- [24] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar, Lucky 13 strikes back, in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, pp. 85–96, 2015.
- [25] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, On the security of TLS-DHE in the standard model, in *Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pp. 273–293, Springer, 2012.
- [26] M. Jakobsson, K. Sako, and R. Impagliazzo, Designated verifier proofs and their applications, in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 143–154, Springer, 1996.
- [27] P. Jauernig, A.-R. Sadeghi, and E. Stapf, Trusted execution environments: properties, applications, and challenges, *IEEE Security & Privacy*, 18(2), pp. 56–60, 2020.
- [28] J. R. Jensen, V. von Wachter, and O. Ross, An introduction to decentralized finance (DeFi), *Complex Systems Informatics and Modeling Quarterly*, (26), pp. 46–54, 2021.
- [29] D. Johnson, A. Menezes, and S. Vanstone, The elliptic curve digital signature algorithm (ECDSA), *International journal of information security*, 1, pp. 36–63, 2001.
- [30] K. Karantias, A. Kiayias, and D. Zindros, Smart contract derivatives, in *Mathematical Research for Blockchain Economy: 2nd International Conference MARBLE 2020, Vilamoura, Portugal*, pp. 1–8, Springer, 2020.
- [31] M. Keller, MP-SPDZ: A versatile framework for multi-party computation, in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pp. 1575–1590, 2020.
- [32] C. Komlo and I. Goldberg, Frost: Flexible round-optimized schnorr threshold signatures, in *International Conference on Selected Areas in Cryptography*, pp. 34–65, Springer, 2020.
- [33] H. Krawczyk and P. Eronen, Hmac-based extract-and-expand key derivation function (HKDF), Technical report, 2010.
- [34] J. Lauinger, J. Ernstberger, A. Finkenzeller, and S. Steinhorst, Janus: Fast privacy-preserving data provenance for TLS, *Proceedings on Privacy Enhancing Technologies*, 2025.
- [35] X. Liu, Z. Zhou, Y. Wang, B. Zhang, and X. Yang, Scalable collaborative zk-SNARK: fully distributed proof generation and malicious security, *Cryptology ePrint Archive*, 2024.

- [36] B. Möller, T. Duong, and K. Kotowicz, This poodle bites: exploiting the SSL 3.0 fallback, *Security Advisory*, 21, pp. 34–58, 2014.
- [37] A. Ozdemir and D. Boneh, Experimenting with collaborative zk-SNARKs, Zero-Knowledge proofs for distributed secrets, in *31st USENIX Security Symposium (USENIX Security 22)*, pp. 4291–4308, 2022.
- [38] Privacy & Scaling Explorations, TLSNotary: Privacy-preserving TLS data verification, <https://tlsnotary.org/>, accessed: 2025-06-22, 2025.
- [39] E. Rescorla, The transport layer security (TLS) protocol version 1.3, Technical report, 2018.
- [40] H. Ritzdorf, K. Wust, A. Gervais, G. Felley, and S. Capkun, TLS-n: Non-repudiation over TLS enabling ubiquitous content signing, in *Proceedings 2018 Network and Distributed System Security Symposium*, Internet Society, 2018.
- [41] M. Sabt, M. Achemlal, and A. Bouabdallah, Trusted execution environment: What it is, and what it is not, in *2015 IEEE Trustcom/BigDataSE/IsPa*, volume 1, pp. 57–64, IEEE, 2015.
- [42] V. Shoup, Practical threshold signatures, in *Advances in Cryptology-EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19*, pp. 207–220, Springer, 2000.
- [43] StackOverflowException, frost-secp256k1-evm: Frost threshold signature scheme for secp256k1 on EVM, GitHub repository, <https://github.com/StackOverflowException/frost-secp256k1-evm>, accessed July 6, 2025, 2025.
- [44] Standards for Efficient Cryptography Group (SECG), SEC 2: Recommended Elliptic Curve Domain Parameters, Technical Report Version 1.0, Certicom Research, 1999.
- [45] U. Şen, M. Osmanoglu, and A. A. Selçuk, An efficient solution for the collusion problem of DECO, in *2024 17th International Conference on Information Security and Cryptology (ISCTürkiye)*, pp. 1–6, IEEE, 2024.
- [46] U. Şen, M. Osmanoglu, and A. A. Selçuk, A collusion-resistant DECO-based attestation protocol for practical applications, in *2025 IEEE International Conference on Cyber Security and Resilience (CSR)*, pp. 234–239, IEEE, 2025.
- [47] U. Şen, DVRF-then-Sign, <https://github.com/seugu/DVRF-then-Sign>, accessed: 2025-10-31, 2025.
- [48] U. Şen, TLS-PRF simulation (gnark), <https://github.com/seugu/tls-prf-simulation-gnark>, accessed: 2025-10-31, 2025.

- [49] U. Şen, M. Osmanoğlu, O. Yayla, A. A. Selçuk, and A. Doğanaksoy, Collusion-minimized TLS attestation protocol for decentralized applications, *Cryptology ePrint Archive*, Paper 2026/277, 2026.
- [50] Wikipedia contributors, Binary option, available at: https://en.wikipedia.org/wiki/Binary_option. Accessed: 15 July 2025, 2025.
- [51] S. Williams, V. Diordiiev, L. Berman, and I. Uemlianin, Arweave: A protocol for economically sustainable information permanence, *Arweave Yellow Paper*, 2019.
- [52] X. Xie, K. Yang, X. Wang, and Y. Yu, Lightweight authentication of Web data via garble-then-prove., *IACR Cryptol. ePrint Arch.*, 2023, p. 964, 2023.
- [53] A. C. Yao, Protocols for secure computations, in *23rd annual symposium on foundations of computer science (SFCS 1982)*, pp. 160–164, IEEE, 1982.
- [54] A. C.-C. Yao, How to generate and exchange secrets, in *27th annual symposium on foundations of computer science (SFCS 1986)*, pp. 162–167, IEEE, 1986.
- [55] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, Town Crier: An authenticated data feed for smart contracts, in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 270–282, 2016.
- [56] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, Deco: Liberating web data using decentralized oracles for TLS, in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 1911–1928, 2020.
- [57] H. Zheng, T. Tran, R. Shadmon, and O. Arden, Decentagram: Highly-available decentralized publish/subscribe systems, in *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 274–287, IEEE, 2024.
- [58] R. Zhu, C. Ding, and Y. Huang, Efficient publicly verifiable 2PC over a blockchain with applications to financially-secure computations, in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 633–650, 2019.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Şen, Uğur

Nationality: Turkish (TC)

ORCID: 0009-0004-9825-7847

EDUCATION

Degree	Institution	Year of Graduation
M.S.	Cryptography, METU-IAM	2019
B.S.	Mathematics, Ege University	2015

PUBLICATIONS

- U. Şen, M. Osmanoğlu, and A. A. Selçuk, "An efficient solution for the collusion problem of DECO," in 2024 17th International Conference on Information Security and Cryptology (ISCTürkiye), Oct. 2024, pp. 1-6.
- U. Şen, M. Osmanoğlu, and A. A. Selçuk, "A collusion-resistant DECO-based attestation protocol for practical applications," in 2025 IEEE International Conference on Cyber Security and Resilience (CSR), Aug. 2025, pp. 234-239.
- U. Şen, M. Osmanoğlu, O. Yayla, A. A. Selçuk, and A. Doğanaksoy, Collusion-minimized TLS attestation protocol for decentralized applications, Cryptology ePrint Archive, Paper 2026/277, 2026.