# REALIZING INFORMATION ESCROWS AND EFFICIENT KEY-MANAGEMENT USING THRESHOLD CRYPTOGRAPHY

by

**Easwar Vivek Mangipudi**

**A Dissertation**

*Submitted to the Faculty of Purdue University*

*In Partial Fulfillment of the Requirements for the degree of*

**Doctor of Philosophy**


Department of Computer Science

West Lafayette, Indiana

August 2022

# THE PURDUE UNIVERSITY GRADUATE SCHOOL
# STATEMENT OF COMMITTEE APPROVAL

**Prof. Aniket Kate**

Department of Computer Science

**Prof. Michail Atallah**

Department of Computer Science

**Prof. Christina Garman**

Department of Computer Science

**Prof. Alexandros Psomas**

Department of Computer Science

**Approved by:**

Prof. Kihong Park

# ACKNOWLEDGMENTS

I like to extend my gratitude to my advisor Prof. Aniket Kate who continuously supported and silently motivated me throughout the journey at Purdue. I am not sure if he is the advisor I deserved but he is definitely the one I needed. He was always enthusiastic and easy to approach for a discussion on any subject. I also like to extend my sincere thanks and gratitude to my committee members and collaborators for the extensive and invigorating discussions.

No amount of thanks shall justify the support extended by parents Sri. Ramana Murty and Smt. Subba Lakshmi and my brother Uday Kiran in always encouraging me to pursue my heart; special thanks to my wife Harshitha Parnandi for standing by my side through the ups and downs of my journey.

I like to thank my friends Sai Raghavendra, Yogesh Dorbala, Siva Chaduvula, and Neha Chauhan with whom I got to have deep discussions about everything under the sun. My heartful thanks to my labmates Pedro Moreno-Sanchez, Mohsen Minaei Bidgoli, Sze Yiu Chau, Duc Le, Donghang Lu, Debajyoti Das, Adithya Bhat, and Tian Tian Gong for the wonderful discussions about life and cryptography.

I like to extend my gratitude to Purdue ICMAP for introducing me to a wonderful set of artistic people. I also thank Purdue Aikido, Taekwondo, Judo, and Aviation clubs for helping me pursue my interests and enhance my skills. Last but not least I like to thank West Lafayette, which held me in its lap with its silent snows, calm river, soothing environment, and wonderful people which always made me feel at home.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# ABSTRACT

In this thesis, we address two applications of threshold cryptography — designing information escrows and key-distribution in cryptocurrency systems. We design escrow mechanisms in two-party and multi-party scenarios such that any unauthorized revelation of data results in loss of cryptocurrency by the dishonest party. Later, we discuss user mental models in adopting cryptocurrency wallets and propose a protocol to efficiently provide cryptographic keys to the users in large-user systems.

An information escrow refers to users storing their data at a custodian such that it can be revealed later. In the case of unauthorized leakage of this data by the custodian (receiver of data), taking legal actions is expensive, time consuming and also difficult owing to difficulty in establishing the responsibility. We address this by automatically penalizing the custodian through the loss of cryptocurrency in case of leakage. Initially, we consider a two party scenario where a sender forwards multimedia data to a receiver; we propose the Pepal protocol where any *total or partial* leakage of data penalizes the receiver. To avoid single point of failure at the receiver in a two-party system, we extend the protocol to a multi-party system where a group of agents offer the escrow as a service. However, this introduces a collusion scenario among the rational agents leading to premature and undetectable unlocking of the data. Addressing this, we propose a *collusion-deterrent* escrow (CDE) protocol where any collusion among the agents is penalized. We show that the provably secure protocol deters collusion in game-theoretic terms by dis-incentivising it among the rational agents.

In the second part of this work, we investigate the mental models of cryptocurrency wallet users in choosing single-device or multi-device wallets along with their preferences. We investigate the user-preferred default (threshold) settings for the key distribution in the wallets. We then propose the D-KODE protocol, an efficient key-generation mechanism for cryptocurrency systems where either the payee or payer may not have the cryptographic setup but wish to transact. The protocol utilizes a practical black-box secret sharing scheme along with a distributed almost key-homomorphic PRF to achieve the threshold key distribution.

# 1. INTRODUCTION

## 1.1 Information Escrows

Information Escrow (IE) refers to the responsibility of safe storage of the data at a custodian [1]; an IE service [2] allows a user to encrypt her sensitive message to some *condition*, such that the message can only be revealed after the condition is met. This condition can be anything the user chooses and can be checked by a program (or smart contract). For example, these escrows can include checking for the share value of some company hitting a certain value, temperatures rising to a certain level to release funds for environmental programs, allegation escrows [3]–[5] or timed-release encryption [6]–[9] where users send messages to the future.

Offering escrow mechanisms as a service is increasing both in terms of service offerings and adoption. Examples include simple use cases of storing data in the cloud by individual users and software escrows by firms like Escrowtech [10] and Iron Mountain [11]. These firms provide software escrow services to technology companies that routinely appear on the Fortune 500 list. However, the cloud platforms and escrow services increasingly find themselves vulnerable to data breach attacks [12]–[15].

In the case of cloud storage or escrow offering by a single party, the user is forced to trust the service provider to not reveal their data to any third party. To overcome this, escrow services are realized using multiple agents in a distributed setting as threshold escrows. However, in such a distributed cryptographic setting, the current protocols [3]–[5], [7]–[9], [16]–[19] make a non-collusion assumption on the parties [20]–[22], assuming that at least a subset of a certain size of parties are always honest. However, this can not be true in a more realistic rational setting; any rational agent would collude with other agents to reveal the information stored in the escrows to maximize their utility.

In this work, we address the problems of leakage and collusion in escrow mechanisms — first, we develop a protocol Pepal that penalizes the dishonest receiver or service provider even under partial data leakage in a two-party setting. Then, we consider a more general threshold escrow setting with *rational* agents; the agents are free to collude to maximize their utility. Our CDE protocol deters any collusion among the agents to pre-maturely reveal or

reconstructs the data by penalizing through loss of cryptocurrency deposit. The proposed provably secure protocol ensures that the agents do not collude in game theoretic terms.

**Two-party escrow mechanisms and data leakage.** Data breach attacks on cloud platforms are increasing every year [12]–[15], the reasons for which vary from compromises of ill-maintained data servers to careless data custodians. Although it has been observed and reported that 90% of the data breaches can be avoided with good security practices on the custodian's infrastructure [23], there is no evident decrease in the number. Any legal recourse is expensive and time-consuming. To address this, we introduce a complementary security mechanism that is inexpensive, automated, and not restricted by the geo-political boundaries to disincentivize leakage of data. In particular, our goal is to make the data custodians more accountable through automatically enforceable monetary penalties resulting in immediate loss of funds. We ensure the penalization of even partial leakages (Pepal) by enforcing a cryptocurrency claim-or-refund smart contract with the deposit made by the data receiver.

Applicability scenarios for Pepal contracts range from industrial media custodianship, data and software escrows, leaking privately shared personal data (pictures and other media files) of others on social media and even to non-disclosure agreements between mutually distrusting entities [24]. Pepal can be useful in such circumstances by automating the monetary recovery procedure. We assume that the data sender and the custodian agree on an amount of money that will be awarded to the owner should specified documents be demonstrably leaked. Towards automatically ensuring that the owner will receive the funds, this amount could take the form of a surety bond that is held in trust by a Bitcoin or other permissionless/permissioned blockchain-based cryptocurrency smart contract. Another interesting use case is with users downloading paid media that should not be publicly shared on online platforms. The downloaders make a timed deposit along with the actual payment, for an agreed time and value for the download. If no dishonest sharing happens, it will be returned to the downloader/customer else it would be forfeited.

**Multi-agent threshold escrow mechanisms.** In a distributed/threshold information escrow (TIE) service, the input message gets shared among a group of agents through a

suitable distributed cryptographic primitive. Here, on one hand, a threshold number of agents are expected to combine their shares to open the message when the opening condition is met, and on the other hand, the message remains secret as long as only a lower than the threshold number of agents get compromised. While this threshold-bounded adversary assumption looks reasonable for many applications of distributed cryptography (such as threshold signatures wallets), it is indeed a stronger assumption for IE applications; these applications at times require significant longevity of usage.

If the escrow agents running a TIE service decide to collude and open a message before the attached condition is satisfied, they can do it passively in an undetectable manner. As the agents may collude among themselves in an undetectable manner, it is difficult to prevent such collusion over a long period of time; unless the protocol design itself makes collusion non-profitable. In this work, we design such a collusion deterrent protocol CDE where the best strategy for any rational agent will be to not collude.

Our protocols do not preclude the use of the court system, they simply complement it or shift the responsibility of bringing legal action to the entity seeking to recover their bond. Allowing an escalation to court is important as some disclosures are in the public interest (whistle-blowing) [25]. In fact, in certain cases, a third party might pay the value of the bond for the information (news media, crowdfunding, etc.). We expect that the proposed mechanisms encourage the parties involved to follow better security practices.

### 1.1.1 Contributions regarding information escrows

**Pepal: Penalizing data breaches in a two-party setting.** We formalize and provide a solution direction to the problem of automatically settling intentional or unintentional data breaches with a blockchain smart contract, eschewing the traditional recourse of costly legal action. Our Pepal protocol which achieves it, is a crypto-augmented smart contract system obtaining an arbitrator-free settlement. It comprises a claim-or-refund smart contract, a robust watermarking scheme, a proposed cryptographic primitive – Doubly Oblivious Transfer (DOT), and a non-interactive zero-knowledge (NIZK) proof for mutually distrusting parties.

In our core protocol, the sender and receiver create a claim-or-refund transaction on Bitcoin [26]–[28] where an amount is deposited that can be spent at any time with a jointly signed transaction or spent after a period of time by an sender-only signed transaction. The document provided to the receiver has the receiver's signing (private) key embedded in it with a robust binary watermarking scheme that cannot be removed (or retrieved) by anyone except the embedding party. The challenging aspects of the Pepal protocol involve arranging for the signing key to be embedded such that (1) the sender does not learn the value of the key at the time of embedding, (2) the receiver does not learn the document contents until the key is embedded, and (3) the sender is convinced the embedded key is the receiver's correct signing key. Within these constraints, to perform the embedding the parties must jointly perform a two-party computation with their respective private inputs. Our novel DOT and committed receiver oblivious transfer (CROT) protocols, securely realize this two-party computation to ensure that the sender can retrieve the receiver's embedded key from the document if it leaks (widely enough to reach the sender) and spend the deposited cryptocurrency.

Given the inherent non-cryptographic robustness guarantees of the robust watermarking system, we also analyze partial data disclosures. In particular, even when the receiver decides to reveal the document partially, our proposed DOT protocol ensures that the embedding party or the sender can retrieve significantly more bits than when the standard oblivious transfer is used for the transfer. For example, when the receiver's 256-bit signing key is embedded 16 times in the multimedia document, even a 15% leakage of document blocks reveals roughly 235 bits of receiver's key to the sender with DOT; as opposed to roughly only 50 bits that are revealed when oblivious transfer is employed.

**CDE: Collusion-deterrent threshold information escrow.** In the form of *collusion deterrent escrow* (CDE), we offer a solution direction and a provably secure protocol to address the *longitudinal trust issue* with threshold information escrows. If more than a threshold number of escrow agents collude to open some particular locked message from a user (say Alice), the proposed distributed protocol ensures that the locked deposits of the agents get released to an anonymous subset of agents (among the colluding agents)

prescribed by Alice. Thus, the protocol disincentivizes collusion, the agents will not attempt to collude to open any user message for the fear of losing their deposits and getting banned from offering any future services. Non-collusion assumption is extremely prevalent in the distributed cryptographic literature [16], [18], [19], [29]–[31] and overcoming it has been a significant barrier for the community for a long time.

It is typically assumed that out of $n$ agents, a maximum of $t$ agents can be corrupted who can act maliciously while the other $n-t$ are honest and follow the protocol without collusion. However, in this work, we let all the agents be rational rather than honest, allowing collusion. They act only to maximize their utilities. The $t$ corrupted parties can deviate arbitrarily from the protocol. Through game-theoretic analysis, we show that with the proposed mechanism, offering the encryption service in a non-collusive manner is the best response strategy for the agents.

We define our collusion deterrent escrow concept as an ideal functionality $\mathcal{F}_{\mathsf{CDE}}$; towards realizing it, we formally define and use a cryptographic primitive called distributed receiver oblivious transfer (DROT). DROT is a natural distributed version of the oblivious transfer [22] protocol, where multiple receivers share the choice bit in a threshold manner. Our $\mathsf{CDE}$ protocol employs a novel combination of DROT, robust bit watermarking, distributed key generation [30], [31], and secure bit decomposition [32]–[34] to securely realize $\mathcal{F}_{\mathsf{CDE}}$ in the mixed-behavior model [35] where the agents are either rational or malicious The protocol supports any condition that can be checked through a blockchain smart contract (even through interaction with the real world) as a condition of data release in the protocol.

### 1.2 Cryptocurrency wallets and threshold key management

The cryptocurrency boom has seen millions of people adopting digital assets; the recent economic successes [36]–[38] have enthused a broad population to explore them. The diversity in the needs and objectives of these cryptocurrency users is vast, ranging from just being enthused by technology to trading, sometimes even using all of their savings. With increasing adoption and valuation, the attacks on the system have also seen a rise. To combat these

attacks, designers constantly improve the security models with different architectures and user preferences in mind.

However, the number of users of each popular cryptocurrency wallet [1] (or crypto-wallet) such as Coinbase [39], [40] and Binance [41],[42] indicates higher popularity of wallets that seem (cryptographically) weaker in the security model they offer. This popularity can be because of various reasons, including people trusting the wallet firms, opting for wallets based on popular opinions and different security attitudes, etc. These variations in knowledge, understanding of security models, and risk perception may also significantly affect the choice of wallets.

Recent studies [43]–[47] attempted to understand usability and challenges while performing transactions with crypto-wallets in use. They analyze the wallets using cognitive walk-through [48] and also study the common misconceptions by the users regarding the role of wallet firm [43].

The focus of most of these previous works has been to characterize the *usability* and understanding of the in-use traditional single-device wallets. However, so far, there have been no studies regarding the emerging (and arguably more secure [49]) *multi-device wallets* that analyze the users' mental model of the security and key management of multi-device wallets to understand the barriers to their adoption. To put simply, a single-device wallet is a wallet with secret information (a secret key) stored in a single location. In contrast, in a multi-device wallet, the secret information is divided and stored on multiple devices, including servers hosted by the wallet firm and the user's devices. Owing to the increasing risks of key-compromise attacks [50], [51] and exchange hacks [52]–[54] on single-device wallets, one may expect a greater enthusiasm for the new and emerging multi-device wallets (e.g., Torus wallet [55], ZenGo [56]) which significantly mitigate these issues. However, in adoption, multi-device wallets lag far behind their single-device counterparts. This raises an important unanswered question: Is there an inherent gap between users' security expectations and the guarantees provided by current multi-device solutions, or are the multi-device wallets just ahead of their time? Here, we seek an answer to this question.

[1]↑A cryptocurrency wallet is an app that allows cryptocurrency users to store and retrieve digital assets. They typically involve a way to guard a secret key associated with the assets.

Specifically, in this work, we attempt to understand the user's perception towards multi-device wallets and qualify the gap between their designed security models of key management and the users' mental model. The study is also the first to consider distributed cryptography [57], [58] and its usability along with user preferences in wallets. Specifically, we conducted a survey-based study of 255 participants; analyzed their responses qualitatively and quantitatively to understand their current usage, choices, and if they are willing to change them given certain minimum information. Primarily, we investigate three research questions (**RQ**s):

**RQ1**: *What are the current usage-based groups, their preferences of wallets, and on what factors are they based?*

We investigated this question by asking the participant detailed questions about their current cryptocurrency wallets, their usage, and the features that made them choose a particular wallet. We enquire if their choice has been affected by ratings and reviews of the existing wallets. We also investigate their familiarity with different wallet types, including single and multi-device wallets, and their security concerns. Based on usage and preference responses, we analyze that all the participants behave as two groups: Newbies and Non-newbies. The newbies are recent users, while the non-newbies are relatively experienced users who have been using the wallets longer and invest more savings. The majority of participants use single-device wallets; however, more than 75% of the participants are concerned about losing funds by losing the key at the client device or compromising the secret key at the servers. At this point in the survey, both the groups are not very familiar with multi-device wallets.

**RQ2**: *Provided essential and sufficient information, are the users willing to shift to multi-device wallets? If not, why not?*

We investigated this question by first providing the users with essential knowledge regarding both single-device and multi-device wallets and then collecting feedback on the preferences. In particular, we asked the participants to watch two short videos on single and multi-device wallets. Our videos explain the single-device wallets, their challenges, and how multi-device wallets can mitigate them. After the videos and knowledge-check, we collected the preferences and feedback if the participants were willing to adopt multi-device wallets.

71.9% of participants mentioned they are ready to shift to multi-device wallets; however, 20.8% of the participants wanted to stick to single-device wallets.

**RQ3**: *What default key-management and architectural settings do they prefer for different wallets?*

We investigated this question by taking feedback for single and multi-device wallets on the secret information (key) location preferences under different possible attacks. We also took feedback regarding the choice of key storage of wallets under various government characteristics where the wallet firm may host servers in locations governed by multiple laws. We find that these government characteristics significantly impact the participants' key-location preferences from the survey. We also analyze how the participants prefer different settings, including the number of servers of the wallet firm storing the user keys. 63.13% of the participants preferred a small number of reputed servers compared to 31.76% choosing a higher number of servers. We provide a principled analysis of users' preferences by obtaining insights into why the users would or would not select multi-device wallets.

We observe that our results offer a few interesting insights and novel research directions for the threshold/distributed cryptography research itself. In the study, the participants expressed a desire for more control over their keys even when using multi-device wallets; the research community can focus on models achieving the same. The researchers should also consider more general adversary and access structures for multi-device wallets; however, the current distributed cryptographic systems literature and practice are pretty thin beyond the standard $(T-1)$-out-of-$N$ adversary. The participants also identified a privacy-accountability trade-off between existing types of multi-device wallets, which presents an exciting challenge for the distributed cryptography community.

### 1.2.1 Threshold key distribution

With the emergence of blockchain systems, for the first time, we are tokenizing our financial and supply-chain assets using cryptographic (private/signing) keys. However, if a user loses their private key, they lose the associated assets—there is no recovery mechanism as with the typical password-based authentication. Given the general lack of familiarity

with the technical aspects of public-private key management and maintenance, most first-time users choose custodial wallets [39], [41], [59], where a third party controls their keys. However, these third parties become single points of failure and can be targeted for large-scale thefts as well as financial surveillance and censorship. In general, this key management problem combined with a lack of simpler tools for key setup is regarded as a bottleneck of large-scale blockchain adoption. While start-ups such as Torus [60] are developing threshold cryptographic solutions towards maintaining secrecy and availability of the clients' keys, their approaches do not scale well as the number of clients grows. This work aims to provide a scalable key management system to generate keys on the fly for the rapid proliferation of blockchains to millions of users.

We consider two typical transaction scenarios with an online client Alice and an offline/new client Bob. In the first simple scenario, a cryptocurrency user Alice does not wish to generate and manage her keys locally on her own device. In the second more interesting scenario, Alice wishes to pay Bob, but she does not have Bob's public key. This can be because Bob either has never generated a key pair and is not available to engage immediately (as in the first scenario), or Bob is offline with his already generated public key not being available to Alice yet. For a concrete example, consider a cryptocurrency firm owned by Alice wishing to introduce their coins to new clients by offering free tokens (airdrops)[61]–[63]. Alice should be able to compute the public key corresponding to Bob's public string (identity) such that later Bob can use the same string to generate the related private key and claim funds.

**Existing distributed key generation (DKG) approaches.** Current solutions [60], [64] off-load the key generation and storage to a set of $n$ servers while preserving the secrecy of the keys against any $t$ servers. The servers generate key shares in a distributed form by running a distributed key generation (DKG) [30] instance for each identity and providing the secret key or public key shares for the identity as required. These architectures do not scale well because the servers need to store shares of every client and have to perform several DKG instances to generate the key shares for all the clients resulting in high computational and communication overhead. The overhead further amplifies if the system, over longer terms,

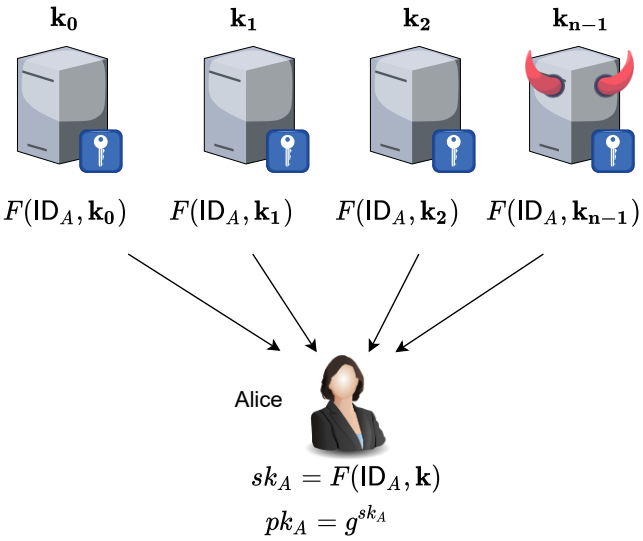



**Figure 1.1.** Scenario 1: Alice uses her public string $\mathsf{ID}_A$, obtains evaluations and computes private key $sk_A$ after authentication

attempts to provide proactive security [65] against mobile adversary [66]: All the millions of key shares need to be refreshed periodically, giving rise to issues of availability while the computation and communication intensive refreshing process are in progress.

**Employing distributed PRF.** In this work, we generate keys on-the-fly as pseudo-random function (PRF) [67]–[69] evaluations. A PRF is a deterministic function of a master (private) key and an input tag that is indistinguishable from a truly random function of the input and we plan to use the PRF output as a private/signing key. As a single node holding a master key $K$ introduces a key escrow and a single-point-of-failure for PRFs, we distribute the trust using distributed PRF (DPRF) such that a set of servers holds the master key $K$ in a secret shared fashion and generates shares of the client's private keys as partial PRF evaluations. Indeed, generating private keys using DPRFs [70]–[72] is already considered in the literature and we can employ any DPRF solution for our first scenario involving only Alice. However, none of the existing solutions is suitable for our second scenario involving Alice obtaining public keys of an offline Bob.

As an illustrative example consider private key generation for an identity (tag) $\mathsf{ID}_A$ using the well-known PRF by Naor *et al.* [70], [72]. This involves computing $sk_A =$

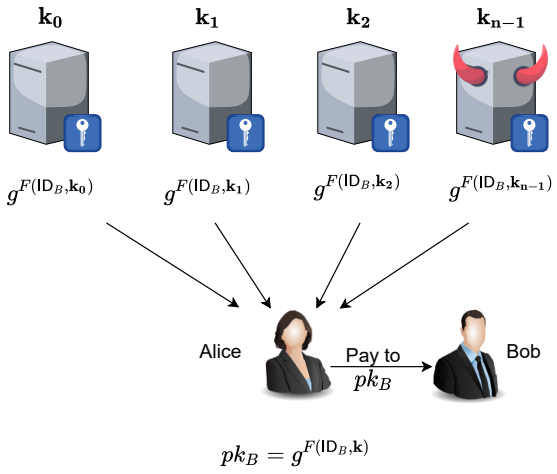



**Figure 1.2.** Scenario 2: Alice uses Bob's public string $\mathsf{ID}_B$ to obtain Bob's public key shares and computes his public key $pk_B$

$H_2\Big(F(K,\mathsf{ID}_A)\Big) = H_2\Big(H_1(\mathsf{ID}_A)^K\Big)$, where hash functions $H_1(\cdot)$ and $H_2(\cdot)$ map to a multiplicative group (of elliptic curve points) $\mathbb{G}$ and a scalar additive group $\mathbb{Z}_p$ respectively. When the key $K$ is shared among multiple servers, computing the key $sk_A$ from partial evaluations is straight-forward for Alice: she first computes $H_1(\mathsf{ID}_A)^K$ using Lagrange interpolations and then applies $H_2$ to the output locally. Our second scenario, instead, asks to securely provide Alice the *public key* $pk_B$ of an offline party Bob with identity $\mathsf{ID}_B$. To ensure that Alice cannot determine $sk_B$, computation of $pk_B = g^{H_2(F(K,\mathsf{ID}_B))}$ involves computing hash function $H_2(\cdot)$ through multi-party computation (MPC)—a highly expensive process in the threshold setting [73], [74].

To be able to efficiently generate the public keys, we need a PRF whose output is a scalar value in $\mathbb{Z}_p$ and which does not involve $H_2(\cdot)$ hash computations in the multi-party setting. We observe that most other distributed PRF [75]–[77] and easy-to-distribute key-homomorphic PRF constructions [78] in literature do not satisfy this requirement.

**Our Approach.** We employ the PRF in the lattice-based cryptography setting, $F(X,\mathsf{k}) = \Big\lfloor H(\mathsf{ID})\cdot\mathsf{k}\Big\rceil_p \in \mathbb{Z}_p, \mathsf{k}\in\mathbb{Z}_q^u, H(\cdot)\in\mathbb{Z}_q^u, p<q$ [71] to generate keys for both the scenarios as in Figures 1.1, 1.2 . It is an almost-key homomorphic PRF, with an error $\{-1,0,1\}$ in the evaluation for every additive term. The master key $\boldsymbol{K}$ is threshold-shared among the servers. However, unlike standard threshold designs [60], [79], we cannot employ Shamir secret shar-

ing (SSS) [80] for sharing $\mathsf{k}$ in almost-homomorphic PRF as the reconstruction (Lagrange) coefficients blow up the error (and error combinations) when computing the PRF output from the partial evaluations. Another common secret sharing mechanism replicated secret sharing (RSS) [81], [82] may be employed as the RSS shares need to be simply added to compute the value, which ensures that the error remains bounded within the range $[-n, n]$. However, the number of RSS shares grows exponentially as $\binom{n-1}{t}$ for an $(n, t)$ threshold structure among servers with $t = O(n)$; this has high storage and share-refreshing computation overhead and RSS-based distributed PRF can only be applied to settings with ten or lower servers. Therefore, solving our distributed PRF problem requires going beyond the commonly employed SSS and RSS schemes.

In this work, we demonstrate that the black-box secret sharing (BBSS) approach [83] can be made practical towards catering to a higher number of servers; this is the first effort that realizes its utility in practice. We propose the D-KODE protocol which generates private and public keys using almost key-homomorphic PRF evaluations, where key-sharing among the servers is performed through BBSS. Our BBSS instantiation ensures that the evaluation coefficients are in the set $\{-1, 0, 1\}$, resulting in the output key being in a very small range of keys linear in the number of servers such that Bob can efficiently compute the private key associated with the public key employed by Alice to pay Bob.

In this work, we demonstrate that the black-box secret sharing (BBSS) approach [83] can be made practical towards catering to a higher number of servers and employ it for sharing the master key among the servers; in fact, this is the first effort that realizes its utility in practice. We propose the D-KODE protocol, which generates discrete-log private and public keys using almost key homomorphic PRF evaluations, where the master key is shared among servers through BBSS. Our BBSS instantiation ensures that the reconstruction coefficients are in the set $\{-1, 0, 1\}$. In the scenario where Alice to pays to a new Bob, the small reconstruction coefficients help Bob efficiently compute the private key of the public key to which Alice paid.

For share refreshing, in D-KODE, we refresh the shares of the master key instead of (all the millions of) user keys. This makes share refreshing using proactive secret sharing *independent* of the number of user keys resulting in only constant overhead. Further, while

computing the user keys from PRF evaluations, we use a verifiability mechanism for the PRF to allow the clients to verify the evaluations. Our prototype implementation provides D-KODE protocol with BBSS-DKG mechanism for network size up to 50 servers. We observe that D-KODE starts to outperform the state of the art at 94K keys for a 20-server system. Using D-KODE, a server supports generating upto ten `secp256k1` keys per second per thread.

### 1.2.2 Contributions regarding threshold key management

- We propose a solution for generating keys where two parties like to transact when either or both the parties do not have mechanisms for securely generating signing key locally, even when one of them is offline and the other party only knows his verifiable identity.
- As a key step, we propose efficient approaches to realize black box secret sharing (BBSS) for practical setting, which can be of independent interest to threshold cryptography [84].
- We instantiate the first DKG mechanism using BBSS scheme and provide a proactive secret sharing scheme that supports different node memberships and access structures for different epochs. Our scheme offers constant computational overhead and hence scales well with a large number of keys in the system. Through our experimental analysis, we observe that D-KODE outperforms the plain DKG approach as the number of keys increases to 100 thousand.
- We also provide a verification mechanism for the PRF $F(X, \mathsf{k}) = \left\lfloor H(X) \cdot \mathsf{k} \right\rfloor_p \in \mathbb{Z}_p, \mathsf{k} \in \mathbb{Z}_p^u, H(\cdot) \in \mathbb{Z}_q^u, p < q$, which can be of independent interest.

## 1.3 Organization of the thesis

In Chapter 2 we introduce the building blocks and cryptographic primitives that are required for the protocols in this thesis. In Section 2.1 of the chapter we describe the various multi-party computation primitives including secret-sharing, distributed key generation (DKG), black-box secret sharing (BBSS) and oblivious transfer (OT). We also introduce robust watermarking, claim-of-refund contracts and monotone boolean formula for majority circuits. In Chapter 3 we first define the two party escrow problem, propose the cryptographic

primitive doubly-oblivious transfer (DOT) in Section 3.3 and propose the Pepal protocol for achieving automatic penalization in a two party scenario in Section 3.6. We also provide the implementation details in Section 3.7. In Chapter 4 we propose the collusion-deterrent escrow (CDE) protocol to achieve automatic penalization in a multi-agent scenario where the agents are allowed to collude. We introduce the problem definition in Section 4.1, describe the cryptographic setup in Section 4.3 and the game theoretic analysis in Section 4.4.

In Chapter 5 we analyze the mental models of users in adopting different cryptocurrency wallets. We describe the different wallet types and the classification of single-device and multi-device wallets in Section 5.1. We analyze the responses of 255 participants, we present the survey methodology including recruiting, participant demographics, limitations, and the procedure for quality control in Section 5.4. We discuss the results of the survey in Section 5.5 and their implications to the industry and academia in Section 5.6. In Chapter 6 we develop the D-KODE protocol which supports efficient threshold key generation and management for a large set of keys in the system. In Section 6.1.1 we discuss the system setup and provide an overview of the proposed D-KODE protocol. In Section 6.4 we describe the different algorithms of the D-KODE protocol and verification procedure of the PRF employed. In Section 6.7 we describe the proactive black-box secret sharing mechanism and provide the implementation and performance details in Section 6.8. Finally we provide the conclusion and possible future directions in Chapter 7.

# 2. BUILDING BLOCKS AND CRYPTOGRAPHIC TOOLS

In this chapter, we introduce different basic building blocks and cryptographic tools that are employed in the development of different protocols in this work.

## 2.1 MPC Primitives

Multi-party computation (MPC) [85]–[88] is an approach allowing mutually distrusting parties to collaboratively compute some functions with their private input. We use MPC modules for distributed key-generation, bit-decomposition, and two-party computation between the user and each of the agents. We follow the standard online/offline MPC paradigm such that an offline phase can be leveraged to generate input-independent pre-processed values. These values are used in the online phase to speed up the computations where the actual input is involved. For instance, Beaver triples [89] are used to multiply two secret shares.

### 2.1.1 Secret Sharing

In a secret sharing scheme [80], [90]–[92], a designated *dealer* shares a secret among a set of parties such that a certain subset of parties can interact to reconstruct the secret. All the subsets that are designated to reconstruct the secret are *qualified* sets and the set of all qualified sets is called an access structure. The threshold-$t$ access structure $T_{(n,t)}$ is the collection of subsets of parties of cardinality greater than $t$. Any subset of parties outside the access structure has no information about the secret. When the total number of parties is $n$, we denote such a scheme as $(n, t)$-secret sharing, where at least $t+1$ parties are needed for reconstruction.

### 2.1.2 Black Box Secret Sharing—BBSS

A black-box secret sharing scheme is a linear secret sharing scheme over a finite Abelian group and can be instantiated with just black box access to group operations and random group elements. The secret generation and reconstruction is by linear combination of share

elements. We use the construction of black box secret sharing scheme such that the reconstruction coefficients lie in the set $\{-1, 0, 1\}$.

In black box secret sharing [83], the dealer shares an element of an Abelian group (e.g., $\mathbb{Z}_q$ with publicly known $q$) where the share elements are computed as linear combination of the secret value and random elements chosen by the dealer. This computation is performed as a multiplication of a distribution matrix $\boldsymbol{M}$ and the random element vector $\boldsymbol{\rho}$. Any set of parties from the qualified set can reconstruct the secret as a linear combination of their shares.

**Share generation.** Consider a dealer sharing a secret $s \in \mathbb{Z}_q$ with a set of parties over the (monotone) access structure denoted by $\Gamma$. To generate shares for the parties in BBSS, the dealer uses a distribution matrix $\boldsymbol{M} \in \mathbb{Z}^{d \times \mathrm{e}}$ and a distribution vector $\boldsymbol{\rho} = (s, \rho_2, \rho_3, \cdots, \rho_{\mathrm{e}})^T$ with secret $s$, $\{\rho_{\mathrm{i}}\}_{\mathrm{i}=2}^{\mathrm{e}}$ uniform randomly chosen from $\mathbb{Z}_q$. The vector of share elements $\boldsymbol{s} = (s_1, s_2, \cdots, s_d)^T$ is computed as $\boldsymbol{s} = \boldsymbol{M} \cdot \boldsymbol{\rho}$.

Each party $P_{\mathrm{i}}, \mathrm{i} \in \{1, 2, \cdots, n\}$ is assigned a set of share elements using a surjective function $\psi : \{1, \cdots, d\} \to \{1, \cdots, n\}, d > n$. The $\mathrm{i}^{\mathrm{th}}$ share element $s_{\mathrm{i}}$ is assigned to the party $\psi(\mathrm{i})$ who is said to *own* the $\mathrm{i}^{\mathrm{th}}$ row of the matrix $\boldsymbol{M}$. For any subset of shareholders $A$, $\boldsymbol{M}_A \in \mathbb{Z}^{d_A \times \mathrm{e}}, \boldsymbol{s}_A \in \mathbb{Z}^{d_A}$ denote the set of rows of $\boldsymbol{M}$ and elements of $\boldsymbol{s}$ jointly owned by the parties in $A$. We let $T_{\mathrm{j}} = \psi^{-1}(\mathrm{j})$ be the set of all row indices held by party $P_{\mathrm{j}}$. Any set $A \in \Gamma$ is a qualified set and sets $A \notin \Gamma$ are *forbidden* sets. The $\mathrm{j}^{\mathrm{th}}$ share holder holds $d_{\mathrm{j}} = \|\psi^{-1}(\mathrm{j})\|$ number of share-units.

The tuple $\mathcal{M} = (\boldsymbol{M}, \psi, \epsilon)$ is called an Integer span program (ISP) when $\boldsymbol{M} \in \mathbb{Z}^{d \times \mathrm{e}}$ and the rows of $\boldsymbol{M}$ are labelled by the surjective function $\psi$. $\boldsymbol{\varepsilon} = \{1, 0, \cdots, 0\} \in \mathbb{Z}^{\mathrm{e}}$ is called the target vector. When $\mathcal{M}$ is an ISP for $\Gamma$, the conditions specified by Definition 2.1.1 hold and $\boldsymbol{M}$ can be used as a distribution matrix to realize the access structure. This defines a reconstruction vector, which is used to reconstruct the secret when $\boldsymbol{M}$ is used as distribution matrix to share the secret value.

**Definition 2.1.1.** *An integer span program (ISP) [83], [93] $\mathcal{M} = (M, \psi, \epsilon)$ is an ISP of the access structure $\Gamma$ if for all $A \in \{1, 2, \cdots, n\}$ the following holds: If $A \in \Gamma$, then there exists a reconstruction vector $\boldsymbol{\lambda}_A \in \mathbb{Z}^{d_A}$ such that $\boldsymbol{M}_A^{\top} \boldsymbol{\lambda}_A = \boldsymbol{\varepsilon}$, where $\boldsymbol{\varepsilon} = \{1, 0, \cdots, 0\}$. If*

*$A \notin \Gamma$, there exists a sweeping vector $\boldsymbol{k} = (k_1, k_2, \cdots, k_{\mathrm{e}}) \in \mathbb{Z}^{\mathrm{e}}$ such that $\boldsymbol{M}_A \boldsymbol{k} = \boldsymbol{0} \in \mathbb{Z}^d$ with $\boldsymbol{k}^\top \cdot \boldsymbol{\varepsilon} = 1$.*

The first condition states that for every qualified set, there exists a reconstruction vector, thereby making the reconstruction of the shared secret possible.

**Reconstruction.** For a qualified set $A$, the secret value $s$ is reconstructed as $s = \boldsymbol{s}_A^\top \cdot \boldsymbol{\lambda}_A$. Here $\boldsymbol{s}_A$ is the vector of all share elements (subset of vector $\boldsymbol{s}$) held by the parties in the set $A$ and $\boldsymbol{\lambda}_A$ is the corresponding reconstruction vector.

To realize a threshold access structure, one needs to compute the corresponding distribution matrix $\boldsymbol{M}$. For that, we use the Benaloh-Leichter (BL) secret sharing construction [93], [94] where the access structure is expressed as monotone boolean formulae. The BBSS scheme using the BL construction ensures that elements of the reconstruction vector $\boldsymbol{\lambda}$ are small and in $\{-1, 0, 1\}$.

**Verification.** : Each party $P_{\mathrm{i}}$ receives the share vector $\boldsymbol{s}_{\mathrm{i}}$ and the broadcast commitment vector $C$. The matrix $\boldsymbol{M}$ corresponding to the access structure is locally computed by all the parties.

The parties verify each of the received share elements as follows: let the $\mathrm{i}^{\mathrm{th}}$ row of matrix $\boldsymbol{M}$ be $(m_{\mathrm{i},1}, m_{\mathrm{i},2}, \cdots, m_{\mathrm{i,e}})$, the party with share element $s_{\mathrm{i}}$ (and $s'_{\mathrm{i}}$) verifies the share using the following verification: $g^{s_{\mathrm{i}}} h^{s'_{\mathrm{i}}} = \prod_{\mathrm{j}=1}^{\mathrm{e}} C_{\mathrm{j}}^{m_{\mathrm{i,j}}}$

$$\prod_{\mathrm{j}=1}^{\mathrm{e}} C_{\mathrm{j}}^{m_{\mathrm{i,j}}} = \prod_{\mathrm{j}=1}^{\mathrm{e}} \left( g^{\rho_{\mathrm{j}}} h^{r_{\mathrm{j}}} \right)^{m_{\mathrm{i,j}}} = \prod_{\mathrm{j}=1}^{\mathrm{e}} \left( g^{\rho_{\mathrm{j}} m_{\mathrm{i,j}}} \right) \left( h^{r_{\mathrm{j}} m_{\mathrm{i,j}}} \right)$$
$$= g^{\sum_{\mathrm{j}=1}^{\mathrm{e}} \rho_{\mathrm{j}} m_{\mathrm{i,j}}} h^{\sum_{\mathrm{j}=1}^{\mathrm{e}} r_{\mathrm{j}} m_{\mathrm{i,j}}} = g^{s_{\mathrm{i}}} h^{s'_{\mathrm{i}}}$$

If the verification does not hold, the party with the share element $s_{\mathrm{i}}$ broadcasts a complaint along with the share elements $(s_{\mathrm{i}}, s'_{\mathrm{i}})$ to all the parties. If more than $t+1$ complaints are broadcast in the system, the dealer is deemed malicious, else the dealer responds to the complaint by broadcasting the share forwarded to the party.

### 2.1.3 Oblivious Transfer

1-out-of-2 oblivious transfer ($\mathsf{OT}_1^2$) is a two-party (a sender and a receiver) computation mechanism, where the sender has two messages $M_0$ and $M_1$ and the receiver has a bit $b \in \{0, 1\}$. The goal is to transfer $M_b$ to the receiver and at the end of the protocol, the receiver should not learn any information about $M_{1-b}$ and the sender should not learn $b$.

**1-out-of-2 Verified Simplest Oblivious Transfer:.**

| **Sender** | | **Receiver** |
|---|---|---|
| Messages $M^0, M^1$ | | choice bit $b$ |
| $a \leftarrow_R \mathbb{Z}_q$ | $\xrightarrow{h=g^a}$ | $r \leftarrow_R \mathbb{Z}_q$ |
| | $\xleftarrow{c=g^r h^b}$ | |
| $k_0 = H(c^a)$ | | |
| $k_1 = H((c/h)^a)$ | | |
| $p = H(H(k_0)) \oplus H(H(k_1)$ | $\xrightarrow{p}$ | $k_b = H(h^r)$ |
| Verify $p' = H(H(k_0))$ | $\xleftarrow{p'}$ | $p' = H(k_b) \oplus pb$ |
| $C_0 = E_{k_0}(M^0)$ | | |
| $C_1 = E_{k_1}(M^1)$ | $\xrightarrow{C_0, C_1}$ | |
| | | Decrypt $C_b$ |

**Figure 2.1.** 1-out-of-2 Oblivious Transfer [22]

In this protocol, by Doerner et.al. [22] (an augmented version of Oblivious Transfer by Chou *et al.* [95]), given a multiplicative group $\mathbb{G}$ and its generator $g$, the sender initially chooses a random value $a \leftarrow_R \mathbf{Z}_q$ and the receiver chooses a random value $r \leftarrow_R \mathbf{Z}_q$. The sender transmits $h = g^a$ to the receiver who computes $c = g^{ab+r}$ and transmits to the sender. The sender then computes two keys $k_0$ and $k_1$ as $k_0 = H(c^a)$ and $k_1 = H(ch^{-1})^a$ and computes a challenge $p = H(H(k_0)) \oplus H(H(k_1)$ and forwards it to the receiver. The receiver computes the key $k_b = H(h^r)$ and returns $p' = H(k_b) \oplus pb$. After verifying if $p' = H(H(k_)$, the sender encrypts $M_0$ and $M_1$ using these two keys generating $C_0$ and $C_1$ which are then forwarded to the receiver. The receiver decrypts the message $M_b$ using the key $k_c = h^r$.

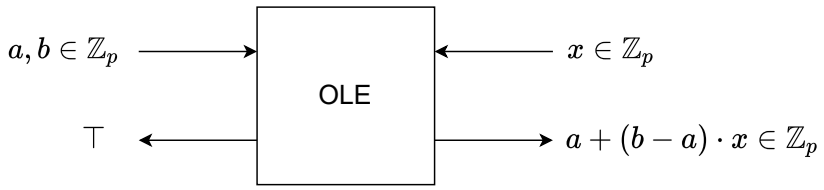



**Figure 2.2.** Oblivious linear function evaluation (OLE)[96]. The sender has two inputs $a, b \in \mathbb{Z}_p$, the reciever has input $x \in \mathbb{Z}_p$, the receiver obtains the value $a + (b - a) \cdot x \in \mathbb{Z}_p$ obliviously.

Depending on $b$, only one of $k_0$ and $k_1$ would be equal to $g^{ar}$ computed by the receiver. The other key $g^{ar-r^2}$ can not be computed by the receiver and hence learns no information about $M_{b-1}$. As the sender just encrypts and forwards the two messages, learns no information about the bit $b$. Figure 2.1 provides the depiction of the protocol. The advantage of adding the verification step is that it forces the receiver to compute the keys before receiving the encryptions and makes the protocol (UC)secure in the real-world ideal paradigm.

### 2.1.4 Oblivious linear function evaluation—OLE

OLE is a two-party computation protocol where the sender has inputs $a, b \in \mathbb{Z}_p$ and the receiver has the input $x \in \mathbb{Z}_p$. After running the protocol, the receiver obtains the value $a+(b-a)\cdot x \in \mathbb{Z}_p$ and the sender obtains $\top$ if the protocol run is successful (refer Figure 2.2). The sender does not have any information about $x$ or the value obtained the receiver. There are several constructions of OLE [96]–[100]; in this work we consider OLE as a UC-secure blackbox. The ideal functionality of OLE is provided in Figure 2.3.

### 2.1.5 Distributed key generation—DKG

A $(n, t+1)$ DKG [30], [31] mechanism allows $n$ parties to generate a public key and shares of the corresponding secret key in a distributed manner. At the end of the generation phase, each node has a share of the secret key $sk$ and at-least $t+2$ parties are needed to reconstruct the key. No subset of parties with the size less than $t+2$ has any knowledge of it. A DKG mechanism is defined by two phases, the sharing phase at the end of which every party holds a share $[\![sk]\!]_j$ of the key $sk$, and the reconstruction phase involving every node

**Functionality $\mathcal{F}_{OLE}$**

The functionality $\mathcal{F}_{OLE}$ interacts with the sender $\mathsf{S}$, receiver $\mathsf{R}$ and adversary $\mathcal{A}$. The sender $\mathsf{S}$ has two messages $a, b$ and the receiver has a random value $x \in \mathbb{Z}_p$.

- Upon receiving the message $(\texttt{inputS}, a, b)$ from $\mathsf{S}$, verify that there is no tuple stored, else ignore the message. Store $a, b$, forward the message $(\texttt{input})$ to $\mathcal{A}$.
- Upon receiving the message $(\texttt{inputR}, x)$ from $\mathsf{R}$, verify that there is no stored tuple, else ignore the message. Store $x$, forward the message $(\texttt{input})$ to $\mathcal{A}$.
- Upon receiving the message $(\texttt{deliver}, \mathsf{S})$ from $\mathcal{A}$, check if both $a, b$ and $x$ are stored, else ignore the message. Send $(\texttt{delivered})$ to $\mathsf{S}$.
- Upon receiving the message $(\texttt{deliver}, \mathsf{R})$ from $\mathcal{A}$, check if both $a, b$ and $x$ are stored, else ignore the message. Set $y = a + (b - a) \cdot x$ and send $(\texttt{output}, y)$ to $\mathsf{R}$.

**Figure 2.3.** Ideal Functionality of OLE [96], [97]

broadcasting their share and running the reconstruction algorithm on the collected shares. The two algorithms for share generation and reconstruction are:

- $\mathsf{dkg.share}(n, t+1, \lambda)$ takes in the total number of parties $n$, the threshold $t+1$, the security parameter $\lambda$ and returns to each party $\mathsf{A}_{\text{IND}}$, a share $[\![sk]\!]_{\text{i}}$ of the secret key $sk$ and the corresponding public key $pk$.
- $\mathsf{dkg.recon}([\![sk]\!])$ takes in the vector of shares with at least $t+2$ verified shares and returns the reconstructed value $sk$.

### 2.1.6 Bit decomposition

A bit decomposition protocol [32]–[34] takes a secret share as input and transforms the share into bit-wise shared values *i.e.,* for a value $sk$, upon input of all the shares $[\![sk]\!]$, the protocol outputs the shares $[\![sk_{\text{i}}]\!]$, where $sk_{\text{i}}, \text{i} \in [0, \lambda - 1]$ are the bits of the value $sk$. It is defined by two algorithms:

- $\mathsf{bit.decomp}(\lambda, [\![sk]\!])$ takes in the total number of users and the shares of the secret key $sk$ and returns to every agent $\mathsf{A}_{\text{IND}}$ a vector $[\![sk_{\text{i}}]\!]_{\text{IND}}$ of shares of the bits of the secret key.

- $\mathsf{bit.recon}([\![sk_i]\!])$ takes in the vector of shares of a particular bit and returns the reconstructed bit $sk_i$.

### 2.2 Pseudo Random Function

A pseudo-random function (PRF) family [67]–[69] is a set of keyed functions with a common domain and range, such that no efficient algorithm can distinguish between a randomly chosen function from the PRF family and a random oracle. A key homomorphic PRF [71], [101] is a PRF that is homomorphic with respect to the key-input of the function. It can be used as a distributed PRF [70], [72], [75] when the key is shared among the servers. With an almost-key homomorphic PRF, the computed evaluation from the shares may differ (may not be equal) from the evaluation of the PRF.

In this work, we use an almost-homomorphic PRF which is based on the Learning-with-rounding (LWR) assumption (Definition 6.1.2).

**Employed PRF [71].** Given a hash function $H : \mathcal{X} \to \mathbb{Z}_q^u$, a key vector $\mathsf{k} \in \mathbb{Z}_q^u$ and with $p < q$, the PRF evaluation $F : \mathcal{X} \times \mathbb{Z}_q^u \to \mathbb{Z}_p$ of the string $X$ is $F(X, \mathsf{k}) = \left\lfloor H(X) \cdot \mathsf{k} \right\rceil_p \in \mathbb{Z}_p$.

That $F(\cdot, \cdot)$ is a PRF in the random oracle model follows directly from the LWR assumption discussed above, where $H(X)$ corresponds to matrix $\boldsymbol{A}$ with a single row ($m = 1$) and the secret key $\mathsf{k}$ to vector $\boldsymbol{s}$.

In this work, we use the above PRF in the distributed setting, with a key $\boldsymbol{K}$ shared among them and among $n$ servers and each server locally computing a (partial) evaluation of the PRF on some input $X$ non-interactively. Any party wishing to compute the PRF $F(X, \mathsf{k})$ inputs $X$ to the servers and obtains evaluations from the each of them to reconstruct $F(X, \mathsf{k})$.

### 2.3 Robust Bit Watermarking

A robust watermarking scheme is defined by the property that the watermark can not be removed without loss of information from the watermarked data. The watermarking scheme is defined by three algorithms, for key generation, embedding the watermark and detection of the watermark. $\mathcal{M}$ is the set of all possible documents, $\mathcal{W} \in \{0, 1\}$ the set of all possible

watermarks, $\mathcal{K}$ is the set of all keys and $n$ is the security parameter. The three algorithms define the scheme:

- wm.gen $(n)$: Given $n$, outputs keys $k_{emd}, k_{det} \in \mathcal{K}$ probabilistically.
- wm.embed $(M, w, k_{emd})$: Takes the document $M$, watermark $w \in \mathcal{W}$ and embedding key $k_{emd}$ as inputs and generates a watermarked document $M'$.
- wm.detect $(M', k_{det}, w)$: Takes the watermarked document $M'$, the detection key $k_{det}$ and the watermark $w$ as input and outputs $\top$ if the watermark in $M'$ matches $w$, else outputs $\perp$.

The watermarking scheme is expected to satisfy the properties of imperceptibility and robustness. To describe the properties, we adapt the watermarking definition suggested by Adelsbach *et al.* [102]. We assume a given similarity function $sim(M, M')$ which returns $\perp$ if the two documents $M$ and $M'$ are not similar and $\top$ if they are.

- *Imperceptibility*: The watermarked and the original versions of the document should be similar i.e., $\forall M \in \mathcal{M}, \forall k_{emd} \in \mathcal{K}$ and $\forall w \in \mathcal{W}$,
  if wm.embed$(M, w, k_{emd}) \rightarrow M'$, then $sim(M, M') = \top$.
- *Robustness:* No known algorithm should be able to effectively change or remove the watermark in the watermarked document without leaving the document itself unusable, even with the detection key.

Our robustness definition is a weaker notion of traditional notion of robustness [103] which states that the any attempt to remove the watermark should result is destruction of watermarked data. In terms of real-world applicability, not all watermarking schemes that are currently used in practice for different kinds of data [104]–[106] have a theoretical proof of robustness. However, for our protocol to be applicable we do need such a proof as long as there is no known attack (till the end of expiration of escrow condition) on the watermarking scheme being employed. This is reflected in the robustness definition provided above. While theoretically, an algorithm may exist which can remove the watermark from the data, we

just require that such an algorithm should not be available or known to humans; Rogaway [107] formalized this approach.

We also need the watermarking to be imperceptible and the watermarked data to be 'similar' to un-watermarked data implying no loss of information from the original data. We do not need cryptographic indistinguishability of watermarked and un-watermarked data because the agent is aware that the data is watermarked, they do not (and should not) know what the bit that is watermarked in the data.

## 2.4 Claim-or-refund deposit

A claim-or-refund escrow deposit involves a deposit that can be claimed when possession of certain information like secret keys is proven or is returned to the creator upon the embedded condition being satisfied. In a timed-release scenario with a time-lock deposit, any party which produces the valid signature will be able to transfer the funds before the time period specified in the contract expires, else the funds are returned to the party creating the deposit. We depict below in Algorithm 1, the claim-or-refund contract logic used in the protocol.

Before the start of the protocol, every agent makes a deposit locking the funds to the contract which requires the signature using the secret key $sk$ of the protocol instance. The condition specified by the user and agreed on by all the agents is embedded into the contract such that as soon as the condition is met (like the expiry of a time period), the funds are transferred back to the agents. Depending on the complexity of the condition and the choice of the user, different cryptocurrency systems can be used for the contract creation.

The cryptocurrency script or smart contract implements the required claim-or-refund functionality with the embedded condition.

### 2.4.1 Bitcoin Claim-or-Refund Contract

Bitcoin [108] is a peer-to-peer decentralized network where participants are represented by a public and private key pair. The hash of the public key serves as the user's address and the private key is used to sign and authorize transactions. *Script* in Bitcoin is a stack-based

**Algorithm 1** Claim-or-refund contract

```
1: if Escrow Condition == True  then
2:     Direct the locked funds back to the contract creator
3: else
4:     if signature corresponding to public key pk of protocol instance is valid then
5:         Direct the funds to the mentioned recipient
6:     else
7:         Transaction is invalid
```

language simulating a Push Down Automata and is used to write a smart contract. Spending funds typically involves executing/running two scripts on the spender's machine. The first is scriptPubKey which is embedded in the input transaction under the script field. It entails the conditions that must be met to spend the unspent transaction outputs (UTXO). The second one is *scriptSig* which is an unlocking script provided by the user who wants to spend the UTXO. When *scriptSig* and *scriptPubKey* are executed in sequence, the user gets to know if the transaction is valid. Bitcoin offers both sender and receiver of the funds an aspect of privacy until the funds in the deposit are directed to a recipient i.e., in our case, after the documents become public and the key gets revealed to the sender. Such privacy is not observable in any other non-blockchain financial system.

*Time-Locked Compensation Deposits:* We construct scriptPubKey with two prominent Bitcoin scripting language operators: OP_CHECKLOCKTIMEVERIFY and OP_CHECK-MULTISIGVERIFY. OP_CHECKLOCKTIMEVERIFY allows users to create transactions whose outputs can only be spent in the future. OP_MULTISIGVERIFY allows the creation of transactions which need multiple signatures. In our case, the receiver creates a deposit which is locked till a future time $t$. The funds of the deposit can be transferred only if both the signatures of sender and the receiver are submitted before the time $t$. After time $t$, the unspent funds are transferred back to the receiver. Embedding such instructions into the funds is commonly referred to as a smart contract. Our smart contract automates the claim-or-refund functionality. The funds are transferred either when the time of the agreement expires or when the signatures of both sender and receiver are available.

The scriptPubKey that receiver uses in the contract is

```
IF
```

OP_CHECKLOCKTIMEVERIFY OP_DROP
$pk_R$ OP_CHECKSIGVERIFY
ELSE
OP_2 $pk_R$ $pk_S$ OP_2
OP_CHECKMULTISIGVERIFY
ENDIF

**Oracles.** Based on the condition specified by the user, the smart contract can indeed interact with the parameters outside the cryptocurrency system. Systems like Ethereum support *Oracles* [109] which interact with the outside world with different APIs for information like weather parameters etc. Depending on the trust imposed on the oracles by the user and the agents, they can agree on the oracles and the value of the deposit before the start of the protocol.

### 2.5 Monotone boolean formula for majority

Majority function [110] of $n$ variables with values in $\{0,1\}$ is defined as taking the value 1 if at least $n/2$ number of variables are 1 and 0 otherwise. Let $\{x_{\mathrm{i}}\}_{\mathrm{i}=1}^{n}$ be the $n$ variables over which Majority function $Maj(\cdot)$ is being computed, then

$$Maj(x_1, x_2, \cdots, x_n) = \begin{cases} 1 & \text{if } \sum_{\mathrm{i}} x_{\mathrm{i}} \geq \frac{n}{2};\ x_{\mathrm{i}} \in \{0,1\} \\ 0 & \text{if otherwise} \end{cases}$$

While majority function of $n$ variables can be realized using non-monotone circuits of size $O(\log n)$, monotonicity places restrictions on the circuit that the circuit should only be realized using AND and OR gates (but not NOT) gates. Valiant [110] first proved that a polynomial size *monotone* circuit is realizable for majority circuit and provided a construction of size $O(n^{5.3})$. Subsequent works like one by Hoory [111] discuss majority circuits and realize threshold structures using majority circuit. Boppanna [112] showed that $O(t^{4.3}n)$ is the optimal upper bound on the majority circuit over $n$ variables for a threshold $t$. Hooray [111] further improved the size of the circuit to $O(n^{1+\sqrt{2}})$ while keeping the circuit depth at $O(\log n)$. Goldreich [113] provided an exposition of Valiant's approach to the majority circuit

construction, a probabilistic proof while using a different probability amplifier (majority-3) than the one used by Valiant.

We briefly explain the construction provided in [113]:
Let the $n$ variables be $x_{\mathrm{i}} \in \{0,1\}$, $\mathrm{i} \in [n]$. Generate $m$ random variables $y_{\mathrm{j}}, \mathrm{j} \in [m]$ by uniform randomly sampling an index among $[n]$ and assigning the corresponding $x_{\mathrm{i}}$ value to each $y_{\mathrm{j}}$ sequentially. When $\Pr(z_{\mathrm{i}} = 1) = p$ for each $\mathrm{i} \in [3]$, the probability that the majority function is 1 is given by $\Pr(MAJ_3(z_1, z_2, z_3)) = 1$ is $3(1-p)p^2+p^3$. If $p = 0.5+\epsilon, \epsilon \leq \epsilon_0 < 0.5$, then $p' \geq 0.5 + (1.5 - 2\epsilon_0^2)\epsilon$. Thus the bias of $\epsilon$ is increased by the factor $(1.5 - 2\epsilon_0^2)$ for each level of the tree. When the number of ones in the initial set of variables $x_{\mathrm{i}}$ is $\frac{n}{2} + 1$, the bias of the variables $y_{\mathrm{i}}$ at the lowest level of the tree would be $\frac{1}{n}$. This bias is increased in three steps: First the bias is brought to a constant ($< \frac{1}{2}$) using $\ell_1$ layers of the tree, then that constant is increased further to be close to 1 using $\ell_2$ layers, finally the probability of majority function being 1 when there is majority in the initial value is taken arbitrarily close to 1, in other words, the probability of function returning 0 when there is majority is made negligibly small $< 2^{-n}$ in another $\ell_3$ layers of the circuit. When using majority circuit, using $p = 0.5$ for a given $n$, when $MAJ_3$ nodes are used as probability amplifiers, this would result in a circuit depth of $\ell_1 + \ell_2 + \ell_3 \sim 2.71 \log n$. When $MAJ_3$ is expanded using fan-in 2 gates, we have a circuit implemented using only gates with fan-in 2. This would result in a total circuit size of $O(n^{5.3})$.

### 2.6 Boolean formula and distribution matrix

The circuit is represented as a boolean formula by expanding $MAJ_3$ $(z_1, z_2, z_3)$ as $(z_1 \wedge z_2) \vee (z_2 \wedge z_3) \wedge (z_1 \vee z_3)$, resulting in a monotone boolean formula computing majority/threshold function. This formula is then used to compute the distribution matrix of the linear integer secret sharing scheme (LISS). The Benolah-Leichter (BL) [94] construction of converting a monotone boolean formula is briefly recollected here.

Consider Boolean functions $f_{OR} = f_1 \vee f_2$ and $f_{AND} = f_1 \wedge f_2$ where $f_1, f_2$ are either Boolean functions or literals. Let $M_a$ and $M_b$ are share distribution matrices of $f_1$ and $f_2$ respectively. The share distribution matrices of $f_{OR}, f_{AND}$ are computed as $M_{OR}, M_{AND}$ as

$$M_{AND} = \begin{array}{|c|c|c|c|} \hline C_a & C_a & R_a & 0 \\ \hline 0 & C_b & 0 & R_b \\ \hline \end{array}$$

$$M_{OR} = \begin{array}{|c|c|c|} \hline C_a & R_a & 0 \\ \hline C_b & 0 & R_b \\ \hline \end{array}$$

**Figure 2.4.** Share distribution matrix for OR and AND functions

shown in Figure 2.4, where $C_a$ is the first column of matrix $M_a$ and $R_a$ is the rest of the matrix except the first column of matrix $M_a$. Similarly $C_b, R_b$ are the first column of matrix $M_b$ and the rest of the matrix except the first column of matrix $M_b$ respectively. If the function contains only one literal, it is taken just as column i.e., for any literal $f_1 = x_{\mathrm{i}}$, the matrix is just [1] with $C_a = 1$ and no $R_a$.

# 3. Pepal: PENALIZING PARTIAL LEAKAGES IN A TWO-PARTY ESCROW MECHANISM

## 3.1 Problem Definition

We consider a scenario where a sender wishes to forward a private multimedia document $M$ to a receiver. The receiver is expected to hold a public key-secret key pair $(pk, sk)$, where the key $sk$ is a signing key of a (say) Bitcoin wallet corresponding to $pk$. Instead of the sender directly sending $M$ to the receiver, we expect the sender and receiver to jointly compute a function $f((M, pk), sk)$ which should provide the receiver a version $M_{sk}$ of $M$ that has been tagged (or robustly watermarked) with the key $sk$. The protocol should abort (or not produce a meaningful $M_{sk}$) if $sk$ from the receiver and $pk$ from the sender are not a matching key pair. At the end of the protocol, the sender does not learn $sk$ or $M_{sk}$ and the receiver does not learn any further information about $M$. A cryptocurrency wallet holds the receiver's escrow deposit for accountability.

We consider the problem in a mutually distrustful setting, and *either* the sender or the receiver can be malicious. A malicious sender can try to learn the signing key of the receiver so as to steal the deposit. When appropriate, he can also make the document public and try to accuse the receiver of dishonest disclosure. The malicious receiver, on the other hand, can try to remove/replace the watermark from the obtained document, and release the modified version to the public without revealing her key. In such an adversarial setting, we wish to satisfy the following privacy and integrity goals:

- Sender Privacy: Before the transfer completes, no information regarding the document is available to the receiver.
- Receiver Privacy: Before the disclosure of document by the receiver, no information regarding the receiver's signing key is available to the sender.
- Sender Integrity: In case of false accusation by the sender, no action is taken.
- Receiver Integrity (Revealing property): In case of disclosure of the document by the receiver, the signing key of the receiver is revealed to the sender.

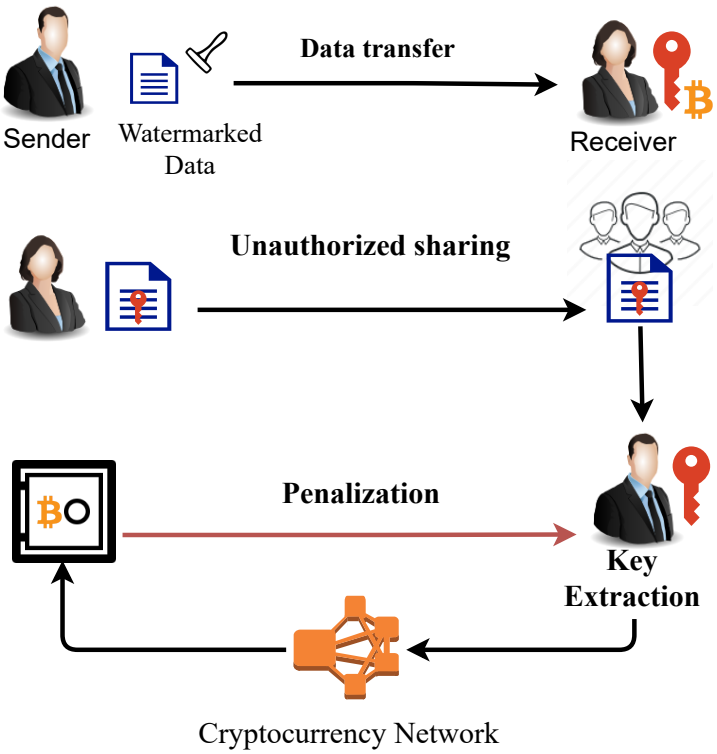



**Figure 3.1.** Pepal protocol's high-level view: The sender transfers a watermarked version of the document to the receiver. The watermark is the secret key of the receiver unknown to the sender. Upon unauthorized sharing publicly, the sender extracts the watermark (secret key) and penalizes the receiver by transferring the pre-setup deposit.

We formalize these properties as an ideal functionality in Figure 3.8 in Section 3.6.

**Solution Overview.** We propose the Pepal protocol, depicted in Figure 3.1 involving the two parties *Sender* and *Receiver*. The sender has the multimedia document $M$ and the receiver has the signing key $sk$. The receiver initially makes a time-locked bitcoin deposit of an agreed value of funds that can be opened only if the signing-key of the receiver is available. The sender divides the document into several blocks and creates two watermarked versions (corresponding to 0 and 1) for *each block*. The parties run multiple *1-out-of-2* Oblivious Transfer ($\mathsf{OT}_1^2$) protocol instances, one for transfer of each of the document blocks. The sender uses the watermarked blocks as inputs while the receiver uses each of the bits of his signing key as choice bits for the $\mathsf{OT}_1^2$ s and obtains one version of each block i.e., for a 256-bit signing key of the receiver, the sender (in the simplest case) divides the document into 256

blocks and creates two versions for each block using *robust watermarking*. The sender and receiver then perform 256 $\mathsf{OT}_1^2$ s, where the choice bit for each $\mathsf{OT}_1^2$ is each of the bits of 256-bit key of the receiver. The receiver also proves to the sender in zero knowledge that the signing key used for the deposit is indeed formed of the bits used for $\mathsf{OT}_1^2$ s.

As the document is transferred through oblivious transfer, the sender can not gain any information about the signing-key of the receiver. However, if the document is revealed/disclosed before the time of expiry of the agreement, the sender learns the signing key of the receiver from the watermark of the revealed document. He can then proceed to penalize the malicious receiver by transferring the funds to himself. The multiple $\mathsf{OT}_1^2$ s, one for each block, ensure that the watermark embedded in the document corresponds to the signing-key bits.

To transfer the funds out of the deposit, the sender needs both his and the receiver's signature which can not be obtained before the document is revealed to the public. Thus, he can penalize the receiver only if she is dishonest. If the receiver is honest, the agreement would expire after the agreed time and the funds will be transferred back to her.

The receiver instead of full disclosure, can disclose the document partially to the public. She can reveal, say, half of the total 256 blocks received, so that only half the number of bits of her signing-key are revealed to the sender. However, for a 256-bit key of the receiver, the sender can in-fact divide the document into more numbers of blocks than just 256. This way, he can embed the key multiple times in the document, for example, the sender can divide it into 512 parts so that the key gets embedded twice. The sender can perform 512 $\mathsf{OT}_1^2$ s with the receiver using her 256-bit key twice for the same. In such a scenario, the sender can extract more number of bits upon partial disclosure. Also, the information in the document may not be "uniform" throughout the document, so the sender can also try to embed the key multiple times in a document part where there is "more" information by dividing it into more number of parts at those document locations.

The receiver understands that one bit of her signing key is watermarked in each of document blocks received using that bit in $\mathsf{OT}_1^2$. She also knows which particular bit is embedded in a particular document block, this is because, the watermark embedded in a block is same as the choice bit used for $\mathsf{OT}_1^2$ in obtaining a document block. Leveraging this

knowledge, the receiver can try to minimize the number of bits revealed to the sender. For example, with the sender dividing the document into 512 blocks and the receiver having a 256-bit signing-key, the receiver can reveal 100 blocks of the received document revealing only 50 bits to the sender. She can achieve this by revealing two blocks received with each bit for 50 bits. To prevent such an attack we propose a primitive called *Doubly Oblivious Transfer* (DOT). DOT prevents the receiver from learning which bit (index of the bit) of her key is watermarked into a certain block.

In DOT the sender has two messages $m_0, m_1$ and the receiver has two bits $s_0, s_1$ (refer Figure 3.2). The sender has an extra choice bit $c$ using which he transfers $m_{s_c}$ (associated with the bit $s_c$) to the receiver. At the end of DOT instance, the receiver cannot determine the value of $c$ and $m_{1-s_c}$ and the sender does not know the bit $s_c$ that has been used in the transfer of $m_{s_c}$.[1] Refer Figure 3.2 for the pictorial depiction of the simplest form of DOT protocol.

For Pepal, the sender can use DOT to transfer the document to the receiver such that she has no information about which of her bits is embedded in a certain document block. As we analyze in Section 3.6.1, this greatly improves the expected number of bits revealed to the sender in case of partial disclosure. For example, with the sender dividing the document into 512 blocks and 256-bit key at the receiver, upon disclosure of 100 blocks, the expected number of bits that the sender can extract is 90.3 instead of 50 while using just oblivious transfer.

Notice that our Pepal protocol augments cryptographic primitive with a smart contract. Given the limited expressibility of Bitcoin contracts our (off-chain) cryptographic solution seems necessary but this may not be the case for turing complete systems like Ethereum [114]. However, defining the complete solution as a smart contract will not be or may not remain inexpensive enough. Further comments regarding the contracts can be found in 3.8 .

The Pepal protocol uses a robust watermarking scheme to watermark either the bit 0 or the bit 1. The actual watermarking scheme varies depending on the type of the data being watermarked. While theoretically, an algorithm may exist which can remove the watermark

[1]↑ For $s_0 = s_1 = b$, the receiver knows that she received $m_b$; however, that does not constitute any privacy leakage in our application as $c$ and $m_{1-s_c}$ remain private.

from the data, we just require that such an algorithm should not be available or known to humans; this approach was formalized by Rogaway[107].

### 3.2 Related work

A closely related subject to penalizing data breaches, one that is well-studied, is traitor tracing [115], [116]. In a traitor tracing scheme, decryption boxes with unique private keys (for a common public key) are distributed to a number of subscribers. If a device is reverse engineered and the key is leaked, the device it came from can be determined by the service provider.

Kiayias and Tang [117] adds a Bitcoin smart contract to hold a bond that is recoverable. This body of work has limited applicability to our Pepal problem for three main reasons: (1) we want to detect leaked documents that have been meaningfully written, not keys which are arbitrary, random values; (2) we want the entity distributing the values to not learn the value until it is leaked; and (3) unlike in the smart contract variant [117], we cannot have the provider provision the signing key for use by both parties. For these reasons, we do not build our solution from traitor tracing schemes.

In another line of work, Nasir *et al.* build a seller-buyer watermarking scheme in [118] where the watermark embedded in the document is not known to seller/sender but can identify the buyer once the document is distributed. The main drawback of their scheme is the requirement of a third trusted authority for providing the watermark for the buyer, also the sender needs to go through the legal procedure and prove to the judge that the buyer is indeed the one who leaked and the penalization is through court system.

In [119], Andre *et al.* propose a zero-knowledge proof based protocol for providing proof of ownership of the document but does not involve proving that a certain party is the leaker or a way to penalize the leak.

Using bitcoin contracts for collatorizing the fair and correct execution of cryptographic protocols has been explored earlier [26], [27], [120]. Our bitcoin contract is a standard claim-or-refund transaction common in this literature. The main difference is that one party must

prove that the singing key used in this transaction is consistent with the one taken as input to a private computation.

### 3.3 Doubly Oblivious Transfer — DOT

**Oblivious Transfer.** 1-out-of-2 oblivious transfer ($\mathsf{OT}_1^2$) is a two-party (a sender and a receiver) computation mechanism, where the sender has two messages $M_0$ and $M_1$ and the receiver has a bit $b \in \{0, 1\}$. The goal is to transfer $M_b$ to the receiver and at the end of the protocol, the receiver should not learn any information about $M_{1-b}$ and the sender should not learn $b$. We consider the oblivious transfer protocol, called the Verified Simplest OT by Doerner *et al.* [22] which is an extended version of OT protocol by Chou et.al. [95], recalled in 2.1.3 along with Figure 2.1.

The multiplicative group $\mathsf{G}$ used for the protocol is Gap-DH [121] and the additional verification step forces the receiver to make oracle queries before receiving the encryptions from the sender, there by making the protocol UC-Secure.

In our solution, the receiver obtains the document blocks by running $\mathsf{OT}_1^2$ multiple times with her signing key bits as the choice bits. However, while running $\mathsf{OT}_1^2$, the receiver understands that each of the message that is received by using choice bit is indeed affected by the choice bit i.e., the receiver knows the index of the bit embedded through watermark in a received message/document block.

To overcome this, we propose a primitive, in which the receiver, after giving multiple bits as input, receives several messages corresponding to the input bits, but the receiver does not have any information about which bit was used as choice bit for choosing a certain message. In the simplest case the sender has two messages $m_0, m_1$ along with a choice bit $c$ and the receiver has two bits $s_0, s_1$ as depicted in Figure 3.2. The sender chooses one of the indices of the bits of the receiver using the bit $c$ and the receiver receives the message $m_{s_c}$ corresponding to the bit of the chosen index. Here, the sender does not know which message has been received by the receiver and the receiver does not know which of her two bits is chosen as the choice bit to choose the messages. Hence we call it *Doubly Oblivious Transfer* ($\mathsf{DOT}$) protocol.

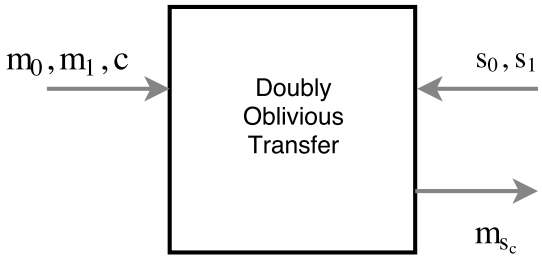



**Figure 3.2.** Doubly Oblivious Transfer Primitive: the sender has two input messages $m_0, m_1$ and a bit $c$. The receiver has two input bits $s_0, s_1$ and obtains $m_{s_c}$.

**Functionality $\mathcal{F}_{\mathsf{DOT}}$**

Ideal functionality $\mathcal{F}_{\mathsf{DOT}}$ interacts with sender $\mathsf{S}$ and receiver $\mathsf{R}$. The sender has two messages $M_0, M_1$ and a choice bit $c$. The receiver has two bits $s_o, s_1$. The adversary $\mathcal{A}$ corrupts either the sender or receiver.

- Upon receiving the message $(\texttt{inputS}, M_0, M_1, c, sid)$ with $M_0, M_1 \in \{0,1\}^*$, $c \in \{0,1\}$ from sender $\mathsf{S}$, record $\langle \mathsf{S}, M_0, M_1, c, sid \rangle$, forward the message $(\texttt{intd}, sid)$ to the receiver $\mathsf{R}$ and $(\texttt{input}, sid)$ to $\mathcal{A}$.
- Upon receiving the message $(\texttt{inputR}, s_0, s_1, sid)$ with $s_0, s_1 \in \{0,1\}$ from receiver $\mathsf{R}$, record $\langle \mathsf{R}, s_0, s_1, sid \rangle$, forward the message $(\texttt{input}, sid)$ to $\mathcal{A}$.
- Upon receiving the message $(\texttt{deliverS}, sid)$ from $\mathcal{A}$, check if $\langle \mathsf{R}, s_0, s_1, sid \rangle$ is stored, else ignore the message. Send $(\texttt{delivered}, sid)$ to $\mathsf{S}$.
- Upon receiving the message $(\texttt{deliverR}, sid)$ from $\mathcal{A}$, check if $\langle \mathsf{S}, M_0, M_1, c, sid \rangle$ is stored, else ignore the message. Forward $(\texttt{output}, M_{s_c}, sid)$ to $\mathsf{R}$.

**Figure 3.3.** Ideal Functionality of $\mathsf{DOT}$

Figure 3.3 represents the ideal functionality of the $\mathsf{DOT}$ protocol. The functionality $\mathcal{F}_{\mathsf{DOT}}$ interacts with the sender $\mathsf{S}$ and receiver $\mathsf{R}$. The adversary $\mathcal{A}$ controls the communication and the delivery of the messages. When the sender and receiver forward their inputs, they use the tags `inputS, inputR` respectively. The functionality delivers the corresponding messages to sender and receiver on receiving the messages `deliverS, deliverR` from the adversary. $\mathsf{DOT}$ hides the index $c$ and $m_{1-s_c}$ from the receiver, but it need not essentially hide the value $s_c$ itself. For $s_0 = s_1 = b$, the receiver knows the value $b$ but not $c$.

| **Sender** | | **Receiver** |
|---|---|---|
| Message blocks: $M_0$ and $M_1$, Choice bit: $c$ | | Bits: $s_0, s_1$ |
| | **Setup** | |
| | Multiplicative (Public) Group $\mathbb{G}$, Generator $g$ | |
| $(pk_S, sk_S)$ | | $(pk_R, sk_R)$ |
| | $pk = pk_S * pk_R$ | |
| | For all i : $0 \leq \text{i} \leq 1$, execute the steps below | |
| | **Symmetric encryption of Message Blocks** | |
| $g_0, g_1 \leftarrow_R \mathbb{G}$ | | |
| $Enc_0 = E_{H(g_0)}(M_0), Enc_1 = E_{H(g_1)}(M_1)$ | | |
| $\widehat{Enc}_{\text{i}} = \pi_1(Enc_{\text{i}})$, for permutation $\pi_1$ | $\xrightarrow{\widehat{Enc}_{\text{i}}}$ | |
| | **El-Gamal encryption of group elements** | |
| Set $g_{c,0} = g_0, g_{c,1} = g_1$ | | |
| $g_{1-c,0}, g_{1-c,1} \leftarrow_R \mathbb{G}$ | | |
| $u_{\text{i},0} = \mathcal{E}_{pk}(g_{\text{i},0})$, $u_{\text{i},1} = \mathcal{E}_{pk}(g_{\text{i},1})$ | | |
| | **Oblivious Transfer** [22] | |
| | Run $\mathsf{OT}_1^2$ for i | |
| Input $u_{\text{i},0}, u_{\text{i},1}$ | | Input $s_{\text{i}}$ |
| | | Output $u_{\text{i},s_{\text{i}}}$ |
| | **Re-randomization, Forwarding and Decryption** | |
| | | $v_{\text{i},s_{\text{i}}} = \mathcal{R}_{pk}(u_{\text{i},s_{\text{i}}})$ |
| | $\xleftarrow{v_{\text{i},s_{\text{i}}}}$ | |
| $x_{\text{i},s_{\text{i}}} = \mathcal{D}_{sk_S}(v_{\text{i},s_{\text{i}}})$ | $\xrightarrow{x_{c,s_c}}$ | |
| | | $g_{c,s_c} = \mathcal{D}_{sk_R}(x_{c,s_c})$ |
| | | Decrypt $\widehat{Enc}_{s_c}$ using $H(g_{c,s_c})$ appropriately |

**Figure 3.4.** Doubly Oblivious Transfer ($\mathsf{DOT}$) Protocol

Each session of the protocol run is identified by a session id *sid*. The sender $\mathsf{S}$ forwards the two messages $M_0, M_1$ and the choice bit $c$ along with the tag `inputS` to the functionality indicating the input from sender $\mathsf{S}$. Upon the initiation of the session by the sender, the functionality informs the receiver $\mathsf{R}$ by forwarding the `intd` message along with the session id *sid*. The session initiation is also intimated to the adversary $\mathcal{A}$. Upon the initiation of the session, the receiver forwards the two bits $s_0, s_1$ to the functionality along with the session id to the functionality. The adversary controls the delivery of messages from the functionality, this is modelled by the `deliverS, deliverR` messages from the adversary. On receiving the

`deliverS` message with the session id *sid*, the functionality checks if the a corresponding previous input from the receiver is received, if yes, the sender is intimated that the receiver input has been received by forwarding the `delivered` message to the sender. Finally upon receiving the `deliverR` message along with *sid* from the adversary, the functionality checks if it received the input from the sender for that session id and if yes forwards the message $M_{s_c}$ to the receiver.

**Construction.** We provide a construction which realizes the ideal functionality of $\mathsf{DOT}$ with two messages $M_0, M_1$ and a choice bit $c$ at the sender and two bits $s_0, s_1$ at the receiver as given in the Figure 3.3. Both the parties possess public key-secret key pairs (refer Figure 3.4) and $pk = pk_S * pk_R$ where $pk_S, pk_R$ are public keys of sender and receiver. The sender samples two elements from the group (can be points from the elliptic curve), encrypts the two messages using a symmetric encryption $E_{(.)}(.)$ with the keys obtained by hashing the elements. These encryptions are randomly permuted and forwarded to the receiver in the form of $\widehat{Enc}_{\mathrm{i}}$. This is the first step in $\mathsf{DOT}$. The sender then transfers the elements to the receiver such that the receiver can only decrypt $M_{s_c}$. The encryption and forwarding of messages prevents the need to map random message strings onto group elements for the ElGamal encryption in the next step.

The sender samples two more elements, populates $g_{\mathrm{i,j}}$, $\mathrm{i,j} \in \{0,1\}$ as shown in Figure 3.4 and encrypts all $g_{\mathrm{i,j}}$ to the public key $pk$ using $\mathcal{E}_{pk}(.)$ - a Re-randomizable encryption like ElGamal encryption to obtain $u_{\mathrm{i,j}}$. Now two $\mathsf{OT}_1^2$ instances are run, one for each i with $u_{\mathrm{i,j}}$ as inputs. The receiver inputs $s_{\mathrm{i}}$ as the choice bit for the instance i of $\mathsf{OT}_1^2$.

Here the sender $\mathsf{S}$ initiates the protocol, this would correspond to the `inputS` message of the functionality. Once the protocol is initiated, the receiver inputs the bits $s_0, s_1$ into the $\mathsf{OT}_1^2$ protocol instance. This would correspond to the `inputR` message of the ideal functionality messages where the receiver forwards the bits $s_0, s_1$.

The encryption of the elements to the key $pk$ later helps the receiver to hide which keys have been obtained by her through $\mathsf{OT}_1^2$ and helps the sender to hide the order in which the keys have been forwarded. Hiding the order implies hiding the mapping between bits $s_{\mathrm{i}}$ and elements obtained by the receiver through $\mathsf{OT}_1^2$. The receiver after receiving the different

$u_{\mathrm{i},s_\mathrm{i}}$ through $\mathsf{OT}_1^2$ proceeds by applying $\mathcal{R}_{pk}(.)$, a re-randomization operation to obtain $v_{\mathrm{i},s_\mathrm{i}}$. These re-randomized encryptions of obtained encrypted elements are now forwarded back to the sender. If there was no re-randomization step, the sender would know what elements have been obtained by the receiver and so will know what version of the message was taken by the receiver. Hence we use the re-randomization step to hide from the sender, information regarding which messages have been obtained by the receiver through $\mathsf{OT}_1^2$. The sender from $v_{\mathrm{i},s_\mathrm{i}}$, decrypts his layer of ElGamal encryption using the decryption operation $\mathcal{D}_{sk_S}(.)$ to obtain $x_{\mathrm{i},s_\mathrm{i}}$. He then drops $x_{1-c,s_\mathrm{i}}$ and forwards only the element $x_{c,s_c}$ to the receiver. The element $x_{c,s_c}$ (which at this point is only encrypted to the receiver's public key) is then decrypted by the receiver using her private key using $\mathcal{D}_{sk_R}(.)$ to obtain the element $g_{c,s_c}$. The key obtained as hash of $g_{c,s_c}$ is used to decrypt the initially obtained random permutation of messages. Only one of them gets decrypted *correctly*. The receiver, while decrypting the encrypted messages, would not know which message is the correct encryption using the obtained key, she tries to decrypt each of the messages. For the receiver to be able to recognize the correct message for the key, we need a mechanism.

To achieve the decryption and identification of the correct message block by the receiver, the sender initially appends each of the messages with a string which is obtained as a certain public function $\widehat{f}(.)$ of key (like hash of the key) used to encrypt the message before the encryption process. After decrypting each block with the key, the receiver matches the appended string with the locally calculated string using $\widehat{f}(.)$ of the key. Whichever message has the correct match, is the correct message. Thus the receiver decrypts $M_{s_c}$.

Imagine the case when the initial encryptions are not permuted, then the receiver knows that the the encryptions received correspond to bit indices 0 and 1 in that order, so she can try to attack the system by setting one of the bits, say $s_0 = 0$ and the other $s_1 = 1$, then which ever encryption gets decrypted, will reveal which of the two $s_\mathrm{i}$s has been chosen by the sender. To prevent such a scenario, the initial permutation of the encryptions is necessary.

Before we prove the security of the $\mathsf{DOT}$ protocol, we briefly introduce the functionality to prove the knowledge of discrete logarithm in zero-knowledge and the definition of UC-security.

**Functionality $\mathcal{F}_{ZK}^{DL}$ [22].** The functionality is parameterized by group $\mathbb{G}$ and runs with two parties $P_1$ and $P_2$. The parties can be sender $\mathsf{S}$ and receiver $\mathsf{R}$.

*Proof*: On receiving $(\mathsf{prove}, a, \mathsf{g})$ where $a \in \mathbb{Z}_q, \mathsf{g} \in \mathbb{G}$ from party $P_\mathrm{i}$, store this message. On receiving, $(\mathsf{prove}, h, \mathsf{g})$ from party $P_\mathrm{j}$, where $h, \mathsf{g} \in \mathbb{G}$, if $h = \mathsf{g}^a$, send $(\mathsf{accept})$ to $P_\mathrm{j}$, otherwise send $\mathsf{fail}$ to $P_\mathrm{j}$.

The parties can be sender $\mathsf{S}$ and receiver $\mathsf{R}$. The parties use the functionality $\mathcal{F}_{ZK}^{DL}$ to prove in zero-knowledge, that they own the secret keys of the corresponding public keys.

**Definition 3.3.1.** *A protocol $\rho$ UC-realizes an ideal functionality $\mathcal{F}$ if for any adversary $\mathcal{A}$, there exists a simulator such that for any environment $\mathcal{E}$, the ensemble $\mathsf{EXEC}_{\rho,\mathcal{A},\mathcal{E}}$ and $\mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are computationally indistinguishable.*

**Theorem 3.3.1.** *The $\mathsf{DOT}$ protocol securely implements the functionality $\mathcal{F}_{\mathsf{DOT}}$ under the following conditions:*

***Corruption Model:*** *Static corruption (the sender or receiver is corrupted at the beginning of the protocol).*

***Hybrid Functionalities:*** *$H$ is modelled as a random oracle and secure channels between the parties are assumed.*

***Computational Assumption:*** *The encryption scheme used in the initial step is symmetric, non-committing and robust [95]. Group used for $\mathsf{OT}_1^2$ module $\mathsf{G}$ is a Gap-DH group.*

*Proof.* We prove the security of $\mathsf{DOT}$ by constructing a simulator which generates an indistinguishable view in the real world - ideal world paradigm for the adversary. The parties use the functionality $\mathcal{F}_{ZK}^{DL}$ to prove in zero-knowledge, that they own the secret keys of the corresponding public keys.

**Malicious Sender.**

- Receive $(\mathsf{prove}, sk_\mathsf{S}, pk_\mathsf{S})$ on behalf of $\mathcal{F}_{ZK}^{DL}$. On accepting, forward $\mathsf{accept}$ to the sender, else abort.
- Answer all oracle queries of the sender randomly and store the query and reply pairs in the form of $(q_k, r_k)$.

- Receive the encrypted messages $\widehat{Enc}_{\mathrm{i}}$, $\mathrm{i} \in \{0,1\}$ from the sender and participate in oblivious transfer for the next step.
- Set the bits $s_{\mathrm{i}}$, $\mathrm{i} \in \{0,1\}$ randomly with values from $\{0,1\}$ as choice bits before participating in the $\mathsf{OT}_1^2$ protocol.
- For $\mathsf{OT}_1^2$ part of the protocol, invoke multiple instances corrupted sender phase of the simulator of the UC-secure OT [22] developed by Chou *et al.* [95], [121] (call it, $\mathcal{S}_{OT}$). The simulator $\mathcal{S}_{OT}$ extracts the sender inputs for each of the instances; obtain the inputs.
- Perform the operations like an honest receiver. Receive the elements $u_{\mathrm{i},s_{\mathrm{i}}}$ and try to decrypt (own layer of encryption, the sender is expected to encrypt the messages with $\mathcal{E}_{pk}(.)$).
- If any of the received elements results in an error during decryption, abort. Else, re-randomize the encryption using $\mathcal{R}_{pk}(.)$ to obtain $v_{\mathrm{i},s_{\mathrm{i}}}$ and forward them back to the sender. Receive an encrypted group element as $x_{c,s_c}$, try to decrypt and hash it to obtain the decryption key. Decrypt one of the received messages with the obtained key. If it results in an error, abort.
- Decrypt the initial $\widehat{Enc}_{\mathrm{i}}$ as follows: for each $\mathrm{i}, k$, from the initially stored pairs $(q_k, r_k)$, perform $Dec_{r_k}(\widehat{Enc}_{\mathrm{i}})$. The first value that gets decrypted meaningfully is set as $M_{\mathrm{i}}$ for any i. If no key $r_k$ decrypts meaningfully, set $M_{\mathrm{i}} = \perp$.
- Obtain the choice bit $c$ of the sender as follows: during the $\mathsf{OT}_1^2$ protocol, the simulator $\mathcal{S}_{OT}$ extracts the message inputs of the sender side [95] and forwards them to $\mathcal{S}_{DOT}$. For each $\mathsf{OT}_1^2$ instance i, $\mathcal{S}_{\mathsf{DOT}}$ receives two messages $g_{\mathrm{i},0}, g_{\mathrm{i},1}$ from $\mathcal{S}_{OT}$, the simulator $\mathcal{S}_{\mathsf{DOT}}$ stores all the elements in the form of $g_{\mathrm{i,j}}$. For each i, the simulator checks which of the elements $g_{\mathrm{i,j}}$, $\mathrm{j} \in \{0,1\}$, matches with the decrypted element (obtained from sender in the last step of the protocol). Whenever a match is seen, $c$ is set to i.
- Forward the messages $M_{\mathrm{i}}$, $\mathrm{i} \in \{0,1\}$ and choice bit $c$ to the ideal functionality $\mathcal{F}_{\mathsf{DOT}}$.

The adversary can not distinguish between a real world view and simulated view owing to the following facts: the simulator $\mathcal{S}_{OT}$ is UC-Secure [22]; ElGamal encryption offers semantic

security when DDH is hard; the real world honest receiver's output will be different only if the simulator decrypts the encryptions received to a different value apart from the ones used by the sender, but this happens with a negligible probability owing to the robustness of the encryption scheme.

**Malicious Receiver.**

- Receive $(\mathsf{prove}, sk_{\mathsf{R}}, pk_{\mathsf{R}})$ on behalf of $\mathcal{F}_{ZK}^{DL}$. On accepting, forward $\mathsf{accept}$ to the receiver, else abort.
- Generate two strings $C_1 \leftarrow \mathcal{A}_1(1^\lambda)$ and $C_2 \leftarrow \mathcal{A}_1(1^\lambda)$ and forward to the receiver.
- Sample four group elements $g_{\mathrm{i,j}}$ for $\mathrm{i,j} \in \{0,1\}$ and encrypt them using ElGamal encryption $\mathcal{E}_{pk}(.)$ to obtain $u_{\mathrm{i,j}}$.
- Performs two instances of $\mathsf{OT}_1^2$ and use $u_{\mathrm{i,j}}$ as inputs for instance i of $\mathsf{OT}_1^2$.
- The receiver inputs $s_{\mathrm{i}}$ to the $\mathsf{OT}_1^2$ instance i. For the $\mathsf{OT}_1^2$ protocol, the simulator invokes the corrupted receiver phase of simulator of Verified Simplest Oblivious Transfer [22] (call it $\mathcal{S}_{OT}$).
- Obtain re-randomized elements $v_{\mathrm{i},s_{\mathrm{i}}}$, decrypt own layer of encryption using $\mathcal{D}_{sk_S}()$ to obtain $x_{\mathrm{i},s_{\mathrm{i}}}$ and forward $x_{c,s_c}$ for a randomly chosen bit $c$.
- Answer all oracle queries randomly except at the points $g_{\mathrm{i,j}}$. When queried on any of the points $g_{\mathrm{i,j}}$, sends the bits $\mathrm{j,j}$ to the functionality and obtain the message $m'$.
- Reply to the query with a key $k \leftarrow \mathcal{A}_2(C_p, m')$ where $p$ is uniformly picked from $\{1,2\}$ for every instance of the simulation.

The receiver can not distinguish the real and simulated view. This is because: ElGamal encryption offers semantic security when DDH is hard, $\mathsf{OT}_1^2$ used is UC-secure [22] and the fact that when the simulator does not abort, the indistinguishability holds from non-committing property of the encryption scheme. The UC-security of the $\mathsf{DOT}$ follows from Definition 3.3.1.

□

### 3.4 Generalization of DOT protocol

The DOT protocol can be easily extended to work with multiple messages at the sender and $\kappa$-bit signing key of the receiver as shown in Figure 3.5.

| **Sender** | | **Receiver** |
|---|---|---|
| | **Setup** | |
| | Multiplicative (Public Group) $\mathbb{G}$, Generator $g$ | |
| $(pk_S, sk_S)$ | | $(pk_R, sk_R)$ |
| | $pk = pk_S * pk_R$ | |
| | For all i : $0 \leq \mathrm{i} \leq \kappa - 1$, j $\in \{0, 1\}$ execute the following steps | |
| Message blocks: $M_{\mathrm{i,j}}$ | | Bits: $s_k$ for $0 \leq k \leq \kappa - 1$ |
| $g_{\mathrm{i,j}} \leftarrow_R \mathbb{G}$ | | |
| $Enc_{\mathrm{i,j}} = E_{H(g_{\mathrm{i,j}})}(M_{\mathrm{i,j}})$ | | |
| $\widehat{Enc}_{\mathrm{i,j}} = \boldsymbol{\pi}_1(Enc_{\mathrm{i,j}})$ for permutation $\boldsymbol{\pi}_1$ | $\xrightarrow{\widehat{Enc}_{\mathrm{i,j}}}$ | |
| | **El-Gamal encryption of group elements** | |
| For each j, $\widehat{g}_{\mathrm{i,j}} = \boldsymbol{\pi}(g_{\mathrm{i,j}})$ for Permutation $\boldsymbol{\pi}$ | | |
| $u_{\mathrm{i,j}} = \mathcal{E}_{pk}(\widehat{g}_{\mathrm{i,j}})$ | | |
| | **Oblivious Transfer**[22] | |
| | (Run $\mathsf{OT}_1^2$ once for i) | |
| Input $u_{\mathrm{i,j}}$ | | Input $s_{\mathrm{i}}$ |
| | | Output $u_{\mathrm{i},s_{\mathrm{i}}}$ |
| | **Re-randomization, Forwarding and Decryption** | |
| | $\xleftarrow{v_{\mathrm{i},s_{\mathrm{i}}}}$ | $v_{\mathrm{i},s_{\mathrm{i}}} = \mathcal{R}_{pk}(u_{\mathrm{i},s_{\mathrm{i}}})$ |
| $x_{\mathrm{i},s_{\mathrm{i}}} = \mathcal{D}_{sk_S}(v_{\mathrm{i},s_{\mathrm{i}}})$ | | |
| $w_{\mathrm{i},s_{\mathrm{i}}} = \boldsymbol{\pi}^{-1}(x_{\mathrm{i},s_{\mathrm{i}}})$ | $\xrightarrow{w_{\mathrm{i},s_{\mathrm{i}}}}$ | $\widehat{g}_{\mathrm{i},s_{\mathrm{i}}} = \mathcal{D}_{sk_R}(w_{\mathrm{i},s_{\mathrm{i}}})$ |
| | | Decrypt $\widehat{Enc}_{\mathrm{i},s_{\mathrm{i}}}$ using $H(\widehat{g}_{\mathrm{i},s_{\mathrm{i}}})$ appropriately |

**Figure 3.5.** Doubly Oblivious Transfer Protocol (General Case)

In the general case, the sender has a total of $2\kappa$ messages $M_{\mathrm{i,j}}$, for $0 \leq \mathrm{i} \leq \kappa - 1, \mathrm{j} \in \{0, 1\}$ and the receiver has bits $s_n, 0 \leq n \leq \kappa - 1$. After participating in the protocol, the receiver receives $M_{\mathrm{i},l}, l = s_{\boldsymbol{\pi}(\mathrm{i})}$ for a permutation $\boldsymbol{\pi}$ of set of indices i chosen at the sender. The permutation of indices is the general case equivalent of the choice bit $c$ of the two bit case.

Forwarding a random permuted order of encrypted messages remains similar for the general case. When the elements are sampled in the general case, sampling extra elements

is not *necessary*. The sender performs a permutation $\pi$ on the rows i of the elements $g_{\mathrm{i,j}}$ to obtain $\widehat{g}_{\mathrm{i,j}}$ which are encrypted using $\mathcal{E}_{pk}(.)$ as before. Now, $\widehat{g}_{\mathrm{i,j}}$ are input to i instances of $\mathsf{OT}_1^2$ to which the receiver inputs $s_{\mathrm{i}}$ as the choice bits for each instance i. The receiver obtains $u_{\mathrm{i},s_{\mathrm{i}}}$, re-randomizes the encryption using $\mathcal{R}_{pk}(.)$ and sends back $v_{\mathrm{i},s_{\mathrm{i}}}$. After receiving $v_{\mathrm{i},s_{\mathrm{i}}}$, the sender reverses the permutation order to obtain $w_{\mathrm{i},s_{\mathrm{i}}} = \pi^{-1}(v_{\mathrm{i},s_{\mathrm{i}}})$. He then decrypts his layer of encryption using $\mathcal{D}_{sk_S}(.)$ and forwards $x_{\mathrm{i},s_{\mathrm{i}}}$ to the receiver who decrypts her layer of decryption to obtain $\widehat{g}_{\mathrm{i},s_{\mathrm{i}}}$. These $\widehat{g}_{\mathrm{i},s_{\mathrm{i}}}$ are hashed to obtain the final keys which are then used to decrypt the $\widehat{Enc_{\mathrm{i}}}$ received in the first step. Note that if the number of messages is not a multiple of $2\kappa$, the sender can sample extra elements and encrypt them to input them in $\mathsf{OT}_1^2$. After receiving the encrypted elements from the receiver, he can discard the elements at the indices where the extra elements have been placed in the $\mathsf{OT}_1^2$ step. Also, if the receiver tries to attack the protocol by manipulating the cipher texts after the re-randomization step, she will not be able to receive meaningful keys for the correct decryption, she can gain no information regarding the sender's messages or permutation applied on encrypted messages.

### 3.5 Committed Receiver Oblivious Transfer

An oblivious Transfer instance transfers one message $M_b$ where $b \in \{0, 1\}$ of the two messages $M_0$ and $M_1$ from the sender to the receiver with bit $b$. In our protocol which uses $\mathsf{DOT}$ (which in-turn uses $\mathsf{OT}_1^2$), we further *require* the bit $b$ to be a bit of the signing-key of the receiver. With a simple $\mathsf{OT}_1^2$, the sender can not be sure if that is the case. To overcome this, we propose the committed receiver oblivious transfer ($\mathsf{CROT}$) primitive.

In $\mathsf{CROT}$, the receiver forwards a non-interactive zero knowledge (NIZK) proof of knowledge to prove that the bit inputs from the receiver are in fact bits of the signing key. The functionality of the protocol $\mathsf{CROT}$ is presented in the Figure 3.7. The functionality $\mathcal{F}_{CROT}$ interacts with the sender $\mathsf{S}$ and receiver $\mathsf{R}$. The sender has $2\kappa$ messages $M_{\mathrm{i,j}}, \mathrm{i} \in [0, \kappa - 1], \mathrm{j} \in 0, 1$. The receiver has bits $s_{\mathrm{i}}$ which form the secret key $sk$ of the key pair $(sk, pk)$. The adversary $\mathcal{A}$ controls the communication and the delivery of the messages.

The sender $\mathsf{S}$ uses the session id *sid* and tag `inputS` to forward the messages $M_{\mathrm{i,j}}$ to the functionality and to initiate the protocol instance. The functionality stores the sender

input messages using the record $\langle \mathsf{S}, M_{\mathrm{i,j}}, sid \rangle$. The functionality intimates the initiation of the session to the adversary and the receiver using the messages `input, intd` respectively. Upon the intimation, the receiver $\mathsf{R}$ forwards the bits $s_{\mathrm{i}}$ with the `inputR` tag to the functionality. After receiving the message $(\texttt{inputR}, s_{\mathrm{i}}, sid)$, the functionality stores the record $(\mathsf{R}, s_{\mathrm{i}}, sid)$ and intimates the adversary that the input has been received. The adversary sends the messages `deliverS` and `deliverR` to ask the functionality to deliver the outputs to the sender and the receiver. Upon receiving `deliverS`, the functionality checks if a record $\langle \mathsf{R}, s_{\mathrm{i}}, sid \rangle$ is stored and if yes, it sends the message $(\texttt{delivered}, sid)$ to the sender. On receiving the message (`deliverR, sid`), the functionality checks if there is a corresponding record $\langle \mathsf{S}, M_{\mathrm{i,j}}, sid \rangle$. If it exists, it verifies whether the bits $s_{\mathrm{i}}$ forwarded by the receiver correspond to the secret key $(sk)$ of the public key $pk$. On successful verification, it forwards the messages $M_{\mathrm{i},s_{\mathrm{i}}}$ to the receiver.

We depict the construction of the protocol in Figure 3.6 .

**Construction.** The protocol construction for the ideal functionality $\mathcal{F}_{\mathsf{CROT}}$ as given is the Figure 3.7 is presented here. The sender has messages $M_{\mathrm{i,j}}$ for $0 \leq \mathrm{i} \leq \kappa - 1$ and $\mathrm{j} \in \{0, 1\}$. The receiver has a signing key $\mathsf{sk}$ ($s_{\mathrm{i}}$ for $0 \leq \mathrm{i} \leq \kappa - 1$ are the bits of $\mathsf{sk}$). The sender and receiver inputs are modelled using the `inputS` and `inputR` messages of the functionality; the sender initiates the protocol using the `inputS` message. Given a multiplicative group $\mathsf{G}$ and its generator $\mathsf{g}$, the sender initially chooses a random value $a \leftarrow \mathbf{Z}_q$ and forwards $\mathsf{h} = \mathsf{g}^a$ to the receiver. This would be the *Setup* phase. In the next *Commit and Prove* phase, the receiver chooses random $r_{\mathrm{i}} \leftarrow_R Z_q$ and computes $\mathsf{c}_{\mathrm{i}} = \mathsf{g}^{r_{\mathrm{i}}}\mathsf{h}^{s_{\mathrm{i}}}$ for $0 \leq \mathrm{i} \leq \kappa - 1$. The $\mathsf{c}_{\mathrm{i}}$ values are forwarded to the sender as commitments to the bits $s_{\mathrm{i}}$. The receiver also forwards $r = \sum_{\mathrm{i}=0}^{\kappa-1} 2^{\mathrm{i}} r_{\mathrm{i}}$ to the sender. Along with these, for $0 \leq \mathrm{i} \leq \kappa - 1$, the receiver forwards non-interactive zero knowledge (NIZK) proofs of knowledge of exponents $r_{\mathrm{i}}$ and $s_{\mathrm{i}}$ such that $\mathsf{c}_{\mathrm{i}} = \mathsf{g}^{r_{\mathrm{i}} + a s_{\mathrm{i}}}$.

Each of these NIZK proofs is realized using the standard Fiat-Shamir transformation [122] of an interactive sigma protocol for Pedersen commitments in the random oracle model.

| **Sender** | | **Receiver** |
|---|---|---|
| Multiplicative (Public) Group $\mathsf{G}$, generator $\mathsf{g}$ | | |
| $\mathsf{pk} = \mathsf{g}^{\mathsf{sk}}$ | | $\mathsf{sk} \in \{0,1\}^{\kappa}$ |
| For all $\mathrm{i} : 0 \leq \mathrm{i} \leq \kappa - 1$ | | |
| Message blocks: $M_{\mathrm{i},0}$ and $M_{\mathrm{i},1}$ | | Bit decomposition of $\mathsf{sk}$: $s_{\mathrm{i}}$ |
| **Challenge** | | |
| $a \leftarrow_R \mathbb{Z}_q$ | $\xrightarrow{\mathsf{h}=\mathsf{g}^a}$ | |
| **Commit and Prove** | | |
| For all $\mathrm{i} : 0 \leq \mathrm{i} \leq \kappa - 1$ | | |
| | | $r_{\mathrm{i}} \leftarrow_R \mathbb{Z}_q$ |
| | | $r = \sum_{\mathrm{i}=0}^{\kappa-1} 2^{\mathrm{i}} r_{\mathrm{i}}$ |
| | | $\mathsf{c}_{\mathrm{i}} = \mathsf{g}^{r_{\mathrm{i}}}\mathsf{h}^{s_{\mathrm{i}}}$ |
| | $\xleftarrow{\mathsf{g}^r, \mathsf{c}_{\mathrm{i}}, \mathsf{PoK}\{(r_{\mathrm{i}}, s_{\mathrm{i}}) \| \mathsf{g}^{r_{\mathrm{i}}}\mathsf{h}^{s_{\mathrm{i}}}\}}$ | |
| $\mathsf{c} = \prod_{\mathrm{i}=0}^{\kappa-1} \mathsf{c}_{\mathrm{i}}^{(2^{\mathrm{i}})}$ | | |
| Abort if $\mathsf{c} \neq (\mathsf{g}^r \mathsf{pk}^a)$ or | | |
| if verfication of NIZK fails | | |
| **Transfer** | | |
| For all $\mathrm{i} : 0 \leq \mathrm{i} \leq \kappa - 1$ and $\mathrm{j} \in \{0,1\}$ | | |
| $k_{\mathrm{i,j}} = H((\mathsf{c}_{\mathrm{i}} \cdot \mathsf{h}^{-\mathrm{j}})^a)$ | | $k_{\mathrm{i},s_{\mathrm{i}}} = H(\mathsf{h}^{r_{\mathrm{i}}})$ |
| $p_{\mathrm{i}} = H(H(k_{\mathrm{i},0})) \oplus H(H(k_{\mathrm{i},1}))$ | $\xrightarrow{p_{\mathrm{i}}}$ | $p'_{\mathrm{i}} = H(k_{\mathrm{i},s_{\mathrm{i}}}) \oplus p_{\mathrm{i}} s_{\mathrm{i}}$ |
| Verify $p'_{\mathrm{i}} = H(H(k_{\mathrm{i},0}))$ | $\xleftarrow{p'_{\mathrm{i}}}$ | |
| $C_{\mathrm{i,j}} = E_{k_{\mathrm{i,j}}}(M_{\mathrm{i,j}})$ | $\xrightarrow{C_{\mathrm{i,j}}}$ | Decrypt $C_{\mathrm{i},s_{\mathrm{i}}}$ using $k_{\mathrm{i},s_{\mathrm{i}}}$ |

**Figure 3.6.** Committed Receiver Oblivious Transfer (CROT) Protocol

Following the formal symbolic notation introduced by Camenisch and Stadler [123], each proof is depicted as $\mathsf{PoK}\ \{(r_{\mathrm{i}}, s_{\mathrm{i}}) \| \mathsf{g}^{r_{\mathrm{i}}}\mathsf{h}^{s_{\mathrm{i}}}\}$ in Figure 3.6. This phase is used by the receiver to prove that the bits $s_{\mathrm{i}}$ used for the transfer are indeed the bits of the signing key $\mathsf{sk}$.

The sender verifies if $\mathsf{c} = \mathsf{g}^r \mathsf{pk}^a$ for the computed $\mathsf{c} = \prod_{\mathrm{i}=0}^{\kappa-1} c_{\mathrm{i}}^{(2^{\mathrm{i}})}$. He also verifies the NIZK proof. If both the verifications succeed, he proceeds with the protocol, else, aborts. The verification would also fail if $(pk, sk)$ are not a key pair.

**Functionality $\mathcal{F}_{\mathsf{CROT}}$**

Ideal functionality $\mathcal{F}_{\mathsf{CROT}}$ interacts with sender $\mathsf{S}$ and receiver $\mathsf{R}$. The sender has messages $M_{\mathrm{i},0}, M_{\mathrm{i},1}$ and the receiver has bits $s_{\mathrm{i}}, \mathrm{i} \in [0, \kappa - 1]$. $s_{\mathrm{i}}$ correspond to the secret key $sk$ whose public key is $pk$. The adversary $\mathcal{A}$ corrupts either the sender or receiver.

**Sender input.** Upon receiving the message $(\mathtt{inputS}, M_{\mathrm{i,j}}, sid)$ with $M_{\mathrm{i,j}} \in \{0,1\}^*$, $\mathrm{i} \in [0, \kappa - 1], \mathrm{j} \in \{0,1\}$ from sender $\mathsf{S}$, record $\langle \mathsf{S}, M_{\mathrm{i,j}}, sid \rangle$, forward the message $(\mathtt{intd}, sid)$ to the receiver $\mathsf{R}$ and $(\mathtt{input}, sid)$ to $\mathcal{A}$.

**Receiver input.** Upon receiving the message $(\mathtt{inputR}, s_{\mathrm{i}}, sid)$ with $s_{\mathrm{i}} \in \{0,1\}$, $\mathrm{i} \in [0, \kappa - 1]$ from receiver $\mathsf{R}$, record $\langle \mathsf{R}, s_{\mathrm{i}}, sid \rangle$, forward the message $(\mathtt{input}, sid)$ to $\mathcal{A}$.

**Sender output.** Upon receiving the message $(\mathtt{deliverS}, sid)$ from $\mathcal{A}$, check if $\langle \mathsf{R}, s_{\mathrm{i}}, sid \rangle$ is stored, else ignore the message. Send $(\mathtt{delivered}, sid)$ to $\mathsf{S}$.

**Receiver output.** Upon receiving the message $(\mathtt{deliverR}, sid)$ from $\mathcal{A}$, check if $\langle \mathsf{S}, M_{\mathrm{i,j}}, sid \rangle$ is stored, else ignore the message. Check if the secret key $sk$ formed from the bits $s_{\mathrm{i}}$ forwarded by the receiver corresponds to the public key $pk$. Else, ignore the message. Forward $(\mathtt{output}, M_{\mathrm{i},s_{\mathrm{i}}}, sid), \mathrm{i} \in [0, \kappa - 1]$ to $\mathsf{R}$.

**Figure 3.7.** Ideal Functionality of $\mathsf{CROT}$

After successful verification the sender computes the keys $k_{\mathrm{i,j}} = H((\mathsf{c}_{\mathrm{i}} \cdot h^{-\mathrm{j}})^a)$ for each $0 \leq \mathrm{i} \leq \kappa - 1$ and $\mathrm{j} \in \{0,1\}$. The sender verifies if the receiver computed the keys using the verification step similar to Verified Simplest OT [22]. He forwards the challenges $p_{\mathrm{i}} = H(H(k_{\mathrm{i},0})) \oplus H(H(k_{\mathrm{i},1}))$ for each i and receives the responses in the form of $p'_{\mathrm{i}}$ and the sender verifies if $p'_{\mathrm{i}} = H(H(k_{\mathrm{i},0}))$. The keys $k_{\mathrm{i,j}}$ are used to encrypt messages $M_{\mathrm{i,j}}$ respectively to obtain the cipher texts $C_{\mathrm{i,j}}$. The cipher texts $C_{\mathrm{i,j}}$ are forwarded to the receiver who attempts to decrypt the blocks $C_{\mathrm{i},s_{\mathrm{i}}}$ using the keys $k_{\mathrm{i},s_{\mathrm{i}}}$ finishing the *Transfer* phase. The receiver can not compute the keys $k_{\mathrm{i},1-s_{\mathrm{i}}}$ (follows from Lemma 1 of [95]) and so can not decrypt $C_{\mathrm{i},1-s_{\mathrm{i}}}$. One can observe that the protocol does not enforce the receiver to use "bits", if the receiver uses any other values other than bits in $\mathsf{CROT}$, the receiver receives encryptions which can not be decrypted.

The model for $\mathsf{CROT}$ includes static corruption of parties, modelling $H$ as random oracle and group $\mathsf{G}$ being Gap-DH [121] while the encryption used is symmetric, non-committing and robust [95].

**Theorem 3.5.1.** *The CROT protocol UC-realizes the ideal functionality $\mathcal{F}_{\mathsf{CROT}}$ in the $\mathcal{F}_{ZK}^{DL}$-hybrid model under the following assumptions:*

***Corruption Model:*** *static corruption*

***Hybrid Functionalities:*** *H is modeled as a random oracle and authenticated channels between users are assumed.*

***Computational Assumptions:*** $\mathsf{G}$ *is Gap-DH. The symmetric encryption used is non-committing and robust.*

*Proof.* The simulator $\mathcal{S}_{\mathsf{CROT}}$ interposes between a corrupted party and the CROT functionality $\mathcal{F}_{\mathsf{CROT}}$. The verified OT is a "Selective-Failure" Oblivious Transfer, in which the sender can guess the choice bit of the receiver and if the guess is correct, he will be notified it is correct and the receiver is not informed of the same. However, in our CROT protocol, all the messages are transferred simultaneously. For the sender to guess the receiver's choice bits, they need to guess all the bits simultaneously. The probability of the sender guessing all the receiver bits correctly is negligible.

**Malicious Sender.** The simulator $\mathcal{S}_{\mathsf{CROT}}$ interposes between a malicious sender and the CROT functionality $\mathcal{F}_{\mathsf{CROT}}$, it outputs the sender's messages $M_{\mathrm{i},0}, M_{\mathrm{i},1}$.

- Receiver $(\texttt{prove}, a, A)$ from sender on behalf of $\mathcal{F}_{ZK}^{R_{DL}}$. On receiving $(\texttt{accept,A})$ forward it to the sender, else abort.
- Sample random values $s_{\mathrm{i}}, r_{\mathrm{i}}, \mathrm{i} \in [0, \kappa - 2]$ and compute the corresponding $c_{\mathrm{i}} = \mathsf{g}^{r_{\mathrm{i}}}\mathsf{h}^{s_{\mathrm{i}}}$ and compute $s_{\kappa-1}, r_{\kappa-1}$ such that $\prod_{\mathrm{i}=0}^{\kappa-1} = \mathsf{g}^{r}\mathsf{pk}^{a}$ where $r = \sum_{\mathrm{i}=0}^{\kappa-1} 2^{\mathrm{i}} r_{\mathrm{i}}$. Compute ZK-PoKs $\boldsymbol{\pi}_{\mathrm{i}}$ proving the knowledge of $r_{\mathrm{i}}, s_{\mathrm{i}}$ for each $\mathsf{g}^{r_{\mathrm{i}}}\mathsf{h}^{s_{\mathrm{i}}}$. Forward $\mathsf{g}^{r}, c_{\mathrm{i}}, \boldsymbol{\pi}_{\mathrm{i}}$ to the sender.
- Invoke $\mathcal{F}_{ZK}^{DL}$ to prove that the sampled bits correspond to the public key $pk$
- Compute the pads $k_{\mathrm{i,j}} = H(c_{\mathrm{i}} \cdot h^{-\mathrm{j}})^{a}$. Compute the expected challenges as $p_{\mathrm{i}}^{\exp} = H(H(k_{\mathrm{i},0})) \oplus H(H(k_{\mathrm{i},1}))$
- Upon receiving the sender's challenges $p_{\mathrm{i}}$, If for any i, $p_{\mathrm{i}} = p_{\mathrm{i}}^{\exp}$, then set the $p'_{\mathrm{i}} = H(H(k_{\mathrm{i},0}))$ and add $(\texttt{guess}, s'_{\mathrm{i}})$ to the set $\mathcal{G}$; Otherwise, let $\mathcal{Q}$ be the set of all queries made by the sender to the random oracle. If there exists queries $Q_{\mathrm{j}}$ such that such that

$H(Q_{\mathrm{j}}) = p_{\mathrm{i}} \oplus H(H(k_{\mathrm{i},1}))$ then set $s'_{\mathrm{i}} = 1$. Otherwise set $s'_{\mathrm{i}} = 0$. Add $\mathsf{guess}, s'_{\mathrm{i}}$ to the set $\mathcal{G}$. Send the set $\mathcal{G}$ to $\mathcal{F}_{\mathsf{CROT}}$. If $\mathsf{cheat\text{-}undetected}$ is received, send $k'_{\mathrm{i}} = H(H(k^{s'_{\mathrm{i}}}))$ to the sender. Otherwise send $k'_{\mathrm{i}} = H(H(k^{s'_{\mathrm{i}}}))$ and halt.

- Upon receiving the cipher texts $C_{\mathrm{i,j}}$ decrypt them using $k_{\mathrm{i,j}}$ and send them to the functionality $\mathcal{F}_{\mathsf{CROT}}$.

**Malicious Receiver.** The simulator interposes between the ideal functionality $\mathcal{F}_{\mathsf{CROT}}$ and the malicious receiver. It outputs the choice bits $s_{\mathrm{i}}$ of the receiver and the corresponding message chosen $M_{\mathrm{i},s_{\mathrm{i}}}$. It makes use of the random oracle $H$ and the functionalities $\mathcal{F}_{ZK}^{R_{DL}}$

- Sample $a \in \mathbb{Z}_p$ and compute $\mathsf{g}^a$ to the receiver on behalf of the functionality $\mathcal{F}_{ZK}^{DL}$.
- Receive $g^r, c_{\mathrm{i}}, \boldsymbol{\pi}_{\mathrm{i}}$ from the receiver just like an honest sender. Verify the proofs and abort if any of the forwarded proofs fail.
- Compute the keys $k_{\mathrm{i,j}}$ like an honest sender.
- Observe the random oracle queries of the receiver. If the receiver ever queries $k_{\mathrm{i},0}$ set $s_{\mathrm{i}} = 0$. If they every query $k_{\mathrm{i},1}$ set $s_{\mathrm{i}} = 1$. Once $b_{\mathrm{i}}$s are set, send $s_{\mathrm{i}}$ to the the functionality $\mathcal{F}_{\mathsf{CROT}}$ and receiver $\mathsf{no\text{-}cheat}$.
- Run the verification as the honest sender would.
- Upon receiving the messages $M_{\mathrm{i},s_{\mathrm{i}}}$ from the functionality, set the corresponding ciphertexts as $C_{\mathrm{i},s_{\mathrm{i}}} = E_{k_{\mathrm{i},s_{\mathrm{i}}}}(M_{\mathrm{i},s_{\mathrm{i}}})$ and set the other ciphertexts to random values.

In the malicious sender case, the first message received by the consists of the $\mathsf{g}^r, c_{\mathrm{i}}, \mathsf{PoK}$. Since $r$ is picked randomly, the view of the sender is identical in both the worlds. The simulator $\mathcal{S}_{\mathsf{CROT}}$ receives the value $a$ on behalf of the functionality $\mathcal{F}_{ZK}^{DL}$ and so can compute the values $r_{\mathrm{i}}, c_{\mathrm{i}}$ such that the zero-knowledge proof and verification check hold. It can also compute the the keys $k_{\mathrm{i,j}}$ and hence verify if the challenges received $p_{\mathrm{i}}$ are correct.

During the verification phase of the transfer, the sender is required to compute values $H(H(k_{\mathrm{i},0})), H(H(k_{\mathrm{i},1}))$, only one set of the hashes are known to the receiver which correspond

to $H(H(k_{\mathrm{i},s_\mathrm{i}}))$. To induce a selective failure, the sender can try to guess the receiver bits and set random values for the opposite ones while calculating the challenges $p_\mathrm{i}$, To guess all the bits correctly and simultaneously, the sender succeeds only with negligible probability $\frac{1}{2^\kappa}$. All the oracle queries made by the sender can be used to compute the sender's guesses in the protocol which can be forwarded to the functionality which aborts if the guesses are incorrect. After this point, the simulator behaves like an honest real world receiver and forwards all the messages accordingly and aborts under same conditions. There the view of the malicious sender under the real world execution of the protocol is indistinguishable from its view while interacting with the simulator $\mathcal{S}_{\mathsf{CROT}}$, he can distinguish the view with no better probability than $\frac{1}{2^\kappa}$.

In the malicious receiver case, $h$ is chosen by the simulator and $c_\mathrm{i}$ is chosen by the receiver. These values fix the computed keys $k_{\mathrm{i,j}}$ to be computed. The receiver can not guess the $k_{\mathrm{i},s_\mathrm{i}}$ values except with probability of $\frac{1}{2^\kappa}$ for each. When the receiver queries the random oracle, the simulator records the queries and finds the corresponding choice bit $s_\mathrm{i}$. If the receiver can query the random oracle at $k_{\mathrm{i},s_\mathrm{i}}$ and $k_{\mathrm{i},1-s_\mathrm{i}}$, then the simulator can not compute the choice bit. However the receiver can not make both those queries, as any such receiver breaks the CDH assumption. The rest of the simulator steps follow a honest sender and the view generated is identically distributed to the real-world paradigm. Thus the view of the malicious receiver is identical in the real world and the ideal world paradigm if the CDH problem is hard in the group selected.

□

### 3.6 The Pepal protocol

Here, we detail the steps of the Pepal protocol which uses DOT with CROT. The watermarking and the DOT protocol are the *off-chain* cryptographic components while the smart-contract and the deposit are the *on-chain* parts.

1. **NetworkSetup**: The sender and receiver setup their Bitcoin identities by generating secret key-public key pairs; the sender has the document $M$.

2. **DepositSetup**$(sk, t, Value)$: A time-locked bitcoin deposit is created by the receiver with the signing key $sk$ for a time $t$ and for a amount of $Value$. The deposit is a 2-of-2 multisig deposit requiring the secret keys of both the sender and the receiver to transfer the funds.

3. **WaterMark**$(M)$: The document $M$ is broken into $\kappa$ blocks $M_{\mathrm{i}}, 0 \leq \mathrm{i} \leq \kappa - 1$ for a $\kappa$-bit long $sk$ and each block $M_{\mathrm{i}}$ is watermarked to generate two versions $M_{\mathrm{i},0}, M_{\mathrm{i},1}$. Any watermarking scheme which satisfies the previously mentioned properties (refer section 2.3) can be used.

4. **DOT with CROT**$(M_{\mathrm{i},0}, M_{\mathrm{i},1}, sk)$: The Doubly Oblivious Transfer protocol, used to transfer the document, takes the watermarked blocks as input. In Pepal, the DOT protocol instead of using $\mathsf{OT}_1^2$, uses CROT. The protocol is same as the general case of DOT (as shown in Figure 3.5) but uses CROT instead of $\mathsf{OT}_1^2$. The sender watermarks the document blocks to obtain $M_{\mathrm{i,j}}$, generates keys from sampled group elements and forwards the permuted symmetric encrypted versions of the blocks to the receiver. He then encrypts the group elements using El-Gamal encryption to the key $pk = pk_S * pk_R$ where $pk_S, pk_R$ are the public keys of sender and receiver. The sender inputs encrypted elements in a permuted order to the CROT protocol. The receiver after proving in zero knowledge that the input to the protocol is her signing key $sk$, receives a set of encrypted elements which she re-randomizes and sends back. The sender, decrypts his layer of encryption, inverts the applied permutation to obtain the elements in their original order and forwards them to the receiver who will be able to decrypt them. The appropriately decrypted symmetrically encrypted blocks are then joined together to form the receiver's version of the document $M_{sk}$.

5. **Penalize**$(M_{sk}, sk_S)$: Upon revelation of the document, the receiver's secret key $sk$ is extracted from the document $M_{sk}$ and is used with the sender's secret key $sk_S$ to transfer the deposited funds to the sender to penalize the receiver.

**Utilizing Bitcoin.** Before the Pepal protocol begins, after the two parties agree on the Pepal process, the sender shares his/her public key $pk_S$ with the receiver to create a deposit. The sender will assert that the receiver creates a transaction $TX$ that is valid for a mutually agreed upon time $t$, and can be redeemed by the sender instantly with the signing keys of

the sender ($sk_S$) and the receiver($sk_R$). Here, the deposit should hold the funds equal to an agreed upon value $Value$. *VerifyDeposit(TX)* at the sender verifies the above mentioned criterion. This algorithm receives the hash of the transaction as an input and verifies that the transaction meets the above mentioned criteria, i.e. it is a valid deposit that directs $Value$ to the sender if the sender has both the signing/private keys. Earlier versions of Bitcoin allowed senders to broadcast time locked transactions and these transactions would be in the unverified transactions pool until the time lock expired or an unlocking *scriptSig* was provided by the spender of $TX$. However, current (as of February 2019) Bitcoin transaction does not permit nodes to propagate transactions that have an active time lock. Therefore, the receiver sends $TX$ over any secure communication channel so that the sender can verify and sign the transaction. Once the document becomes public, we are assured from the watermarking scheme that the leaked copy of the document will have the receiver's signing key. Using the extraction algorithm *Extract($M$, $M_{sk}$)* the sender can reconstruct the signing key $sk$. Once the sender has $sk$, he can sign the transaction $TX$ with the *Sign($TX$, $sk$)* and broadcast the signed transaction directing the funds in $TX$ to his Bitcoin address.

**Ideal functionality.** Figure 3.8 presents the ideal functionality $\mathcal{F}_{\mathsf{Pepal}}$ for Pepal, while Theorem 3.6.1 proves its security.

The functionality $\mathcal{F}_{\mathsf{Pepal}}$ interacts with the sender S and receiver R. The sender has $2\kappa$ watermarked messages $M_{\mathrm{i,j}}, \mathrm{i} \in [0, \kappa - 1], \mathrm{j} \in 0, 1$. The receiver has bits $s_{\mathrm{i}}$ which form the secret key $sk$ of the key pair $(sk, pk)$. While forwarding the input the sender also forwards a permutation $\pi(\cdot)$ of indices $[0, \kappa - 1]$. Before delivering the messages $M_{\mathrm{i},l}, l = s_{\pi(\mathrm{i})}$ to the receiver, the functionality checks if the input bits $s_{\mathrm{i}}$ form the secret key $sk$ corresponding to the public key $pk$.

Here we show that the functionality achieves the desirable properties discussed in Section 3.1. The properties of sender and receiver privacy are trivially satisfied by the functionality as it does not reveal any information except transferring the corresponding watermarked blocks to the receiver. If the receiver discloses the document, the sender can extract the embedded watermark bits and hence the signing key of the receiver, thus satisfying the revealing property. If the sender tries to falsely accuse the receiver by revealing the document

in any form, the receiver does not lose the deposit as the sender does not have the receiver's key without disclosure, this achieves the sender integrity property. Though the penalization is shown as a step of Pepal, as it takes place outside of the transfer mechanism after the data breach in a non-interactive way, it is not included in the ideal functionality of the Pepal protocol.

**Functionality $\mathcal{F}_{\mathsf{Pepal}}$**

Ideal functionality $\mathcal{F}_{\mathsf{Pepal}}$ interacts with sender $\mathsf{S}$ and receiver $\mathsf{R}$. The sender has watermarked messages $M_{\mathrm{i},0}, M_{\mathrm{i},1}$ and the receiver has bits $s_{\mathrm{i}}, \mathrm{i} \in [0, \kappa - 1]$. $s_{\mathrm{i}}$ correspond to the secret key $sk$ where public key is $pk$. The adversary $\mathcal{A}$ corrupts either the sender or receiver.

- Upon receiving the message $(\texttt{inputS}, M_{\mathrm{i,j}}, \boldsymbol{\pi}(\cdot), sid)$ with watermarked $M_{\mathrm{i,j}} \in \{0,1\}^*$, $\mathrm{i} \in [0, \kappa-1], \mathrm{j} \in \{0,1\}$, a random permutation $\boldsymbol{\pi}(\cdot)$ from sender $\mathsf{S}$, record $\langle \mathsf{S}, M_{\mathrm{i,j}}, \boldsymbol{\pi}(\cdot), sid \rangle$, forward the message $(\texttt{intd}, sid)$ to the receiver $\mathsf{R}$ and $(\texttt{input}, sid)$ to $\mathcal{A}$.
- Upon receiving the message $(\texttt{inputR}, s_{\mathrm{i}}, sid)$ with $s_{\mathrm{i}} \in \{0,1\}$, $\mathrm{i} \in [0, \kappa - 1]$ from receiver $\mathsf{R}$, record $\langle \mathsf{R}, s_{\mathrm{i}}, sid \rangle$, forward the message $(\texttt{input}, sid)$ to $\mathcal{A}$.
- Upon receiving the message $(\texttt{deliverS}, sid)$ from $\mathcal{A}$, check if $\langle \mathsf{R}, s_{\mathrm{i}}, sid \rangle$ is stored, else ignore the message. Send $(\texttt{delivered}, sid)$ to $\mathsf{S}$.
- Upon receiving the message $(\texttt{deliverR}, sid)$ from $\mathcal{A}$, check if $\langle \mathsf{S}, M_{\mathrm{i,j}}, sid \rangle$ is stored, else ignore the message. Check if the secret key $sk$ formed from the bits $s_{\mathrm{i}}$ forwarded by the receiver corresponds to the public key $pk$. Else, ignore the message. Forward $(\texttt{output}, M_{\mathrm{i},l}, sid), l = s_{\pi(\mathrm{i})}, \mathrm{i} \in [0, \kappa - 1]$ to $\mathsf{R}$.

**Figure 3.8.** Ideal Functionality of Pepal

**Theorem 3.6.1.** *The Pepal protocol securely implements the ideal functionality $\mathcal{F}_{\mathsf{Pepal}}$ under the following assumptions:*

***Corruption Model:*** *static corruption*

***Hybrid Functionalities:*** *H is modeled as a random oracle and authenticated channels between users are assumed.*

***Computational Assumptions:*** *CDH and DDH are assumed to be hard in $\mathbb{G}$, $\mathsf{G}$ is Gap-DH. The symmetric encryption used is non-committing and robust.*

*Proof* Pepal protocol uses DOT which internally uses CROT instead of multiple instances of the standard $\mathsf{OT}_1^2$ for the transfer of messages/document blocks from the sender to the

receiver. The simulator for the Pepal protocol simply invokes the corresponding simulator $\mathcal{S}_{\mathsf{DOT}}$ which invokes the simulator $\mathcal{S}_{\mathsf{CROT}}$ instead of instances of $\mathcal{S}_{OT}$. The UC-security of the CROT protocol is already established through Theorem 3.5.1.

**Malicious sender.**

The simulator $\mathcal{S}_{\mathsf{CROT}}$ interposes between a malicious sender and the Pepal functionality $\mathcal{F}_{\mathsf{Pepal}}$.

- Receive $(\mathsf{prove}, sk_{\mathsf{S}}, pk_{\mathsf{S}})$ on behalf of $\mathcal{F}_{ZK}^{DL}$. On accepting, forward accept to the sender, else abort.
- Sample a random secret key $sk_{R'}$ and parse the bits of the secret key into $s_{\mathrm{i}}, \mathrm{i} \in [0, \kappa - 1]$ and participate in the DOT protocol.
- Invoke the malicious sender phase of the simulator $\mathcal{S}_{\mathsf{DOT}}$ for the same.
- The simulator $\mathcal{S}_{\mathsf{DOT}}$ receives the ElGamal encryptions from the sender just as a receiver would
- For the message transfer, $\mathcal{S}_{\mathsf{DOT}}$ inturn invokes a single instance of the malicious sender phase of the CROT simulator $\mathcal{S}_{\mathsf{CROT}}$ (instead of multiple instances of $\mathcal{S}_{OT}$) during the transfer phase.
- The simulator $\mathcal{S}_{\mathsf{CROT}}$ after interacting with the malicious sender, outputs the sender messages $u_{\mathrm{i,j}}$. Since the simulator acts as the receiver it has access to $sk_{\mathsf{R}}$. It also has access to $sk_{\mathsf{S}}$ through the $\mathcal{F}_{ZK}^{DL}$ functionality. Hence it can decrypt the messages $u_{\mathrm{i,j}}$.
- After this the simulator behaves like a honest receiver and participates in all the further protocol steps.
- The keys $u_{\mathrm{i,j}}$ are used to decrypt the messages $M_{\mathrm{i,j}}$. Forward the messages $M_{\mathrm{i,j}}$ to the functionality $\mathcal{F}_{\mathsf{Pepal}}$ as $\langle \mathsf{inputS}, M_{\mathrm{i,j}}, \pi, sid \rangle$ for the session id $sid$.

**Malicious receiver.** The simulator $\mathcal{S}_{\mathsf{CROT}}$ interposes between a malicious receiver and the Pepal functionality $\mathcal{F}_{\mathsf{Pepal}}$.

- Receive $(\mathsf{prove}, sk_{\mathsf{R}}, pk_{\mathsf{R}})$ on behalf of $\mathcal{F}_{ZK}^{DL}$. On accepting, forward $\mathsf{accept}$ to the receiver, else abort.

- Invoke the malicious receiver phase of the simulator $\mathcal{S}_{\mathsf{DOT}}$ which forwards the encryptions of the keys.

- As a part of steps of $\mathcal{S}_{\mathsf{DOT}}$, invoke the malicious receiver phase of $\mathcal{S}_{\mathsf{CROT}}$ instead of multiple instances of the simulator $\mathcal{S}_{OT}$.

- $\mathcal{S}_{\mathsf{CROT}}$ outputs the choice bits $s_{\mathrm{i}}$ of the receiver.

- Forward the choice bits to the functionality $\mathcal{F}_{\mathsf{DOT}}$ to obtain the messages $M_{\mathrm{i},s_{\mathrm{i}}}$.

- Use the receiver bits $s_{\mathrm{i}}$ through $\mathsf{DOT}$ simulator to set the encryptions which can be opened by the receiver to the values forwarded by the functionality $\mathcal{F}_{\mathsf{DOT}}$.

The simulator $\mathcal{S}_{\mathsf{Pepal}}$ is the simulator $\mathcal{S}_{\mathsf{DOT}}$ which invokes the simulator $\mathcal{S}_{\mathsf{CROT}}$ instead of multiple instances of $\mathcal{S}_{OT}$ for the transfer protocol. The UC-security follows from the UC-security of the $\mathsf{DOT}$ and the $\mathsf{CROT}$ protocols. $\mathcal{S}_{DOT}$ which internally invokes $\mathcal{S}_{\mathsf{CROT}}$ (instead of $\mathcal{S}_{OT}$), produces an indistinguishable view for the adversary in the real world-ideal world paradigm.

### 3.6.1 Illustration

We illustrate the utility of $\mathsf{Pepal}$ with $\mathsf{DOT}$ using $\mathsf{CROT}$ with an example. The sender can break the document down into more than $\kappa$ blocks, say $2\kappa$, to perform $\mathsf{CROT}$ twice, there by embedding the receiver's key two times. The finer he breaks the document, the more number of times he will be able to embed the receiver's key and so can extract more number of bits upon partial disclosure. For a receiver with 256 bit key, the sender for embedding the key twice divides the document into 512 blocks and creates two watermarked versions for each of the 512 blocks and wishes to transfer 512 messages.

The receiver wishes to selectively reveal parts of the document to the public while not revealing too much of her key bits to the sender. It is understood that the receiver reveals at least enough number of blocks (not too few) to carry useful/sufficient information. Let

**Table 3.1.** Time (mean ± standard deviation) taken (in seconds) for steps of the protocol when signing key is embedded for $\ell = 1, 4$ and 16

| | Watermarking | Full protocol |
|---|---|---|
| $\ell = 1$ | 0.357 ± 0.009 | 1.737 ± 0.226 |
| $\ell = 4$ | 1.346 ± 0.213 | 16.067 ± 0.638 |
| $\ell = 16$ | 1.643 ± 0.283 | 83.101 ± 1.623 |

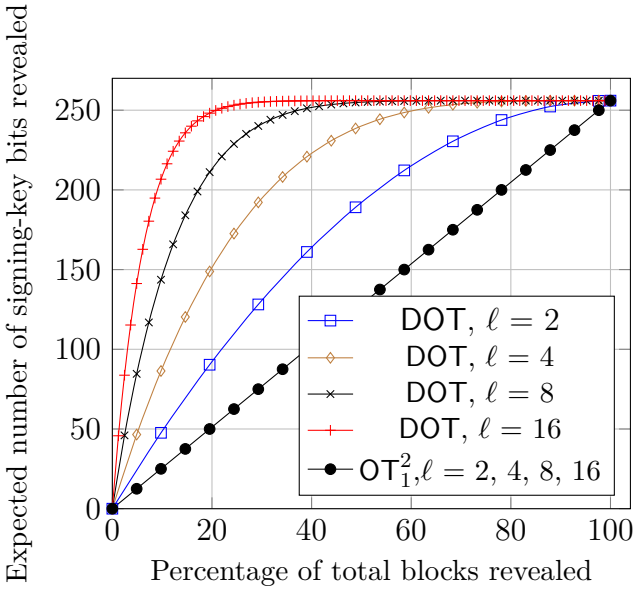
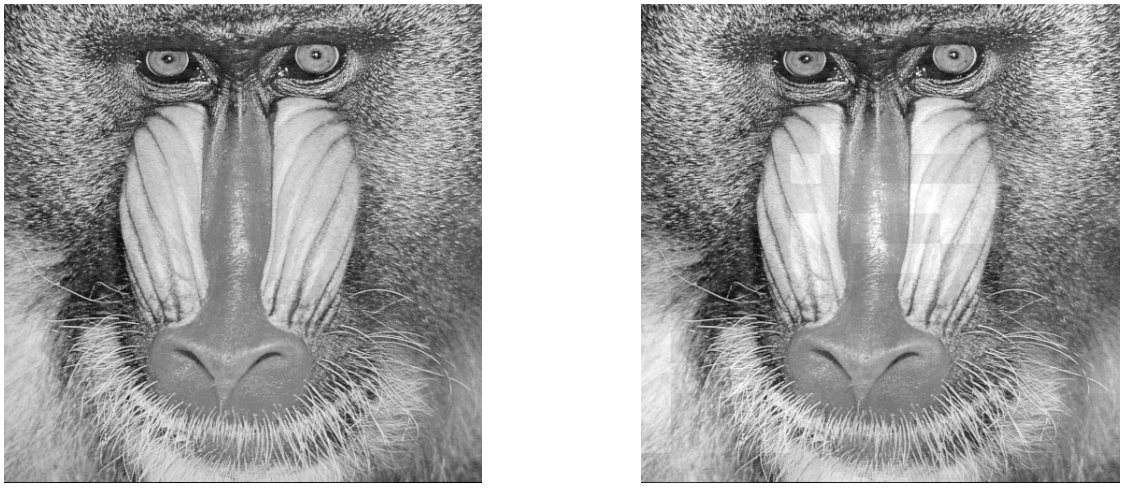



**Figure 3.9.** Number of bits revealed to sender upon dishonest disclosure by receiver when Pepal is employed with $\mathsf{OT}_1^2$ and DOT with 256-bit signing key

us assume she wishes to reveal 100 document blocks. We wish to compare how many bits she will actually reveal to the sender when she reveals 100 document blocks when Pepal with DOT is used, to a scenario where just $\mathsf{OT}_1^2$ is used to transfer the messages instead of DOT.

If the sender uses just $\mathsf{OT}_1^2$ for the message transfer, he inputs one pair of messages for each $\mathsf{OT}_1^2$ and performs 512 such $\mathsf{OT}_1^2$ instances to transfer the 512 messages. In this case, the receiver knows which document block has been obtained using a particular key bit and so knows which two blocks have a certain key bit embedded in them. As she knows which two blocks have the same bit embedded in them, she will reveal 50 such pairs (with the same key bit) to the public so that the sender can learn only 50 of her signing key bits .



(a) Original image before watermarking at the sender

(b) Watermarked image reconstructed by the receiver

**Figure 3.10.** Original and reconstructed images

However, if the sender uses DOT with CROT to transfer the document and the receiver decides to reveal 100 document blocks, as she does not know which key bit is embedded in a certain document block, she randomly picks 100 document blocks and reveals to the public. The expected number of key bits revealed to the sender in such a scenario would be 90.3 for 100 blocks as opposed to 50 bits with just $\mathsf{OT}_1^2$. Following [124], [125], the expected number of bits revealed to the sender when $m$ blocks of the document are released with $\kappa$ -bit key being watermarked over $\ell$ times in the document is $\kappa\left[1-\left[\binom{(\kappa-1)*\ell}{m}/\binom{\kappa*\ell}{m}\right]\right]$.

Figure 3.9 indicates the number of bits revealed to the sender against the percentage of blocks revealed to the public when the signing-key is watermarked $\ell$ times with $\ell \in \{2, 4, 8, 16\}$. When the key is embedded 8 times, a leakage of 20% of the document/file can leak up to 211 bits of the key whereas, when it is embedded 16 times, even a 15% leak reveals as many as 235 bits. This scenario is particularly useful with larger files like video files, where the key can be embedded many number of times such that even a minor clip

of the video can reveal close to the whole of the signing key. The plot in the Figure 3.9 compares the number of signing-key bits revealed to the sender when Pepal uses DOT and $\mathsf{OT}_1^2$. It clearly indicates that higher the number of times the key is embedded, higher are the number of bits revealed to the sender upon leakage. However, one has to note that the maximum number of times a key can be embedded by dividing the document depends on the document and its entropy.

**Computation and Communication Overhead.** For the transfer protocol, the number of exponentiations at the sender and receiver is linear in $\ell$. When DOT uses CROT, the number of exponentiations performed by the sender would be $11\ell\kappa + \ell$ and by the receiver would be $7\ell\kappa$. The communication in the DOT protocol involves forwarding two versions of AES encrypted blocks, messages of CROT and forwarding of $\kappa$ ElGamal encrypted points by the receiver and the sender. In CROT, the sender forwards $2\kappa$ ElGamal encrypted elements while the receiver forwards $3\kappa$ elements including the proof of knowledge messages.

### 3.7 Implementation and Analysis

We have implemented the Pepal protocol as a single-threaded program and analyzed its performance on a MacOS machine with 3.1 GHz Intel Core i7 and 16 GB RAM. Our implementation involves the DOT protocol with robust watermarked images and a claim-or-refund contract as a Bitcoin script. An execution run involves the transfer of an image to the receiver, and we examine the execution times for the different involved modules. The receiver's key is 256-bit long and the sender breaks the document into blocks before proceeding with the protocol.

**Watermarking.** The sender, after creating the document blocks, watermarks each block with 0 and 1 to generate two versions. We employ the watermarking system by Meerwald [126] which implements the Cox algorithm [127] of robust watermarking for the image blocks. The Cox algorithm is well-studied and benchmarked against several attacks [128]. In our scheme, we watermark the image document by embedding the key multiple times, Table 3.1 indicates the watermarking time taken where the 256-bit is embedded for $\ell = 1, 4$ and 16 indicating embedding once, 4 and 16 times. For $\ell = 1, 4$ the document in divided into 256

and 1024 blocks respectively which are transferred using the DOT protocol to the receiver who reconstructs the image from the received blocks. For demonstrative purposes, the original image before watermarking and the image reconstructed at the receiver for $\ell = 1$ are available in Figure 3.10. While we use the Cox algorithm which is not proven to be robust, we reiterate that depending on the data type and application, any robust watermarking scheme can be used in our protocol for that specific application. Works such as [129], [130], [131] present different audio watermarking schemes while works like [132], [133] deal with robust video watermarking. For software watermarking, schemes suggested in [134], [135] can be considered.

**Cryptographic Module - DOT.** For the cryptographic part, we use the RELIC library [136]. The receiver's key is 256-bit long. The sender breaks the document into blocks, encrypts each of the watermarked document and forwards the blocks to the receiver in the first step of DOT protocol. The encryption used to for this step is AES in the counter mode. The sender generates group elements while participating in the DOT protocol to transfer the blocks which are ElGamal encrypted, which are later re-randomized by the receiver. The receiver decrypts the AES encrypted document blocks with the keys obtained through the ElGamal encryption and oblivious transfer.

Table 3.1 provides the computation timing details for the complete protocol i.e., the time including breaking the document into blocks to the point where the receiver reconstructs the document from received watermarked blocks. It presents the statistics of execution times taken over 100 runs of the experiment. Notice that the timing values reported are when the process is running in a single-thread. With multi-threading and pre-processing ElGamal encryption exponentiation, we expect significant improvement in performance and reduction in timing. To simulate the dishonest breach and eventual procurement of the leaked document by the receiver, the reconstructed image is sent to the sender of the document. The sender runs the key-extraction algorithm on the obtained image and extracts the receiver's key to perform the penalization.

### 3.8 Discussion

**Multiple Receivers.** In a scenario involving multiple receivers of the same document, the sender can embed the signing key of a each receiver multiple times into each receiver's version of the document. He can do so by dividing the document into higher number of parts compared to the receiver's key length. This ensures that, in case of collusion and each receiver contributing a small portion of his document while colluding, the sender can still extract considerable amounts of signing keys from the revealed document.

**Contracts.** In Section 2.4.1, we developed a penalization smart contract for the Bitcoin scripting system, which intentionally has a limited set of instructions. Systems like Ethereum [114] expand this set of instructions into a fully-featured programming language allowing it to perform much elaborate tasks where it is easily possible to write our claim-or-refund contract. However, despite the much better expressivity, it does not seem to be possible to create an elaborate contact that can *efficiently* substitute the required DOT protocol and robust watermarking scheme.We implemented the penalizing claim-or-refund smart contract as a Bitcoin smart contract as well as a Hyperledger chaincode, as they allow the systems to be executed in a permissionless as well as permissioned blockchain setting. In the future, it would be interesting to create similar solutions using Solidity over the Ethereum network that can at least partially reduce the required cryptographic tools.

**Fairness.** The receiver deposits the bitcoins before the commencement of the protocol and so, if the document transfer does not go through, his funds will be locked till the end of the deposit time period. This is not 'fair' for the receiver. However, in a more realistic setting, in such a scenario the parties would just re-run the protocol and transfer the document.

**Miner.** The receiver can indeed be a miner in a Bitcoin system. He can try to pre-mine transactions to escape penalty incase of disclosure. This scenario can be prevented by the approach taken in [28, Sec. 6]. In case the sender has the knowledge only of the breach without having access to the revealed document, he can choose to make the watermarking algorithm's private-key public to make the receiver lose her deposit.

**Data Custody.** In case of storing data at a custodian, the user should be retrieving or downloading the data after the end of time period, this is because if the sender retrieves the data, he can get a copy of the receiver's data with receiver's key embedded in it, he may reveal it to the public and try to blame the receiver for the leak. In such a scenario, the parties can agree to retrieve the deposit and nullify the contract and when the sender decides to store the data again, can perform the protocol. Another way is to have a mechanism in which along with the cooperation of the sender, the receiver can forward a copy of the data with the watermark stripped, such an approach can be looked at in the future.

# 4. CDE: COLLUSION-DETERRENT THRESHOLD INFORMATION ESCROW

## 4.1 System Model and Problem Definition

The system consists of $n$ agents who offer a threshold information escrow (TIE) service, and a user engaging the service in a multi-party computation (MPC) setting. The agents are associated with fixed identities (typically connected with real-world identities), the user verifies the identities of the agents before engaging the service. All the communication is over secure and authenticated channels.

We consider the *mixed-behavior* model [35] where the agents are either *rational or malicious*. Rational agents aim to maximize their utility at any given point in time, the malicious ones can deviate from the protocol arbitrarily. Any number of agents among the $n$ agents can collude to increase their utility.

A malicious adversary can make the corrupted parties deviate arbitrarily from the protocol. We consider a $t-$bounded adversary under a static corruption setting:[1] Up to $t$ of $n$ parties can be corrupted before the start of the protocol execution and the corrupted parties remain corrupted throughout the execution.

**Problem Definition.** The user has a private message that she wishes to encrypt to a certain condition and $t$ agents offer the information escrow service. Each of the agents makes a cryptocurrency deposit to the public key $pk$ of the protocol instance, with an embedded condition in the smart contract such that the funds can be transferred when the user-specified condition is met or with the associated secret key $sk$. The user requires that her secret information would not be revealed before the condition is met. The agents can collaborate to selectively open the user message at *any* point of time; however, if the agents open the message, the system should ensure that all the agents' deposits are available to a *subset* of agents. This subset is chosen by the user at the time of using the service. In this setting, we wish to achieve the following security goals:

[1] ↑The employed protocols secure against the static adversary can be made secure against adaptive adversary using standard techniques [137], [138].

*Correctness:* Any secret message can be retrieved if more than a threshold number of agents collaborate (even before the user-specified condition is met).

*Privacy:* No secret information can be retrieved from the system unless more than a threshold number of agents collaborate.

*Revealing:* If the data of a user is decrypted, all other secret information of the protocol will be available to the user-chosen subset of agents.

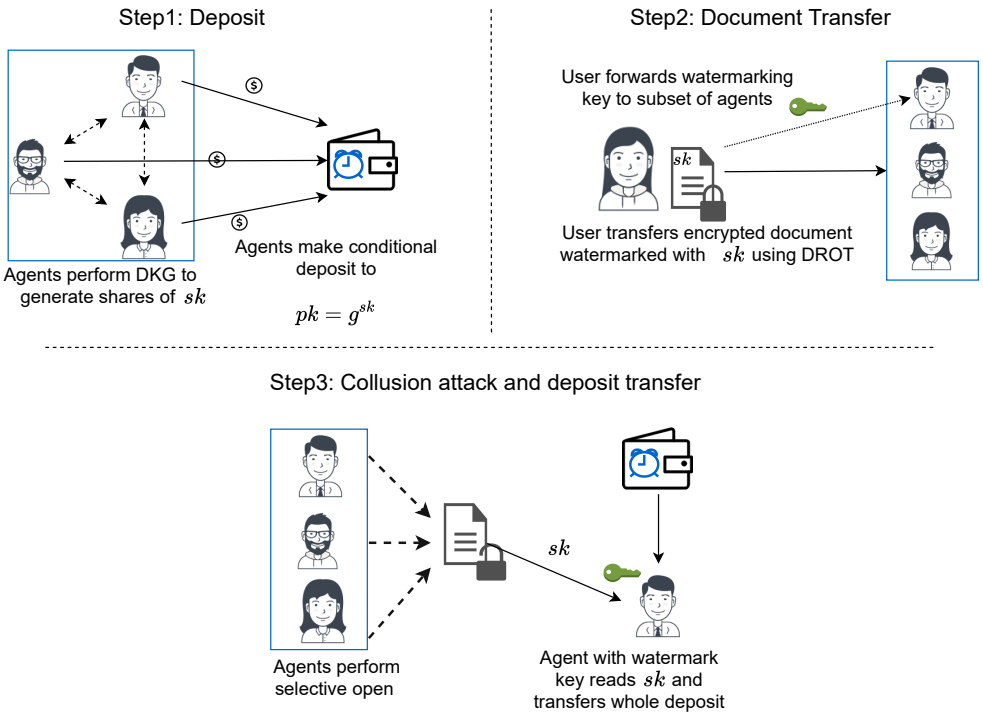



**Figure 4.1.** Steps involved in the collusion deterrent escrow mechanism. Distributed key generation (DKG) is used the the agents to generate shares of secret key $sk$ corresponding to public key $pk$. Distributed receiver obliviovs transfer (DROT) is used the user to transfer a copy of the document to the agents such that it contains the secret key $sk$ as watermark.

### 4.1.1 Key Idea

The escrow agents offer the information escrow service. A public-secret key pair $(pk_\tau, sk_\tau)$ is associated with a condition $\tau$, corresponding to which the user's document is encrypted.

Each secret key $sk$[2] is $(n, t+1)$ secret-shared among the agents using Shamir secret sharing (SSS) [80], where at-least $t+2$ agents are needed to reconstruct the secret. The key idea involves MPC among the agents and the user such that the user transfers a copy of the document *watermarked with the secret key* $sk$ to the agents. Unless at least $t+2$ agents collaborate, none of the agents or the user can determine the transferred, watermarked document. When they collaborate to decrypt the watermarked document, it can reveal the watermark (i.e., the secret key $sk$). However, the user introduces an interesting *information asymmetry* by choosing only a subset of agents and providing them with the watermarking detection key. Only the agents in the user-chosen subset can detect the watermark or $sk$.

Before interacting with the user, all the agents make a claim-or-refund cryptocurrency deposit to an address associated with $pk$ with the user-defined condition $\tau$; here, the funds can be claimed before the condition is met using $sk$ or the respective agents get the refund after the condition is met. If the agents collude to reveal the watermarked document, any agent with access to the watermarking detection key can compute the watermark $sk$ and claim all the deposits of all agents including his own. In case the deposits are claimed (visible publicly on the blockchain) before the condition is met, *all the agents except the transferring agent are banned* from offering the future IE service. As the agents do not have information on which agents have access to the correct detection key, no agent attempts to collude for the fear of losing the deposit and getting banned from the system. In fact, we prove that the best response strategy of the agents is to *not* collude (refer Section 4.4 for a detailed analysis). Figure 4.1 depicts the three steps of making a conditional deposit, document transfer from the user, and agents attempting to collude with one of the agents with watermarking key transferring the deposits of all agents to himself.

### 4.1.2 Protocol Steps

At the beginning of the protocol, the agents run a distributed key generation (DKG) scheme [30] to generate a public key $pk$ and a $(n, t+1)$ SSS of the corresponding secret key

[2] ↑for ease of exposition, we drop the index $\tau$ further.

$sk$, with each agent $\mathsf{A}_{\text{IND}}$ for IND $\in [1, t]$ receiving the IND$^{th}$ share $[\![sk]\!]_{\text{j}}$. Here, an $(n, t+1)$ threshold secret sharing requires at least $t+2$ agents to reconstruct the secret.

The agents run a secure bit decomposition protocol [32]–[34] on the shared key $sk$ to obtain the SSS shares of the secret key $sk$ bits; i.e., each agent $\mathsf{A}_{\text{IND}}$ now have a SSS share of the bit $sk_{\text{i}}, \text{i} \in [0, \lambda - 1]$ ($sk_{\text{i}}$ is the i$^{th}$ bit of the $\lambda$-bit secret key $sk$), denoted by $[\![sk_{\text{i}}]\!]_{\text{j}}$. A public bulletin board server is available to all the users and agents where they publish non-secret information and encrypted data whenever needed. The public key $pk$ is stored on the server, each agent $\mathsf{A}_{\text{IND}}$ makes a crypto-currency claim-or-refund deposit using a smart contract to the address associated with $pk$ with the condition $\tau$ embedded in it such that the deposit can be transferred if the condition is met or by anyone with the key $sk$.

**Document transfer.** When the user wants to use the TIE service for the message $m$, she splits the message/document into $\lambda$ parts ($\lambda$ is the bit length of secret key $sk$) and watermarks each part with robust bit watermarking to generate two versions of each part; i.e., for each part $m_{\text{i}}$ for $\text{i} \in [0, \lambda - 1]$, she computes two watermarked parts $(m_{\text{i},0}, m_{\text{i},1})$ with watermarks corresponding to bit 0 and 1. She symmetric-key encrypts each part $m_{\text{i},b}$ for $\text{i} \in [0, \lambda - 1]$ and $b \in \{0, 1\}$ using randomly sampled keys $\mathsf{k}_{\text{i},b}$ to obtain the ciphertexts $c_{\text{i},b}$.

She then performs 2−party computation with each of the servers such that each agent obtain SSS shares of key $\mathsf{k}_{\text{i},sk_{\text{i}}} = \mathsf{k}_{\text{i},0} + (\mathsf{k}_{\text{i},1} - \mathsf{k}_{\text{i},0})sk_{\text{i}}$ (Refer Section 4.3.2 for details), where the above equation is a representation of the standard oblivious transfer (OT) functionality. We realize this functionality by running a version of OT protocol where the input of agent $\mathsf{A}_{\text{j}}$ is share $[\![sk_{\text{i}}]\!]_{\text{j}}$ of the secret key bit $sk_{\text{i}}$ and the input of the user is the key pair $(\mathsf{k}_{\text{i},0}, \mathsf{k}_{\text{i},1})$. The user runs $\lambda$ such computations with each server such that the servers obtain key shares for $\lambda$ document parts. All $2\lambda$ ciphertexts $c_{\text{i},b}$ are published. When the agents collaborate, they can reconstruct the keys $\mathsf{k}_{\text{i},sk_{\text{i}}}$ and decrypt the corresponding ciphertexts $c_{\text{i},sk_{\text{i}}}$. However, even through collaboration, they will not be able to decrypt the ciphertexts $c_{\text{i},1-sk_{\text{i}}}$. See Figure 4.2 for an illustration of a transfer of the document from the user to the agents.

The aim of the user is to transfer an encrypted version of each document part such that the transferred part is watermarked with a secret key bit that is shared among all the agents. If the encrypted document part is decrypted, the decrypted part would reveal a secret key

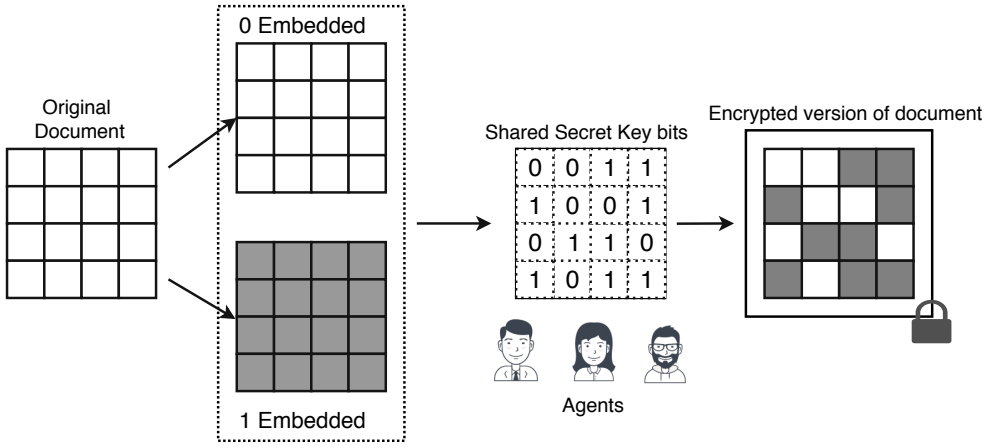



**Figure 4.2.** Watermarking the document blocks and transferring them to the agents. Two versions of each block are obtained by watermarking $0, 1$. The secret key bits are shared among the agents. Depending on the bit of the secret key, the transferred document block consists of the corresponding bit as the watermark. Thus final transferred encrypted document contains the whole secret key as the watermark.

bit to whoever can read the watermark. At a later point of time, when the agents decrypt the encryption $c_{\mathrm{i},sk_\mathrm{i}}$ of the watermarked message $m_{\mathrm{i},sk_\mathrm{i}}$, the detection key would be necessary to detect the watermark. The user forwards this detection key to a subset of agents of her choice as soon as she watermarks the message parts. Once the transfer of the keys and the two-party computation with each of the servers is performed, the interactive part of the user is complete. During the transfer, the agents prove in zero-knowledge to the user that the input share of each agent is indeed the share obtained by bit-wise sharing of the secret key.

When the condition is met (e.g.: time period expires), the agents can come together and reconstruct the keys $\mathsf{k}_{\mathrm{i},sk_\mathrm{i}}, \mathrm{i} \in [0, \lambda - 1]$ by combining the shares $[\![\mathsf{k}_{\mathrm{i},sk_\mathrm{i}}]\!]$. The cipher texts $c_{\mathrm{i},sk_\mathrm{i}}$ are decrypted using the reconstructed keys to reveal the message parts $m_{\mathrm{i},sk_\mathrm{i}}$. All the revealed message parts are combined to form a watermarked version $m'$ of the message/document $m$.

**Collusion and Key revelation.** The agents may decide to collude and decrypt the message $m$ by reconstructing the keys $\mathsf{k}_{(\cdot,\cdot)}$ even before the user condition is met. However, the decrypted message version $m'$ would contain the secret key $sk$ as a watermark. The two-party computation of oblivious transfer functionality ensures that each version of the message part that can be decrypted by the agents contains the secret key bit $sk_\mathrm{i}$ as a watermark.

When all the watermarked bits are read from the message parts, the secret key is revealed. Any agent who has access to the watermark detection key can read the secret key $sk$ (and any information encrypted to the public key $pk$) and transfer all the deposits to an address of his choice. This is the *revealing* and penalizing property of the protocol. When the funds are publicly transferred on the blockchain before the user condition is met, all the agents except the agent who performed the transfer are banned from the system.

The agents may also collude to reconstruct the secret key from the shares and transfer to an address different from any of the agents' addresses. In this case, the transfer is publicly visible and *all the agents* are banned from the system. Any evidence of collusion, including signature generated through the embedded key by multi party computation can result in banning of all the agents – if the deposit is not transferred by a single agent to his/her address. In case the agents try to attack by removing the watermark in the received documents, the robustness of the watermarking ensures that when the agents try to remove the watermark, the data itself is damaged or rendered useless. This is the property that necessitates the use of robust binary watermarking in our protocol.

With the penalizing and banning policy of the protocol, no rational agent would attempt to collude for the fear of loss of deposit and future service offering. As we will prove in Section 4.4, in the game induced by the protocol the equilibrium strategy of rational agents is to not collude. The threshold requirement of $t + 1$ where $t + 2$ agents are needed from reconstruction follows from the game-theoretic analysis in Section 4.4. The threshold of $t + 1$ prevents the adversary from publishing $t$ shares and influencing the equilibrium in the game. In the event of collusion among the agents, even if the agents agree not to transfer the deposit after collusion, any of the agents can unilaterally deviate from such an agreement and increase his pay-off by trying to transfer the deposit. Thus agents inevitably transfer the deposit (and act as whistle-blowers) after collusion.

We further elaborate few implicit assumptions made in the analysis in Section 4.4.

## 4.2 Related Work

Timed-release encryption (TRE) was first introduced by May [6]. Several applications and approaches using TRE have been proposed including sealed bids [139] electronic voting systems [140], spam and denial of service preventions [141] and proof-of-work systems [142]. For time-lock puzzles, a well known puzzle was created by Rivest *et al.* in 1996 [9] based on the RSA assumption which required non-parallelizable repeated squaring. Many other primitives based on their idea for time-lock-puzzles have been proposed including commitments [143], signatures [144], [145] and key escrow [146], [147]. Besides the RSA-based construction of a time-lock puzzles recent results from Bitansky et al. show constructions based on random encodings [148], [149]. Their security is based on the existence of non-parallelizing languages. The inherent problem with time-lock puzzles is that the time needed for solving the puzzle is dependent on the computing speed which can not be accurately predicted into the future.

In the category of schemes using trusted party, one of the first was presented by Rivest et al. [9] where a trusted party creates public keys for encryptions of messages and publishes the corresponding secret keys for different time-periods regularly. This idea was employed by Rabin and Thorpe with multiple trusted agents, using a distributed key generation [17] to distribute the secret key used for encryptions among different parties. There are other schemes based on different approaches, like the timed-release encryption scheme from Crescenzo *et al.* [7] which uses a trusted time-server and a newly created primitive called 'Conditional oblivious transfer'. They create an efficient protocol based on the quadratic residuosity assumption which in addition offers sender anonymity. Watanabe *et al.* used secret sharing where the dealer can choose a time. The shareholders can not reconstruct the secret shared before the time specified is over [150]. Many schemes based on identity based encryption were proposed, where the trusted key distribution center is also used for ensuring the correct time [8], [18], [151], [152]. In these models, any criterion which can be verified by the trusted party can be used as a reason to open a capsule.

For watermarking schemes, depending on the data type and application, many robust watermarking schemes have been proposed in the literature. Works such as [129], [130],

[131] present different audio watermarking schemes while works like [132], [133] deal with robust video watermarking. For software watermarking, schemes suggested in [134], [135] can be considered. The proposed IE protocol admits any robust watermarking scheme with no known attacks [107].

Halpern and Teague [153] introduce rational secret sharing where the agents sharing a secret are rational rather than honest or malicious. They show that at equilibrium, the agents do not contribute shares for reconstruction and propose a randomized protocol for performing the reconstruction. Gordon *et al.* [154] improve on the randomized protocol of [153] by overcoming the impossibility result. Lysyanskaya *et al.* [35] introduces the mixed behaviour model where the parties are either rational or adversarial, the authors provide a framework for multi-party computation in the proposed model. We adopt the mixed-behaviour model in this work.

Collusion in a multi-party setting to achieve collusion free MPC has been considered in works like [155]–[157], however, these works have stringent assumptions including the parties being co-located and communication only through a mediator. Recently, Ciampi *et al.* [158] define collusion proofness for multi-party functionalities, they assume availability of stateful trusted hardware tokens with each of the agents and parties communicate through authenticated broadcast channels. However, none of these works can be used for information escrows as agents can communicate over external channels and reconstruct the stored documents. In this work, we make no assumptions on communication or existence of envelopes and hardware tokens; we allow the agents to communicate freely on external channels and design the mechanism such that non-collusion is a Bayesian Nash Equilibrium for the system.

### 4.3 Cryptographic Construction

The $\mathsf{CDE}$ protocol consists of algorithms for generating the shares of the secret key $sk$, setting up the message blocks by the user, the distributed receiver oblivious transfer ($\mathsf{DROT}$) to transfer the message blocks to the agents and to open the message by the agents. The protocol with a user/sender $\mathsf{U}$ and $n$ agents $\mathsf{A}_\mathsf{j}, \mathsf{j} \in [1, n]$ constitutes the following algorithms:

- $\mathsf{KeySetup}(t, t, \lambda)$ generates a bit-wise shared secret key $sk$ for the $t$ participating agents with threshold $t+1$. It also generates the corresponding public key $pk$ and the other public components.

- $\mathsf{MessageSetup}(m, \lambda, pk)$ outputs $\lambda$ pairs of encryptions of binary-watermarked parts of the message $m$ encrypted to the public key $pk$.

- $\mathsf{DROT}\left((\mathsf{k}_{\mathrm{i},0}, \mathsf{k}_{\mathrm{i},1}), [\![sk_{\mathrm{i}}]\!]\right)$ The $\mathsf{DROT}$ protocol takes the keys $\mathsf{k}_{\mathrm{i},0}, \mathsf{k}_{\mathrm{i},1}$ from the sender and the shares of the bit $sk_{\mathrm{i}}$ from the receivers and transfers sharing $[\![\mathsf{k}_{\mathrm{i},sk_{\mathrm{i}}}]\!]$ corresponding to the bit $sk_{\mathrm{i}}$ to the receivers for $\mathrm{i} \in [0, \lambda-1]$.

- $\mathsf{Open}\left([\![sk_{\mathrm{i}}]\!], [\![\mathsf{k}_{\mathrm{i},sk_{\mathrm{i}}}]\!], c_{\mathrm{i},b}, b \in \{0,1\}\mathrm{i} \in [0, \lambda-1]\right)$ takes the shares of secret key bits along with the cipher texts, decrypts the ciphertexts forming the message blocks and outputs the final combined message.

### 4.3.1 Cryptographic Setup

$\mathsf{KeySetup}(n, t, \lambda, n)$. It provides the bit-wise shared secret key $sk$ using $(t, t+1)$-secret sharing and the corresponding public key to the $n$ agents. The algorithm first generates the public parameters using $\mathsf{grp.gen}(\cdot)$ which takes the security parameter $\lambda$ as input. It generates the cyclic group $\mathbb{G}$ of prime order $p$ and two generators $g, h$. The two generators are used for Pedersen commitments. The agents run the distributed key generation (DKG) algorithm $\mathsf{dkg.gen}(\cdot)$ to generate the public key $pk$ and the vector of secret key shares $[\![sk]\!]$, with each agent $\mathsf{A}_{\mathrm{IND}}$, $\mathrm{IND} \in [1, n]$ obtaining the share $[\![sk]\!]_{\mathrm{IND}}$. The agents run a bit decomposition algorithm $\mathsf{bit.decomp}(\cdot)$ with the shares to obtain the bit-wise threshold-shares $[\![sk_{\mathrm{i}}]\!]_{\mathrm{IND}}$ of the bits $sk_{\mathrm{i}}, \mathrm{i} \in [0, \lambda-1]$ for each agent. Here, $n$ is the security parameter and $\lambda = poly(n)$. Algorithm 2 depicts algorithm $\mathsf{KeySetup}$.

**$\mathsf{MessageSetup}(m, \lambda, k_{emd})$.** It is run by the sender who takes the message $m$ and uses an algorithm $\mathsf{split}(m, \lambda)$ to divide it into $\lambda$ parts $(m_0, \ldots, m_{\lambda-1})$ where $\lambda$ is the bit-length of the secret key $sk$. The sender forms the robust binary watermarked versions of the message parts (using $\mathsf{wm.embed}(\cdot)$) with the watermark embedding key $k_{emd}$, watermarked with $\{0,1\}$ to obtain $\{(m_{0,0}, m_{0,1}), \cdots, (m_{\lambda-1,0}, m_{\lambda-1,1})\}$ and encrypts them using the randomly

**Algorithm 2** KeySetup $(n, t, \lambda, n)$

$\mathbb{G}, g, h, p \leftarrow \mathsf{grp.gen}(\lambda)$
$([\![sk]\!], pk) \leftarrow \mathsf{dkg.share}(t, t, \lambda)$
$([\![sk_0]\!], \cdots, [\![sk_{\lambda-1}]\!]) \leftarrow \mathsf{bit.decomp}(\lambda, [\![sk]\!])$
$k_{emd}, k_{det} \leftarrow \mathsf{wm.gen}(n)$

**Algorithm 3** MessageSetup$(m, \lambda, k_{emd})$

$(m_0, \ldots, m_{\lambda-1}) \leftarrow \mathsf{split}(m, \lambda)$
**for** $\mathrm{i} = 0 \ldots \lambda - 1$ **do**
    $m_{\mathrm{i},0} \leftarrow \mathsf{wm.embed}(m_{\mathrm{i}}, 0, k_{emd})$
    $m_{\mathrm{i},1} \leftarrow \mathsf{wm.embed}(m_{\mathrm{i}}, 1, k_{emd})$
    $\mathsf{k}_{\mathrm{i},0}, \mathsf{k}_{\mathrm{i},1} \xleftarrow{\$} \mathcal{K}$; $\mathcal{K}$ – key space
    $c_{\mathrm{i},0} \leftarrow \mathsf{enc}(\mathsf{k}_{\mathrm{i},0}, m_{\mathrm{i},0})$
    $c_{\mathrm{i},1} \leftarrow \mathsf{enc}(\mathsf{k}_{\mathrm{i},1}, m_{\mathrm{i},1})$

**Figure 4.3.** KeySetup and MessageSetup algorithms

chosen symmetric keys $\mathsf{k}_{\mathrm{i},b}$ to produce $c_{\mathrm{i},b}, \mathrm{i} \in [0, \lambda - 1], b \in \{0, 1\}$. The $\mathsf{split}(\cdot)$ is a simple split/chopping of the message into parts. However, if the watermarking scheme allows reconstruction of the data, the split can be a secret sharing based split outputting secret shares of the message. Algorithm 3 depicts algorithm MessageSetup.

### 4.3.2 Distributed Receiver Oblivious Transfer—DROT

Oblivious transfer is a 2-party computation protocol, in which the sender has two messages $m_0, m_1$ and the receiver has the bit $c$; at the end of the protocol, the receiver receives the message $m_c$. The protocol ensures that sender has no information of $c$ and the receiver has no information about $m_{1-c}$. We develop a multi-party version of oblivious transfer called the *Distributed Receiver Oblivious Transfer* protocol (DROT) which is used in the APBprotocol to transfer the document to the agents. The DROT protocol involves $n + 1$ parties with one sender and $n$ receivers. The sender has the messages $\mathsf{k}_0, \mathsf{k}_1$ and the receivers have the shares $[\![s]\!]$ for the bit $s$. At the end of the computation, each of the receivers receives the shares $[\![\mathsf{k}_s]\!]$. The receivers can reconstruct $\mathsf{k}_s$ by collaboration, however, they cannot reconstruct $\mathsf{k}_{1-s}$. This is similar to the standard oblivious transfer protocol in which the other value $m_{1-c}$ cannot be computed by the receiver.

**Ideal functionality of DROT.** The functionality (refer Figure 4.4) interacts with the sender S and the $n$ receivers $\mathsf{A}_\mathrm{j}, \mathrm{j} \in [1, n]$. S has the messages $\mathsf{k}_0, \mathsf{k}_1$ and each receiver $\mathsf{A}_\mathrm{j}$ has the share $[\![s]\!]$ of the bit $s$. The adversary $\mathcal{A}$ can corrupt a total of $t$ parties in the system — the sender or upto $t$ receivers or sender and $t - 1$ receivers. The sender initiates

**Functionality $\mathcal{F}_{DROT}$**

The functionality $\mathcal{F}_{\mathsf{DROT}}$ interacts with the sender $\mathsf{S}$ and $n$ receivers $\mathsf{A}_\mathsf{j}, \mathsf{j} \in [1, n]$. Each receiver $\mathsf{A}_\mathsf{j}$ has share $[\![s]\!]_\mathsf{j}$ of a random, unknown secret bit $s$ and the sender $\mathsf{S}$ has two messages $\mathsf{k}_0, \mathsf{k}_1$. A maximum of $t$ parties in the system can be corrupted by the adversary $\mathcal{A}$.

- Upon receiving the message $(\texttt{inputS}, \mathsf{k}_0, \mathsf{k}_1, sid)$ from sender $\mathsf{S}$, record $\langle \mathsf{k}_0, \mathsf{k}_1, sid \rangle$, forward the message $(\texttt{intd}, sid)$ to the receivers $\mathsf{A}_\mathsf{j}$ and $(\texttt{input}, sid)$ to $\mathcal{A}$.
- Upon receiving the message $(\texttt{inputR}, [\![s]\!]_\mathsf{j}, \mathsf{j}, sid)$ from receiver $\mathsf{A}_\mathsf{j}$, add $\langle \mathsf{j}, [s]_\mathsf{j} \rangle$ to the set $\mathcal{I}$, forward the message $(\texttt{input}, sid)$ to $\mathcal{A}$. When $\|\mathcal{I}\| \geq t + 2$, compute the secret bit $s$ from the shares $[\![s]\!]_\mathsf{j}$.
- Upon receiving the message $(\texttt{deliverS}, sid)$ from $\mathcal{A}$, check if $\langle \mathsf{k}_0, \mathsf{k}_1, sid \rangle$ is stored and $\|\mathcal{I}\| \geq t + 2$, else ignore the message. Send $(\texttt{delivered}, sid)$ to $\mathsf{S}$.
- Upon receiving the message $(\texttt{deliverR}, sid)$ from $\mathcal{A}$, check if $\langle \mathsf{k}_0, \mathsf{k}_1, sid \rangle$ is stored and $\|\mathcal{I}\| \geq t + 2$, else ignore the message. Generate shares $[\![\mathsf{k}_s]\!]_\mathsf{j}$ of the value $\mathsf{k}_s$. Forward $(\texttt{output}, [\![\mathsf{k}_s]\!]_\mathsf{j}, sid)$ to $\mathsf{A}_\mathsf{j}, \mathsf{j} \in \mathcal{I}$.

**Figure 4.4.** Ideal Functionality Of DROT

by forwarding the input by sending the `inputS` message with the two messages $\mathsf{k}_0, \mathsf{k}_1$ and the session id $sid$ to $\mathcal{F}_{\mathsf{DROT}}$. The functionality stores them and informs the adversary by sending the $(\texttt{input}, sid)$ message. It also informs the receivers that the protocol has been initiated by sending the $(\mathsf{intd}, sid)$ messages. The receivers forward their shares $[\![s]\!]_\mathsf{j}$ using the message `inputR` to $\mathcal{F}_{DROT}$, whose id j is stored in the set $\mathcal{I}$. After receiving at least $t+2$ shares, the functionality computes the bit $s$ by combining the shares $[\![s]\!]_\mathsf{j}$. When $\mathcal{A}$ sends the $(\texttt{deliverS}, sid)$ message, the functionality checks if the inputs from the sender and at least $t + 2$ receivers have been received. If not, the message is ignored. The sender is informed by forwarding the $(\texttt{delivered}, sid)$ message to $\mathsf{S}$. $\mathcal{A}$ sends the $(\texttt{deliverR}, sid)$ message to release the output to the receivers. On receiving the `deliverR` message, the functionality checks if $\|\mathcal{I}\| \geq t + 2$, if at least $t + 2$ receivers have forwarded to release the output. If yes, it generates shares of the value $\mathsf{k}_s$ and forwards the share $[\![\mathsf{k}_s]\!]_\mathsf{j}$ to the receivers $\mathsf{A}_\mathsf{j}, \mathsf{j} \in \mathcal{I}$.

**Protocol.** We realize DROT using multiple instances of two-party computation, realizing an oblivious linear function evaluation (OLE) [96] between the sender and the receivers. In

DROT, the OLE computation is run by the sender with all the $n$ receivers $\mathsf{A}_\mathsf{j}, \mathsf{j} \in [1, n]$. The input of the sender is messages $(\mathsf{k}_0, \mathsf{k}_1)$ and the input of each receiver is share $[\![s]\!]_\mathsf{j}$ of the bit $s$. $[\![s]\!]_\mathsf{j}$ are $(n, t+1)$ shared values of the value $s$. After the DROT protocol run, the receivers obtain the value $\mathsf{k}_0 + (\mathsf{k}_1 - \mathsf{k}_0) \cdot [\![s]\!]_\mathsf{j}$. These are the shares $[\![\mathsf{k}_s]\!]_\mathsf{j}, \mathsf{j} \in [1, n]$ of the value $\mathsf{k}_s$, such that any $t+2$ or more parties can reconstruct the value $\mathsf{k}_s$. The receivers prove in zero-knowledge that the input shares indeed correspond to a bit $s$.

The CDE protocol uses the DROT protocol to transfer the encrypted message blocks and the corresponding keys' shares to the agents. The user encrypts the message blocks $m_{\mathsf{i},b}$, using keys $\mathsf{k}_{\mathsf{i},b}$ to obtain $c_{\mathsf{i},b}$ for $\mathsf{i} \in [0, \lambda - 1]$ and $b \in \{0, 1\}$ and broadcasts the ciphertexts to all the agents. User transfers shares of keys $\mathsf{k}_{\mathsf{i},sk_\mathsf{i}}$ to the agents using DROT. As a part of DROT the user runs $\lambda$ instances of OLE— for each instance i, the input of the user is $(\mathsf{k}_{\mathsf{i},0}, \mathsf{k}_{\mathsf{i},1})$ where the inputs of the agents would be shares $[\![sk_\mathsf{i}]\!]_\mathsf{j}$ of the bit $sk_\mathsf{i}$ of the signing key $sk$. At the end of the runs, each agent $\mathsf{A}_\mathsf{j}$ has the values $[\![\mathsf{k}_{\mathsf{i},sk_\mathsf{i}}]\!]_\mathsf{j}$ which are shares of the values $\mathsf{k}_{\mathsf{i},sk_\mathsf{i}} = \mathsf{k}_{\mathsf{i},0} - (\mathsf{k}_{\mathsf{i},1} - \mathsf{k}_{\mathsf{i},0}) \cdot sk_\mathsf{i}$. While running DROT for CDE, the agents prove to the user that the input bit shares indeed correspond to bits of the secret key $sk$ by forwarding a zero knowledge proof $\pi_{sk}$. This way of directly computing the shares of the keys $\mathsf{k}_{\mathsf{i},sk_\mathsf{i}}$ prevents the agents from learning the other key corresponding to $\mathsf{k}_{\mathsf{i},1-sk_\mathsf{i}}$. The oblivious transfer functionality can be realized using standard oblivious transfer primitives, however, they involve computing hash functions in a distributed manner which is computationally intensive. Realizing the oblivious transfer functionality as oblivious linear function evaluation using secret sharing based 2-party computation avoids the computational bottlenecks.

The security analysis of DROT is discussed in Section 4.5.

### 4.3.3 Post-processing

**Open.** The algorithm open decrypts a message with the collaboration of $t+2$ or more agents. The agents combine their shares for every key bit $[\![\mathsf{k}_{\mathsf{i},sk_\mathsf{i}}]\!], \mathsf{i} \in [0, \lambda - 1]$ for each i and the corresponding ciphertext $c_{\mathsf{i},sk_\mathsf{i}}$ is decrypted to $m_{\mathsf{i},sk_\mathsf{i}}$. After all parts of the messages $m_{\mathsf{i},sk_\mathsf{i}}$ are recovered, they are combined to form the final message $m'$ which is a watermarked

**Algorithm 4** Open $([\![sk_{\mathrm{i}}]\!], [\![\mathsf{k}_{\mathrm{i},sk_{\mathrm{i}}}]\!], c_{\mathrm{i},b}, b \in \{0,1\}\mathrm{i} \in [0, \lambda - 1])$

1: **for** $\mathrm{i} = 0 \ldots \lambda - 1$ **do**
2: $\quad sk_{\mathrm{i}} \leftarrow \mathsf{bit.recon}([\![sk_{\mathrm{i}}]\!])$
3: $\quad \mathsf{k}_{\mathrm{i},sk_{\mathrm{i}}} \leftarrow \mathsf{dkg.recon}([\![\mathsf{k}_{\mathrm{i},sk_{\mathrm{i}}}]\!])$
4: $\quad m_{\mathrm{i},sk_{\mathrm{i}}} \leftarrow \mathsf{dec}(c_{\mathrm{i},sk_{\mathrm{i}}}, \mathsf{k}_{\mathrm{i},sk_{\mathrm{i}}})$
5: $m' \leftarrow \mathsf{combine}(m_{0,sk_0}, \ldots, m_{\lambda-1,sk_{\lambda-1}})$
6: **return** $m'$

version of $m$. $m'$ contains the whole of the secret key $sk$ as watermark embedded in it. Algorithm 4 presents algorithm Open.

**Detection key distribution.** In the protocol, the user forwards the detection keys of the watermark she employed to some of the agents to ensure that if the agents decide to decrypt the user message, the agents with the watermark detection key will be able to detect the watermark $sk$ embedded in the document. These agents will be able to transfer the deposit of all other agents. The user distributes the correct detection key to a subset of agents such that any subset of $t + 2$ agents has at least one agent with the detection key. The user forwards either a correct or an incorrect (dummy) key to all the agents, who can not distinguish if the received key is a correct detection key unless they collude and decrypt the document.

### 4.3.4 Escrow deposits

Before the start of the protocol, the user and the agents agree on the deposit value $D$. The agents proceed to deploy smart-contracts embedding the condition $\tau$ specified by the user to the address corresponding to the public key $pk$. Each agent $\mathsf{A}_{\mathrm{j}}$ creates a deposit transaction $TX_{\mathrm{j}}$ for the value $D$ and the user verifies that the agreed-on values of funds have been held in the deposit. When the condition $\tau$ is met, the deposits can be immediately transferred back to the agents. Any agent with $sk$ can claim/transfer all the deposits to the address of his choice before the condition $\tau$ is met. If the deposit is ever transferred before $\tau$, all the agents except the agent that transferred the deposit are banned from offering any

future service thereby penalizing them. This ban can be either permanent or for a specified period of time.

In the non-colluding scenario, after the condition $\tau$ is met, we expect the agents to take a non-zero time to compute the secret key $sk$. Indeed, we expect that the deposits are transferred back instantaneously to all the agents by the smart contracts before the $sk$ is computed by the agents once $\tau$ is met.

Before the transfer of the document by the user, she includes partial payment in the cryptocurrency deposit smart contract which will be paid to the agents when the condition is met and if the agent deposits are not transferred by then. This payment can be in addition to partial initial payment made by the client to the agents for the service.

As the agents provide strong identities linked to their physical identities, they cannot launch any Sybil attack to target and ban other agents.

### 4.4 Game-theoretic Modelling and Analysis

We model and analyze the collusion deterrent escrow ($\mathsf{CDE}$) protocol as a mechanism through which the user induces a game between the agents offering the service. Two collusion scenarios are possible: (i) collude and reconstruct the keys such that the document can be decrypted (ii) reconstruct the secret key $sk$ from the shares. These two scenarios are modeled as collusion strategies played by the agents. For simplicity, we initially analyze the scenario with one regulator (user) and two agents $\mathsf{A}_1, \mathsf{A}_2$.

**Further assumptions.** : Before we analyze the system, we mention some of the (implicit) assumptions under which the proposed protocol is applicable:

- The agents provide physical verifiable identities before offering the service such that banning is effective. This prevents the banned agents from offering the service again under a different pseudonym.

- When multiple agents try to transfer the deposit after collusion, each agent transfers (wins the race) with equal probability. We make this assumption for the ease of analysis.

**Table 4.1.** (Trivial) Pay-off matrix for the two agent threshold escrow *without* collusion-deterrence. $v$ is the fee offered by the user and $d$ is the value of the escrowed document to the agents.

| $A_2$ \ $A_1$ | *reject* | *accept (not collude)* | *accept and collude* |
|---|---|---|---|
| *reject* | (0,0) | (0,0) | (0,0) |
| *accept (not collude)* | (0,0) | $(v, v)$ | $(v, v)$ |
| *accept and collude* | (0,0) | $(v, v)$ | $(v + \frac{d}{2}, v + \frac{d}{2})$ |

However, if the probabilities are different, it deters the agents further to collude since the agents do not know if they have higher/lower probability of success.

- The sum total of all the future payments ($F$) that are receivable by the agents for the future services to be offered is greater than the value $v$ of individual documents encrypted for any time period.

When the user engages the agents for the escrow service, she induces a trivial game between the agents. They have a choice of either accepting to offer the service or rejecting, indicated by *accept* or *reject* in Table 4.1. If both the agents accept, the user offers a transfer of value $v$ to each of the agents. This results in a trivial pay-off matrix as shown in Table 4.1 with only the *accept* and *reject* strategies. *accept* indicates the strategy of offering the service without collusion. Each of the agents makes a deposit of value $D$ before the user interacts with them.

The agents are free to interact, they can collude with each other to open the user's escrow message and share the value of the document among themselves. We assume the agents are symmetrical and value the document equally. If $d$ is the value of the document and if agents are assumed to share the value equally under collusion, the pay-off of each of the agents is $\alpha = v + \frac{d}{2}$. This extends the pay-off matrix in Table 4.1 to include the *collude* strategy. If only one of the agents wishes to collude and the other does not, collusion does not occur and the pay-off of each is still $v$. Since the agents accrue a pay-off strictly greater than $v$, colluding is a dominant strategy equilibrium of the game [159] and hence both the agents play the *accept and collude* strategy at equilibrium.

To prevent such a collusion attack and to prevent *accept and collude* as the equilibrium strategy, the user who acts as a principal/regulator designs a mechanism implemented by offering the grand contract. In our protocol, the agents collude to attack the system either by decrypting the document or by reconstructing the secret key. These two cases are considered as the collusion scenarios inducing two collusion games among the agents.

Along with the document/data, the user(regulator) transfers information types $\{\mathcal{I}, \mathcal{D}\}$ to the agents such that the agent with information $\mathcal{I}$ is considered the efficient agent and the agent with $\mathcal{D}$, the inefficient agent. The typeset $\Theta = \{\mathrm{e}, \mathrm{ie}\}$ indicates efficient and inefficient agents. The regulator forwards the information $\mathcal{I}$ (resp. $\mathcal{D}$) with probability $p$ (resp. $1-p$) to each of the agents. Thus the user induces an *information asymmetry* among the agents. One can note that, unlike a typical setting, in this model the regulator induces types on the agents.

In the implementation of the game as a protocol, the information $\mathcal{I}$ would correspond to the correct detection key and $\mathcal{D}$, an incorrect detection key. $D > 0$ is the value of conditional deposit made by each agent and $F$ is the approximate sum of future payments to be received if the agent/node is not banned from the system. The two collusion scenarios Collusion-1, Collusion-2 would be collusion scenarios using document decryption and secret key ($sk$) reconstruction respectively. In the protocol, when the deposits are transferred before the condition is met, whoever transfers the deposits can gain the value $D$ (deposit of the other agent apart from getting back own deposit). However, every agent *except* the agent that performs the transfer is banned from the system. Getting banned would make the agents lose all future payments/pay-offs that they can receive from other clients/users whose total is approximated to a value $F \gg 0$.

### 4.4.1 Collusion-game-1

When the agents decide to collude and threshold-decrypt the document, we call it Collusion-1 and the game induced is Collusion-game-1. The following parameters define the game.

- The set of agents $N = \{\mathsf{A}_1, \mathsf{A}_2\}$

- The typeset of the agents $\Theta = \{\text{e}, \text{ie}\}$
- The strategy space of each agent IND is $S_{\text{IND}} = \{wait_1, transfer_1\}$
- The utilities corresponding to the strategy and the type of the agent $u_{\text{IND}}(\theta_{\text{IND}}, s_{\text{IND}}, s_{-\text{IND}}), \theta_{\text{IND}} \in \Theta, s_{\text{IND}} \in S_{\text{IND}}$. $s_{-\text{IND}}$ indicates the strategy(ies) of player(s) other than player j.

**Table 4.2.** Pay-offs of rational agents under different strategies in Collusion-1-document decryption (strictly dominant strategies indicated in grey)

| $A_2$ \ $A_1$ | | Efficient($e$) | | Inefficient($ie$) |
|---|---|---|---|---|
| | | $wait_1$ | $transfer_1$ | $wait_1$ |
| Efficient ($e$) | $wait_1$ | $(\alpha, \alpha)$ | $(\alpha + D, \alpha - F - D)$ | $(\alpha, \alpha)$ |
| | $transfer_1$ | $(\alpha - F - D, \alpha + D)$ | $(\alpha - \frac{F}{2}, \alpha - \frac{F}{2})$ | $(\alpha - F - D, \alpha + D)$ |
| Inefficient ($ie$) | $wait_1$ | $(\alpha, \alpha)$ | $(\alpha + D, \alpha - F - D)$ | $(\alpha, \alpha)$ |

When the agents/nodes collude and decrypt the document version (Collusion-1), the agent that has access to the watermark detection key can 'transfer' the deposit, however, he can choose not to transfer the deposit and 'wait'. These two actions are indicated by $transfer_1$ and $wait_1$. The agent that does not have the detection key can not transfer the deposit and hence can only wait after collusion.

**Payoffs.** The expected pay-offs of the agents under Collusion-1 is captured by the pay-off matrices in Table 4.2. An efficient agent (e) has the correct detection key and inefficient agent (ie) does not. When both agents are efficient and decrypt the user document and not transfer the deposit (play $wait_1$), they both can accrue a pay-off of $\alpha = v + \frac{d}{2}$. If one of the agents transfers (plays $transfer_1$) the deposit, he gets a pay-off of $\alpha + D$ where as, the other agent loses his deposit and gets banned thereby accruing a pay-off of $\alpha - (F + D)$. In the case when both the efficient agents attempt to transfer the deposit simultaneously, since only one agent can succeed in the transfer, it creates a race condition and we assume that each will succeed in the transfer with equal probability

[3]; hence in expectation they accrue a pay-off $0.5(\alpha + D) + 0.5(\alpha - F - D) = \alpha - \frac{F}{2}$. $u_{\text{IND}}(\text{e}, \textit{transfer1}, s_{-\text{IND}}) > u_{\text{IND}}(\text{e}, \textit{wait1}, s_{-\text{IND}}) \forall s_{-\text{IND}} \in \{\textit{transfer}_1, \textit{wait}_1\}$ (from Table 4.2). For an efficient agent $\textit{transfer}_1$ is a strictly dominant strategy, he always plays $\textit{transfer}_1$. For the inefficient agent, $\textit{wait}_1$is the only strategy available to play in Collusion-1, trivially, it is the dominant strategy. The dominant strategies have been indicated in grey in Table 4.2, the pay-off submatrix of Table 4.2 when both the agents are efficient are similar to the pay-offs in well known prisoners' dilemma [160]. The agents playing their dominant strategy for the given type according to the payoff matrix of Table 4.2 indeed forms the Bayesian Nash Equilibrium as defined below.

**Definition 4.4.1** (Bayesian Nash Equilibrium [161]). *A strategy profile $s(\cdot)$ is a Bayesian Nash equilibrium if $Eu_{\text{j}}(s_{\text{j}} \| s_{-\text{j}}, \theta_{\text{j}}) \geq Eu_{\text{j}}(s'_{\text{j}} \| s_{-\text{j}}, \theta_{\text{j}})$ for all $\theta_{\text{j}} \in \Theta_{\text{j}}$, for all $s'_{\text{j}}(\theta_{\text{j}}) \in S_{\text{j}}$ where the expected utility $Eu_{\text{j}}(s_{\text{j}} \| s_{-\text{j}}, \theta_{\text{j}}) = \sum_{\theta_{-\text{j}} \in \Theta_{-\text{j}}} u_{\text{j}}(s_{\text{j}}, s_{-\text{j}}(\theta_{-\text{j}}), \theta_{\text{j}}, \theta_{-\text{j}}) p(\theta_{-\text{j}} \| \theta_{\text{j}})$.*

**Expected payoff.** The regulator chooses each agent and transfers information $\mathcal{I}$ with probability $p$ independently. Hence the expected payoff of each agent playing their dominant strategies (Table 4.2) is:

$$\begin{aligned} \hat{a} &= p^2(\alpha - \frac{F}{2}) + p(1-p)(\alpha + D) + (1-p)p(\alpha - F - D) + (1-p)^2\alpha \\ &= \alpha(p^2 + 2p(1-p) + (1-p)^2) - \frac{p^2 F}{2} - (p - p^2)F = \alpha - F(p - \frac{p^2}{2}) \end{aligned} \quad (4.1)$$

### 4.4.2 Collusion-game-2

In the collusion scenario where the agents reconstruct the secret key (Collusion-2), the game induced is Collusion-game-2. The agents can either choose to transfer the deposit – play $\textit{transfer}_2$ or wait without transferring – play $\textit{wait}_2$. The actions do not depend on the availability of the detection key and hence do not depend on the type of the agent. The following parameters define the game.

- The set of agents $N = \{\text{A}_1, \text{A}_2\}$

[3]↑If we assume different probabilities, the agent succeeding with lesser probability has lesser pay-off and is further dis-incentivzed from collusion

**Table 4.3.** Pay-offs of rational agents under different strategies in Collusion-2-secret key reconstruction

| $A_2$ \ $A_1$ | $wait_2$ | $transfer_2$ |
|---|---|---|
| $wait_2$ | $(\alpha, \alpha)$ | $(\alpha + D, \alpha - F - D)$ |
| $transfer_2$ | $(\alpha - F - D, \alpha + D)$ | $(\alpha - \frac{F}{2}, \alpha - \frac{F}{2})$ |

- The strategy space of each agent IND is $S_{\text{IND}} = \{wait_2, transfer_2\}$
- The utilities corresponding to the strategy.

The obtained pay-offs of different actions in Collusion-game-2 are presented in Table 4.3. After the reconstruction of the secret key, if both the agents wait and do not attempt to transfer the deposit (play $wait_2$), both the agents obtain a pay-off of $\alpha$, however, if one of the agents transfers the deposit (plays $transfer_2$), he obtains $\alpha + D$ whereas the other agent obtains a payoff of $\alpha - (F + D)$. In the case when both the agents attempt to transfer the deposit, they obtain an expected payoff of $\alpha - \frac{F}{2}$. Again, playing $transfer_2$ is a strictly dominant strategy of every agent in this collusion scenario, and hence the agents always play $transfer_2$. Both the agents play $transfer_2$ as their dominant strategy, accruing a pay-off of $b$ as shown in Table 4.3 where,

$$\hat{b} = \alpha - \frac{F}{2} \tag{4.2}$$

**Maximum pay-off from collusion.** It can be seen from Equation (4.1), Equation (4.2) the maximum expected pay-off that can be obtained by each agent through either collusion scenarios is

$$\beta = \max(\hat{a}, \hat{b}) = \max\Big((\alpha - F(p - \frac{p^2}{2})), (\alpha - \frac{F}{2})\Big) \tag{4.3}$$

As $F$ is the sum of all future payments from many users, we have, $F \gg D, F \gg d, v$. The regulator or the user sets the value of $p$ such that $\beta < v$. The expected pay-off from the collusion game $\beta$ is strictly less than $v$ making collusion unviable.

*Bargaining*: Apart from the two collusion scenarios with the actions defined, each agent may indulge in bargaining/bribing by offering a positive payment through the side contract to other agents to prevent them from playing $transfer_2$. We assume that the other agents

can not impose a negative pay-off on the whistle blower (playing $transfer_1/transfer_2$) outside the system. Hence the agent will *always* play his dominant strategy of transfer to improve his pay-off even after accepting a payment from the other agent. From the strictly dominant strategies, it is clear that whenever the agents decide to collude, they always attempt to transfer the deposit and act as whistle-blowers in the protocol irrespective of the actions of other agents.

**Theorem 4.4.1.** *In CDE protocol under Bayesian Nash equilibrium (ref Definition 4.4.1) with $n = 2$, the agents do not collude; for $n > 2$, no more than threshold $t+1$ agents collude when the secret key is shared in a $(n, t+1)$ threshold structure (at least $t+2$ agents are required to obtain secret information).*

For $n = 2$ when the agents collude, the expected pay-off as computed in Equation (4.3) is $\beta = \max\Big((\alpha - F(p - \frac{p^2}{2})), (\alpha - \frac{F}{2}\Big)$. Since $F \gg d, v, \alpha = v + \frac{d}{2}$, we have $\beta \ll v$. The maximum expected pay-offs when the agents play accept and collude are $(\beta, \beta)$. Thus the trivial pay-off matrix of Table 4.1, under the CDE protocol mechanism changes to Table 4.4. Since $\beta < v$, it is evident from Table 4.4 that *accept and collude* is not an equilibrium as agents can deviate unilaterally and improve their pay-offs. Thus at equilibrium the agents do not collude. When the number of agents is $n$ and the secret is shared with $(n, t+1)$ threshold secret sharing, collusion occurs only when at least $t+2$ agents collude with each other. When $t$ agents play the collusion strategy, no rational $t + 2^{nd}$ agent plays accept and collude as his expected pay-off will drop from $v$ to $\beta$. Thus irrespective of the $t+1$ agents, no rational agent attempts to collude and collusion is not an equilibrium in the pay-off matrix. The equilibrium pay-off of the agents is $v$.

This is also evident from the extensive-form game depiction of the overall game played by the agents as shown in Figure 4.5. 'Collude' indicates the two collusion scenarios and the pay-off is the maximum obtained from either of the two scenarios when the agents play the collusion strategy. The leaf nodes are associated with the pay-offs of the agents for a run of the game. As evident from the last decision node of the tree, player 2 never chooses to collude when agent 1 plays *accept and collude* as *accept, not collude* offers strictly higher pay-off. Similarly, when the game tree is formed for $n$ agents, at a node where $t+1$ agents

**Table 4.4.** Pay-off matrix of the two agents under CDE. $v$ is the pay-off when agents do not collude and $\beta \ll v$ is the pay-off when the agents collude.

| $A_2$ \ $A_1$ | *reject* | *accept (not collude)* | *accept and collude* |
|---|---|---|---|
| *reject* | (0,0) | (0,0) | (0,0) |
| *accept (not collude)* | (0,0) | $(v, v)$ | $(v, v)$ |
| *accept and collude* | (0,0) | $(v, v)$ | $(\beta, \beta)$ |

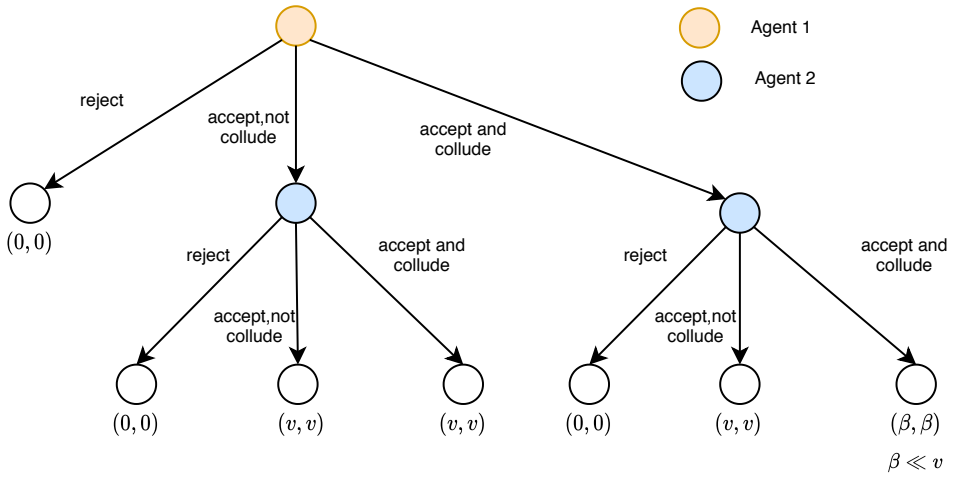



**Figure 4.5.** Extensive form game induced by the user in CDE between two agents. $v$ is the pay-off when agents do not collude and $\beta \ll v$ is the pay-off when the agents collude

collude, the $t + 2^{nd}$ agent chooses not to collude as the equilibrium strategy. It can be seen that 'accept and not collude' of every agent is the strategy that survives the iterated deletion of weakly dominated strategies and hence the unique equilibrium strategy profile of the game consists of each player playing 'accept and not collude'.

**After collusion.** Tables 4.2, 4.3 show that during collusion irrespective of what the other agents' strategy is, the dominant strategy of any rational agent is to attempt to transfer the deposit to himself. Thus whenever collusion occurs (with $> t + 2$ parties) irrespective of which set of agents collude during the collusion phase, every colluding rational agent attempts to transfer to himself leading to an expected pay-off strictly lower than obtained without collusion. The adversary controlling $t$ agents can approach any rational agent to reveal all

the secret shares with the adversary, however, since the threshold of secret information is $t+1$, at least 2 rational agents need to participate in collusion along with the adversary to transfer all the deposit. If more than $t+1$ agents participate in collusion, the expected pay-off is strictly less than without collusion. Thus no rational agent participates in the collusion.

In works involving rational secret sharing[153], [154], the authors show with the utility structure where the parties do not prefer others to know the secret information, the parties do not reconstruct the shared secret in equilibrium. In CDE protocol, we have a similar utility function structure, however, in this protocol not sharing the information (here, not colluding) survives the iterated deletion of weakly dominant strategies and hence an equilibrium. What is a road-block in works like [153], [154], [162] is actually made use of as an advantage in the CDE protocol. *After* the condition set by the user is met, the utilities of the agents change such that other agents learning the secret information does not affect the agent and the equilibrium strategy would be to decrypt and obtain the pay-off.

### 4.4.3 Remarks

We now discuss two practical issues:

**Multiple-rounds.** : Even though the analysis considers only one round of play of the game, since collusion and corresponding banning of agents occurs only once for a set of agents, the same analysis can be used to depict multiple-rounds by appropriately modifying the values of $d, v, \beta, F$. As 'future' always exists and the longer the system runs the longer can be the future service offering, the value of $F$ is still higher even if values of $d, v, \beta$ are considered cumulatively for multiple rounds and multiple documents. Once the whistle-blower transfers the deposit, all other colluding parties are banned from the system and the whistle-blower can join another set of agents to continue offering the service without any damage/change to his reputation.

**Partial Decryption.** : The receivers may try to decrypt the document partially (and not the full document) so that the agents with the detection key will not be able to detect the whole secret key watermarked in the document. This can be prevented by the user as follows:

she can split the document into much more number of parts (multiples of $\lambda$ instead of just $\lambda$) and transfer them to the agents. Through this, she embeds multiple copies of the secret key in the document such that any small decrypted portion contains the whole secret key as the watermark. She can introduce dummy blocks, randomize them in the total message blocks and also embed multiple copies of the key in the part of the document which has more information or high entropy, whereby decrypting any useful part of the document reveals the full key. Also, when an agent with the detection key obtains the key partially, he can brute force the remaining bits of the key whenever possible. The exact number of copies of the secret key to be embedded depends on the entropy of the document and the information it contains, the analysis of which is beyond the scope of this document.

## 4.5 Security Analysis

### 4.5.1 Security Definition

The system consists of $t$ agents $\mathsf{A}_\mathsf{j}, \mathsf{j} \in [1, n]$ and a user $\mathsf{U}$ with an input $m_\mathsf{U}$. The agents and the user are interactive Turing machines that communicate with an ideal functionality. The adversary is a PPT machine with access to a corrupt interface that takes an agent/user identifier and returns the internal state of the agent to the adversary. All subsequent incoming and outgoing communication of the agent is then routed through the adversary. The adversary is $t$-bounded, and can corrupt up to $t$ agents and the user. For formal security, as discussed earlier, we consider the static corruption model i.e., the adversary commits to the identifiers of the agents it wishes to corrupt ahead of time. The adversary is also informed whenever some communication happens between two agents and it can arbitrarily delay the delivery of the message between honest parties; however, it cannot drop messages between two honest agents or between the honest user and the honest agent.

**Ideal Functionality.** In our ideal functionality $\mathcal{F}_{\mathsf{CDE}}$ (See Figure 4.6), the user chooses a set of agents $\mathcal{S}_\mathcal{U} = \{\mathsf{A}_{\mathsf{u}_1}, \cdots, \mathsf{A}_{\mathsf{u}_\mathsf{q}}\}$ where $\mathcal{U} = \{u_1, \cdots, u_q\}$ is the set of indices. The user initiates the document transfer using the `inputU` message and forwards the data $m_\mathsf{U}$ and the set $\mathcal{S}_\mathcal{U}$ to $\mathcal{F}_{\mathsf{CDE}}$ with the session id $sid$. $\mathcal{F}_{\mathsf{CDE}}$ receives and stores $\langle \mathsf{U}, \mathcal{S}_\mathcal{U}, m_\mathsf{U}, sid \rangle$. The `intd` message is sent to all the agents indicating that the session has been initiated by the

**Functionality** $\mathcal{F}_{\mathsf{CDE}}$

The functionality interacts with a user $\mathsf{U}$ and $t$ agents with identities $\mathsf{A}_\mathsf{j}$, $\mathsf{j} \in [1, n]$. $\mathsf{U}$ has a message $m_\mathsf{U}$ to be locked and selects the subset of agents $\mathcal{S}_\mathcal{U} = \{\mathsf{A}_{\mathsf{u}_1}, \cdots, \mathsf{A}_{\mathsf{u}_q}\}$ where $\mathcal{U} = \{u_1, \cdots, u_q\}$ is the set of indices of agents chosen. The user and a maximum of $t$ agents can be corrupted by the adversary $\mathcal{A}$.

**Init Session and user input:**

- Upon receiving the message $(\texttt{inputU}, \mathcal{S}_\mathcal{U}, m_\mathsf{U}, sid)$ from user $\mathsf{U}$, record and store $\langle \mathsf{U}, \mathcal{S}_\mathcal{U}, m_\mathsf{U}, sid \rangle$, forward the message $(\texttt{intd}, \mathsf{U}, sid)$ to each agent $\mathsf{A}_\mathsf{j}$ and forward $(\texttt{input}, sid)$ to $\mathcal{A}$.

**Open message:**

- Upon receiving the message $(\texttt{openR}, \mathsf{j}, sid, \mathsf{U})$ from the agent $\mathsf{A}_\mathsf{j}$ and store $\mathsf{j}$ in the set $\mathcal{Q}_{sid}$ and forward $(\texttt{open}, sid)$ to $\mathcal{A}$.

**Release message:**

- Upon receiving the message $(\texttt{releaseR}, \mathsf{j}, sid, \mathsf{U})$ from the agent $\mathsf{A}_\mathsf{j}$, store $\mathsf{j}$ in the set $\mathcal{R}_{sid}$ and forward $(\texttt{release}, sid)$ to $\mathcal{A}$.

**Delivery:**

- Upon receiving the message $(\texttt{deliverU}, sid)$ from $\mathcal{A}$, check if $\langle \mathsf{U}, \mathcal{S}_\mathcal{U}, m_\mathsf{U}, sid \rangle$ is stored, else ignore the message. Send $(\texttt{delivered})$ to $\mathsf{U}$.
- Upon receiving the message $(\texttt{openA}, sid)$ from $\mathcal{A}$, check if $\langle \mathsf{U}, \mathcal{S}_\mathcal{U}, m_\mathsf{U}, sid \rangle$ is stored and $\|\mathcal{Q}_{sid}\| \geq t + 2$, otherwise ignore the message. Send $(\texttt{openoutput}, m_\mathsf{U}, sid)$ to all the agents $\mathsf{A}_k$ for $k \in \mathcal{Q}_{sid}$ and forward the message "$\texttt{key}$" to agents $\mathsf{A}_d$ for $d \in \mathcal{Q}_{sid} \cap \mathcal{U}$.
- Upon receiving the message $(\texttt{releaseA}, sid)$ from $\mathcal{A}$, check if $\langle \mathsf{U}, \mathcal{S}_\mathcal{U}, m_\mathsf{U}, sid \rangle$ is stored and $\|\mathcal{R}_{sid}\| \geq t + 2$, otherwise ignore the message. Send $(\texttt{releaseoutput}, m_\mathsf{U}, sid)$ to all the agents $\mathsf{A}_k$ for $k \in \mathcal{R}_{sid}$.

**Figure 4.6.** Ideal Functionality of CDE

user. $\mathcal{U}$ is the set of indices of all agents to whom detection key is forwarded in the protocol implementation.

Each agent $\mathsf{A}_\mathsf{j}$ can wish to open the user message, they forward the message $(\texttt{openR}, \mathsf{j}, sid, \mathsf{U})$ to indicate they wish to open the message of the user $\mathsf{U}$. The functionality stores the index in the set $\mathcal{Q}_{sid}$. Similarly, the users may wish to release the user message and forward $(\texttt{releaseR}, \mathsf{j}, sid, \mathsf{U})$ to the functionality. Opening the message indicates the

selective opening of the message, using threshold decryption in the protocol implementation. Releasing the message implies reconstructing the secret key using the shares of secret key bits among the agents and decrypting the user message. When the agents forward `openR` and `releaseR` messages from agents, the functionality informs the adversary by forwarding `open` and `release` messages respectively.

When the adversary $\mathcal{A}$ sends the message deliverU to the functionality, it checks if the input from the user is received and informs the user by forwarding the message delivered to the user. When $\mathcal{A}$ forwards the message openA, the functionality checks if at least $t+2$ agents have forwarded an openR message, if yes, it forwards the user message to all such agents. Apart from that it forwards the `key` to those agents in the set $\mathcal{Q}_{sid} \cap \mathcal{U}$. The `key` message models the release of secret key $sk$ to agents in the real world protocol. The watermarked secret key is revealed to the agents to whom the detection key has been forwarded by the user, indicated by the set $\mathcal{U}$. On receiving the `releaseA` message from $\mathcal{A}$, the functionality checks if at least $t+2$ agents forwarded the `releaseR` message, if yes, it forwards the user message to those agents. This models the release of the user message by collaboration from at least $t+2$ agents.

From the different steps of the CDE protocol described in Section 4.3 it can be seen that, when the agents decide to decrypt the secret message, they reconstruct the keys $\mathsf{k}_{\mathrm{i},sk_{\mathrm{i}}}$ and the decrypted message consists the secret key embedded as the watermark. A subset of agents who have the correct detection key will be able to detect the watermarked secret key, this corresponds to receiving the '`key`' message in the open phase of the ideal functionality.

Towards analyzing the security of our protocols under the mixed-behaviour model [35], we offer theorem statements and proof sketches for ideal-real world security paradigm. We employ all our functional blocks in a black-box manner and find that the formal cryptographic analysis to be canonical. We first consider the security of the DROT protocol from Section 4.3.2, which is the key cryptographic construction of CDE. DROT uses multiple instances of UC-Secure OLE protocol for forwarding the key shares to the agents. We present the simulator $\mathcal{S}_{DROT}$ of the DROT functionality in Figure 4.7 before we proceed to provide the proof-sketch for the CDE protocol.

**Theorem 4.5.1.** *Assuming a secure two-party computation of the OLE protocol, the* DROT *protocol (Section 4.3.2) securely implements the ideal functionality $\mathcal{F}_{DROT}$ (in Figure 4.4) under the mixed-behaviour model.*

The simulator $\mathcal{S}_{\mathsf{DROT}}$ in Figure 4.7 interacts with the user $\mathsf{U}$ and $n$ agents $\mathsf{A_j}, \mathsf{j} \in [1, n]$, the adversary $\mathcal{A}$ corrupts a maximum of $t$ parties. The simulator presents an indistinguishable view to the adversary in the real-world ideal-world paradigm. In the DROT protocol, the user inputs pair of of keys $\mathsf{k}_0, \mathsf{k}_1$ where as the agents input shares $[s]$ of a secret key bit $s$ which is $(n, t+1)$ threshold shared among the agents.

The transfer of secret key shares from user to the agents is realized using the secure 2-party computation of oblivious linear function evaluation (OLE) between the user and each of the agents. The simulator $\mathcal{S}_{\mathsf{DROT}}$ invokes $\mathcal{S}_{OLE}$ [97] with corresponding inputs while generating the view for the adversary. The simulator for the OLE protocol $\mathcal{S}_{OLE}$ provided by Ghosh *et al.* [97] is a generalization where the inputs are $t$ element vectors. This can be used in a straightforward manner for $t = 1$. $\mathcal{S}_{DROT}$ uses $\mathcal{S}_{OLE}$ in a black box manner while interacting with corrupt parties, invoking the corresponding side (corrupt sender or corrupt receiver) of it.

While interacting with corrupt $\mathsf{S}$, $\mathcal{S}_{DROT}$ invokes $n$ instances of $\mathcal{S}_{OLE}$ each simulating one agent. $\mathcal{S}_{OLE}$ extracts the sender inputs $\mathsf{k}_0, \mathsf{k}_1$ which are forwarded to the functionality using the message $(\texttt{inputS}, \mathsf{k}_0, \mathsf{k}_1)$.

Up to $t$ receivers can be corrupted by the adversary. While interacting with corrupted receivers, $\mathcal{S}_{DROT}$ invokes $t$ instances of $\mathcal{S}_{OLE}$ which interact with the corrupted receivers. Each instance of $\mathcal{S}_{OLE}$ extracts the receiver input $[\![s]\!]_\mathsf{j}, \mathsf{j} \in [n-t, n]$. The values $[\![s]\!]_\mathsf{j}$ are forwarded to $\mathcal{F}_{DROT}$ using the message $(\texttt{inputR}, sid, [\![s]\!]_\mathsf{j}, \mathsf{j})$. After receiving all the shares, the funcationality computes the value $\mathsf{k}_s$ and forwards the shares $[\![\mathsf{k}_s]\!]_\mathsf{j}$, $t$ of which are received by $\mathcal{S}_{DROT}$. These values are input the $t$ instances of $\mathcal{S}_{OLE}$, which take the values $[\![\mathsf{k}_s]\!]_\mathsf{j}$, set the two inputs while interacting with the corrupted receivers.

When the sender and $t-1$ receivers are corrupted, $\mathcal{S}_{DROT}$ simulates $n-t+1$ receivers to the sender and forwards all the messages between the corrupted sender and the receivers without any modifications. $\mathcal{S}_{DROT}$ invokes $n-t+1$ instances of $\mathcal{S}_{OLE}$ which simulate the

receivers. $\mathcal{S}_{OLE}$ extracts the sender inputs $\mathsf{k}_0, \mathsf{k}_1$ which are forwarded to $\mathcal{F}_{DROT}$ using the message $(\texttt{inputS}, sid, \mathsf{k}_0, \mathsf{k}_1)$.

The indistinguishability of the view follows from the UC-security of $\mathcal{F}_{OLE}$. In the case when the sender and $t-1$ receivers are corrupted, all the messages are forwarded between them without modification. we know that any collusion during or after the document transfer with secret information being revealed to the agents, results in a lower expected pay-off as computed in Section 4.4.1 and Table 4.4. No rational agent deviates from the protocol as participating in the protocol without abort results in a positive non-zero pay-off. Hence the agents neither collude nor deviate from the protocol realizing the functionality securely.

Next, we analyze the security of the CDE protocol assuming that we have access to secure protocols for dkg, bit-decomp, robust bit watermarking and DROT.

**Theorem 4.5.2.** *Let dkg, bit-decomp be secure MPC protocols, $\mathsf{wm}(\cdot)$ is a robust bit watermarking algorithm, and DROT is secure (as in Theorem 4.5.1). The CDE protocol $\pi_{CDE}$ securely realizes the ideal functionality $\mathcal{F}_{CDE}$ under the mixed-behaviour model.*

The system consists of user U and $n$ agents $\mathsf{A_j}, \mathsf{j} \in [1, n]$; at any instance of time a maximum of $t$ parties can be corrupted by the adversary $\mathcal{A}$. The CDE protocol involves distributed key generation, bit-decomposition and DROT protocols, where DKG and bit-decomposition are performed among the agents and DROT protocol is run between the user and the agents. We use these three protocols in a black-box manner with their corresponding simulators. We refer the reader to the works [30] and [163], [164] for the simulators for DKG and bit-decomposition algorithms. We refer to them as $\mathcal{S}_{DKG}$ and $\mathcal{S}_{\mathsf{BitDec}}$.

The simulator $\mathcal{S}_{\mathsf{CDE}}$ generates an indistinguishable view for the adversary in the real-world ideal-world paradigm. It invokes the three simulators $\mathcal{S}_{DKG}$, $\mathcal{S}_{\mathsf{BitDec}}$ and $\mathcal{S}_{\mathsf{DROT}}$ for each of the phases of the protocol run. The DKG protocol [30] is based on polynomial evaluation and Pedersen commitments for generating verifiable secret shares for the parties. Each of the parties generates shares of a random value and locally compute the secret share value from the shares of the qualified set of parties. The bit-decomposition algorithm takes the shares values and generates bit-wise shares of each of the bits of the secret key shared among the users. The simulator $\mathcal{S}_{\mathsf{CDE}}$ invokes $\mathcal{S}_{DKG}$, $\mathcal{S}_{\mathsf{BitDec}}$ during these phases of the

**Simulator $\mathcal{S}_{DROT}$**

The Simulator $\mathcal{S}_{DROT}$ interacts with the sender $\mathsf{S}$ and receivers $\mathsf{A}_\mathsf{j}, \mathsf{j} \in [1, n]$.
The sender $\mathsf{S}$ has two messages $\mathsf{k}_0, \mathsf{k}_1$ and the receivers have shares $[\![s]\!]_\mathsf{j}$ of a random bit $s$.
No more than $t$ parties are corrupted by the adversary $\mathcal{A}$.
$\mathcal{S}_{DROT}$ invokes the simulator $\mathcal{S}_{OLE}$ wherever necessary. $\mathcal{S}_{OLE}$ extracts the inputs of the parties to the OLE protocol while simulating an indistinguishable view in the real world - ideal world paradigm.

**Corrupt Sender**

The simulator simulates $n$ agents to the sender.

- Invoke corrupted sender version of $\mathcal{S}_{OLE}$ while interacting with the sender as each of the agents. The sender input for each of the instances is $(\mathsf{k}_0, \mathsf{k}_1)$. $\mathcal{S}_{OLE}$ extracts the inputs $\mathsf{k}_0, \mathsf{k}_1$.
- Forward the message $(\texttt{inputS}, \mathsf{k}_0, \mathsf{k}_1)$ to the functionality $\mathcal{F}_{DROT}$.

**Corrupt Receivers:**

The simulator simulates the sender to $t$ corrupted receivers. For a bit $s$, the honest receivers forward their shares to the functionality using $(\texttt{inputR}, sid, [\![s]\!]_\mathsf{j}, \mathsf{j})$ for $\mathsf{j} \in [1, n-t-1]$. Assume wlog. that agents $\mathsf{A}_\mathsf{j}, \mathsf{j} \in [n-t, n]$ to be the corrupted receivers.

- For each of the corrupted receivers, invoke corrupted receiver side of $\mathcal{S}_{OLE}$. The input of the receiver $\mathsf{A}_\mathsf{j}$ is $[\![s]\!]_\mathsf{j}$. $\mathcal{S}_{OLE}$ extracts the inputs $[\![s]\!]_\mathsf{j}, \mathsf{j} \in [n-t, n]$.
- Forward the message $(\texttt{inputR}, sid, [\![s]\!]_\mathsf{j}, \mathsf{j})$ for $\mathsf{j} \in [n-t, n]$ to the functionality.
- The functionality forwards the value $(\texttt{output}, [\![\mathsf{k}_s]\!]_\mathsf{j})$ to all the agents, input the value $[\![\mathsf{k}_s]\!]_\mathsf{j}$ to $\mathcal{S}_{OLE}$ instances which using the value, set the corresponding inputs and interact with the corrupted receivers.

**Corrupt Sender and $t-1$ corrupt receivers**

The simulator just forwards messages between the corrupted sender and $t-1$ corrupted receivers.

- Invoke corrupted sender version of $\mathcal{S}_{OLE}$ while interacting with the sender as each of the remaining $n-t+1$ receivers. $\mathcal{S}_{OLE}$ extracts the inputs $\mathsf{k}_0, \mathsf{k}_1$.
- Forward the message $(\texttt{inputS}, \mathsf{k}_0, \mathsf{k}_1)$ to the functionality $\mathcal{F}_{DROT}$.

**Figure 4.7.** Simulator for DOT

protocol simulation. Any deviation from the protocol is detected and the instance is aborted in-case of such deviation. However, before the next two party computation phase commences between the user and the agents, the simulator answers all oracle queries of the user and

stores the query values $q_i$. These queries are used by the user to sample keys to encrypt the different message blocks. The simulator $\mathcal{S}_{\mathsf{CDE}}$ invokes $\lambda$ instances of $\mathcal{S}_{\mathsf{DROT}}$ once for each of the bits of the secret key. In cases with corrupted sender, the simulator computes the keys used for encryption $k_i$ by checking over all the stored queries $q_i$ and inputs those keys.

Since the rational parties have an incentive to obtain the correct key shares for the protocol to proceed, they have an incentive to not deviate from the protocol during the setup. After obtaining the watermarked and encrypted user data and the corresponding key shares through DROT, the agents do not collude at equilibrium as was proved by From Theorem 4.4.1. If the watermark can be removed without destruction of the data, the agents would collude and decrypt the user data. Thus, robust watermarking ensures that collusion is not an equilibrium of the collusion-game. The agents neither deviate nor collude at any point of the protocol there by securely realizing the functionality.

### 4.6 Implementation

We implement the CDE protocol using HoneybadgerMPC [86], SCALE-MAMBA [85], and Charm cryptographic library [165]. Our implementation includes realizing the DROT protocol to transfer encrypted watermarked images and their corresponding key shares. Each run of the protocol involves splitting the data into blocks, watermarking the blocks, and transferring them. The DKG and bit-decomposition protocols are run by the agents as setup before the user enters the system.

**Distributed Key Generation—DKG.** The DKG protocol is realized using HoneybadgerMPC [86], a secret sharing based python MPC framework supporting malicious security. [4] We implement the DKG protocol proposed by Gennaro *et al.* [30], we realize Pedersen verifiable secret sharing (VSS) leveraging the communication layer of HoneybadgerMPC. The DKG is realized by letting each agent perform VSS of random values, sum them up to get the shared private key, and compute the public key from the commitments obtained. The agents generate shares of the 256−bit secret key with the key pair on the curve secp256k1. The DKG protocol has a message complexity of $O(n^3)$ for $n$ parties. When adversary can

[4]↑HoneybadgerMPC supports the robust reconstruction of secret shared values and requires the adversary to control up to $t < n/3$ parties. All test cases we run on honeybadgerMPC satisfy this threshold.

corrupt $t$ agents, the threshold of secret sharing is $t+1$ (at-least $t+2$ agents are needed for reconstruction ), requiring $n > 3t+2$ for safety and liveness of the DKG protocol.

**Bit-Decomposition.** The bit-decomposition protocol is implemented through HoneybadgerMPC framework and the protocol realized is based on the one proposed by Catrina *et al.* [32]. The round complexity of protocol is $O(log(\kappa))$ where $\kappa$ is the number of bits that we want to extract. We implement its variant with constant round complexity by using an alternative sub-protocol for bit-wise addition [163]. Moreover, we use the prefix multiplication protocol introduced in [166] to replace the prefix AND sub-protocol in [163]. Finally, we achieve a bit decomposition protocol with $O(\kappa^2)$ communication complexity. The protocol requires $O(\kappa^2)$ beaver triples and $O(\kappa^2)$ random shares as the offline cost. The field size for bit-decomposition is the same as the curve `sect571k1` to support decomposing 256-bit values with a sufficiently large security parameter. With a large number of multiplications involved, this is the most expensive operation in our protocol.

**Two-party computations.** Two party computation is required for the DROT protocol (Refer Section 4.3.2) and we implement it using SCALE-MAMBA [85] since HoneybadgerMPC does not support a full threshold protocol. In SCALE-MAMBA, fully homomorphic encryption is used in the offline phase and SPDZ-style secret sharing [85] is leveraged in the online phase. The whole two-party computation involves one multiplication and one reconstruction. The computation is performed by the user with each agent so the total communication complexity is $O(n)$ for $n$ agents.

**Watermarking.** The user splits the data into blocks and generates two versions of each block by watermarking with bits $0, 1$. For a 256-bit secret key of the agents, the user divides his data into 256 blocks. While the data can be of any form including multi-media and document data, we use bit map images for the prototype. For image watermarking we use the combined DWT-DCT watermarking algorithm proposed by Ali Al-Haj [167] and the implementation based on [168]. The DWT-DCT algorithm works by altering the wavelets of the Discrete Wavelet Transform (DWT) sub-bands and applies Discrete Cosine Transform (DCT) on few sub-bands.

**Table 4.5.** Time taken and data transferred for DKG and bit-decomposition phases for number of parties $n = 5, 8$. DROT involves two-party computation with $n = 2$

| $n$ | Phase | Time | Data |
|---|---|---|---|
| 5 | DKG | 2.155sec | 3.6 KB |
| | BitDec | 34.1sec | 92MB |
| 8 | DKG | 2.165sec | 6.3 KB |
| | BitDec | 34.2sec | 108MB |
| 2 | DROT | 124.6msec | - |

**Table 4.6.** Time (mean ± standard deviation) taken in seconds for different steps of watermarking for different images

| Image | Size(KB) | Splitting | Watermarking | Extraction |
|---|---|---|---|---|
| Cameraman.bmp | 66 | 0.00710 ± 1.2e-05 | 0.0542 ± 2.2e-04 | 0.023 ± 1.6e-04 |
| Bridge.bmp | 263 | 0.04176 ± 4.8e-08 | 0.1241 ± 6.7e-04 | 0.046 ± 3.2e-04 |
| Sailboat.bmp | 769 | 0.0713 ± 2.8 e-06 | 0.1827 ± 17.8e-04 | 0.065 ± 7.6e-04 |
| Airplane.bmp | 769 | 0.0785 ± 1.09e-06 | 0.1811 ± 3.69e-04 | 0.062 ± 5.4e-04 |

### 4.6.1 Experimental results

To evaluate the performance of our building blocks, we deploy our prototype on AWS clusters and run the protocols on the c5.2xlarge instances (8 cores and 16GB RAM). The instances are allocated in 5 regions across 4 continents. The benchmark data has been averaged over 10 runs of the protocol each for $n = 5, 8$.

Table 4.5 presents the times taken and the data transferred by each node for the DKG and the bit-decomposition (BitDec) phases. The DKG and the bit-decomposition take around 40 seconds, which would be the time for setup of the protocol before the users interact. We observe that the running times for $n = 5, 8$ are almost the same. The reason is that when $n$ increases, the number of instances in each region also increases, thus parties can receive shares from geometrically closer parties, and this advantage cancels the workload caused by a larger threshold. DROT realized through SCALE-MAMBA, involves two-party computation ($n = 2$) between the user and an agent. The interaction takes around 120 milliseconds, which is dominated by network latency (the local computation takes only around 0.1 milliseconds).

Table 4.6 shows the times taken for image watermarking including splitting the image, watermarking each block, and extracting the watermark from the whole reconstructed image after the revelation. The mean and variance have been reported when each step is run 100 times. The user performs the watermarking offline before interaction with the agents.

# 5. UNDERSTANDING USERS' MENTAL MODEL IN ADOPTING MULTI-DEVICE WALLETS

## 5.1 Classifying wallets

All cryptocurrency wallets today use paired secret keys and public keys [169], [170], where a wallet's address is derived from its public key. However, storing and accessing a secret key is a non-trivial problem and varies from one class of wallets to another. Here, first, we briefly summarize the existing classification of wallets and identify that they often ignore the underlying security models. Then we present a new classification to address this issue.

**Existing classifications of the cryptocurrency wallets.** Several classes [171]–[173] of cryptocurrency wallets exist today depending on different dimensions—*hot and cold* wallets, *custodial and non-custodial* wallets [44], [172] etc. Hot wallets are connected to the internet while cold wallets are not. To perform a transaction with wallet's firmhe wallet's firmcold wallet, the secret key needs to be taken from the offline storage like paper or QR code and employed. In another classification, a *non-custodial wallet* refers to a simple model of wallets where the secret key resides at user device. These wallets are notorious for loss or misplacement of keys and subsequent loss of funds—∼20% of all mined bitcoin lost this way [174]. In contrast, *custodial wallets* refer to ones where the secret key is not of the user (device) but at the firm which is offering the wallet. Every time the user makes a transaction, they authenticate to the firm which performs the transaction on their behalf. While this safeguards against the loss of key at the user, it forces the user to trust the firm operating the wallet. A popular way to achieve a custodial wallet mechanism is to place the keys at the cryptocurrency exchanges that offer wallets and transact on behalf of the users. This approach is susceptible to attacks by hackers on exchanges and affects very large user bases [52]–[54], [175]–[178]. Thus, it is quite evident that storing the keys at a single location is a security risk, irrespective of the user (client-side) or the firm (server-side).

A relatively new type of wallet solves these issues—it distributes the secret keys into multiple shares [58], [179] and places them at different locations. Depending on the specific

application, these locations can be a combination of different firms/servers, and devices owned by a single or multiple users.

**Need for new security-focused classification.** Note that the existing classifications of wallets focus far more on how the wallet is *used* rather than the underlying nuanced security models (e.g., how the security of the keys is guaranteed). While Hot-Cold classification focuses on wallets' connection to the internet, Custodial-Non custodial notion classifies the wallets as whether the key is only with the user or the remote server. However, in all presented cases, irrespective of if the key is placed at the user or the server, compromising the single key location compromises the funds; the multi-device wallets mitigate this security risk [49]. Understanding this risk by explicitly stating the security model is essential. If the users appreciate the underlying security model, they can make informed choices about their wallets. Hence, to investigate the user risk perception and mental model regarding the security of different wallets which is invariably related to the key location, we classify all the wallets into *single-device* and *multi-device* wallets with the key being stored at a single location or distributed among multiple locations.

### 5.2 Different single device wallets

- *Brain wallet*: In this, users choose to remember the passphrase or key associated with the wallet. This wallet is a single device wallet as the key is in a single location, the brain. If the user forgets the secret information, they can not access the funds.
- *Paper wallet*: The secret key of the wallet is placed on paper, typically as a QR code etc.
- *Desktop/Mobile wallet*: The wallet and the corresponding secret key are placed on the desktop or the mobile device of the user. The user can access the wallet only from that particular device. Eg: Electrum
- *Exchange wallet*: The secret key is placed at the exchange hosting the wallet. The exchange performs the transactions on behalf of the user. Eg: Coinbase.com, Binance
- *Web wallet*: The secret key is stored at the firm offering the wallet. This wallet is accessed through the web and hence is not device dependant.

- *Hardware wallet*: The secret key is stored on a particular hardware token. The client needs to plugin the hardware token every time a transaction is made. Eg: Trezor, Ledger Nano

### 5.2.1 Single- and Multi-device wallets

Single-device wallets store the keys at a single location; either on a client device or a remote server hosting the data of the firm offering the wallet. If the user loses access to the device, they can not access any funds associated with the account. The different well-known single-device wallet types, including paper, desktop/mobile, hardware, and exchange wallets, are presented in Section 5.2. These wallets provide control of the key to a single entity – the user or the wallet firm. In a multi-device wallet, the secret information is placed on multiple locations/devices; any subset of a particular size or higher of the devices should respond to authorize the transaction. These devices are held by one or more entities, including users and remote servers of the firm.

**Single-device and multi-device wallets - Security.** In a single-device wallet, since the key is in a single location, it introduces a single point of failure for the loss of the key. Loss of keys by the users and exchange hacks [52]–[54], [174]–[178] show that the single-device wallets are highly vulnerable to loss or compromise from an adversary. In an multi-device wallets, the key is distributed among multiple locations, loss or compromise of a single device does not lead to loss of key; the attackers need to compromise multiple servers simultaneously to compromise the keys. Hence they are more resistant to stealing keys by adversaries and are less prone to key loss.

Recently, Eyal [49] has shown that for a wallet, an increase in the number of associated heterogeneous keys improves security; the probability of users losing access and adversaries gaining access is lower for multi-device (multi-key) wallets than single key scenarios. Hence, multi-device wallets are more *secure* than their single-device counterparts. Several different approaches [49], [180]–[182] mitigating the security risks of the single-device wallets also indicate that multi-device wallets have been invariably proposed as schemes to achieve better security than single-device wallets. In this study, we investigate the users' mental model

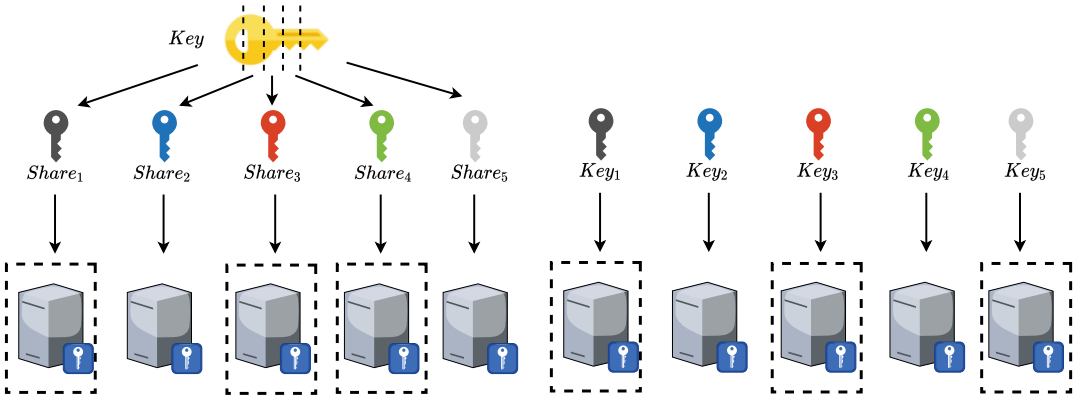
regarding the security offered by the multi-device wallets and the gap between the proposed and perceived security.

**Single-device and multi-device wallets - Trust and Usability.** The trust and usability aspects of single-device and multi-device wallets are more nuanced. For single-device wallets, since the key is placed in a single location, the users need to trust the single entity or location not to get compromised for the safety of their funds. In contrast, for multi-device wallets, users need not trust a single entity like in exchange wallets since the secret information is distributed. Naturally, since the key is distributed among multiple entities, multi-device wallets achieve higher replication of the keys.

For an multi-device wallet, when part of the key is placed on the client device, key-recovery is straightforward in case of device loss since the other parties can generate new shares. Also, with a good choice of threshold structure, the keys can be made highly available [182] similar to the single-device scenario. It should be noted that depending on the setting multi-device wallets can also provide complete control of the key to the user like the single-device wallets. For example, in a scenario when the key is divided into two shares and one of the shares is placed on the client device, the transaction does not go through without client authorization irrespective of how the second share is shared among multiple servers.

Though the interface of many multi-device wallets (Eg: ZenGo, Torus) is similar to single-device wallets for making transactions, multi-device wallets typically have a higher setup time. The usability issues and misconceptions of users regarding wallets [43] like participants' confusion regarding transaction and mining fees, cancellation of transactions, lack of blockchain transparency regarding the transaction state is likely to be common between both single-device and multi-device wallets since they are not dependent on the location of key or authorization.

While the focus of this work is on the security model of different wallets and the users' perception of them, we uncover interesting mental models regarding usability aspects. The perceptions of usability directly affect the different preferred settings and thresholds for the multi-device wallets (discussed in Section 5.5).




(a) $(5, 3)$-Threshold wallet (b) $(5, 3)$-Multisig wallet

**Figure 5.1.** Multi-device wallets. (a) Threshold Wallet: Key-shares of a single key are generated and stored in different locations. (b) Multisig Wallet: Multiple (different) keys are stored on different devices (can be different client devices). A subset of shares or keys – threshold $T$ or more – are required to sign the transaction in each case.

### 5.2.2 Subclasses of Multi-device wallets

We further classify the multi-device wallets into two types *Multisig wallets* and *Threshold wallets*. In a multisig (multi-signature) wallet [183]–[185], $N$ different keys are generated and placed on $N$ devices such that signatures [186]–[188] from at least $T$ devices are needed to authorize the transaction. These keys may be placed on devices of different users or a single user. For example, multiple keys are given to different people in a board of a firm such that at least a subset of them need to provide the signature for the transaction or payment to go through. The set of signatures authorizing the transaction, reveals the access structure $(N, T)$ of the distribution of the keys used. Both multisig wallet and threshold wallet (depicted in Figure 5.1) employ an access structure where the secret information is distributed among $N$ locations such that any $T$ or more locations need to respond to authorize the transaction. We call it the $(N, T)$ access structure. In a threshold wallet[189], [190], a single secret key is *secret-shared* [191], [192] among $N$ devices out of which $T$ or more devices provide a partial signature. The partial signatures are collected and aggregated into a single (threshold) signature [188], [193] to authorize the transaction.

The signature generated as a threshold signature does not reveal [193], [194] the underlying access structure among the clients or which parties signed the transaction. A threshold signature is similar to a single regular signature, unlike multisig, which is a concatenation of multiple signatures, so it offers better storage efficiency. However, threshold signatures are dependent on the exact cryptographic scheme and are not widely available for all signature schemes. This is not an issue with multisig schemes as they can be realized using any signature scheme, but they impose scripting [195] requirements.

## 5.3 Related Work

### 5.3.1 Usability and security of crypto-wallets

Many recent studies [43]–[46], [196], [197] have focused on usability issues and challenges of cryptocurrency systems. Few studies have also explored issues of trust in blockchain systems, the trust challenges/risks and ways to mitigate them [198]. Recently, Mai *et al.* [199] brought out the general misconceptions of users in using cryptocurrency systems regarding keys, anonymity, and fees. They investigate misconceptions about the generation of cryptographic keys, which may lead to their mishandling and loss of funds. Voskobojnikov *et al.* [196] study the risk perceptions of both users and informed non-users of cryptocurrencies. They discuss several perceived risks, including loss of keys by the participants and risk mitigation strategies for different cryptocurrencies. They [196] observed that some non-users (non-crypto wallet users) are concerned that governments can trace the transactions back to them (loss of pseudonymity/anonymity). In contrast, we consider only participants who have used crypto-wallets. We aim to understand their preferences, e.g., under different government policy and capability scenarios where the governments can access or block the secret keys. The considered multi-device wallet settings include different settings hindering such overreach.

**Usability.** Cryptocurrency users face challenges regarding usability and understanding of the security implications of features of the wallets. Blockchains and cryptocurrencies also suffer from entry barriers and the perception of usability between users and non-users [45], [46], [196]. Voskobojnikov et al. [43] study the user experience of wallets by analyzing the

($> 45K$) ratings of famous cryptocurrency wallet applications. They reveal that users have several misconceptions regarding the features and interface, including how mining and transaction fees are collected, leading to grave errors in handling the secret keys and currency transfers. They [43] briefly observed that some of the users preferred access to the secret keys (like in iOS Apple devices) compared to custodial wallet settings. Furthermore, some of their participants were also concerned about losing the device and the secret keys. In contrast, we explore how participants wish to overcome such concerns if they wish to shift to multi-device wallets. For the multi-device wallets, we explore and understand user preferences among the varied settings offered by multi-device wallets differing in the levels of control, availability, and security of secret keys. In fact, our user study is the first of its kind for multi-party computation (MPC) or threshold cryptography, although MPC is almost as old as public-key cryptography itself.

Krombholz *et al.* [47] performed a large-scale survey and evaluation of different security practices of Bitcoin users and brought out the perceptions and flaws in the usage of bitcoin wallets. Halpin *et al.* [197] studied the usability problems in using crypto-wallets like ZCash while achieving privacy through Tor and VPNs. They identify that most users find it difficult to set up wallets and integrate with anonymization tools like Tor. Frohlich *et al.* [44] study the usability of wallets and security practices by conducting semi-structured interviews of participants and propose a model to map the users by their exposure to the internet and key management. More recently, Abramova *et al.* [200] classified all the crypto-wallet users into three groups cypherpunks, hodlers, and rookies. They measured multiple factors, including perceived notions of self-efficacy, vulnerability, concern, etc, for clustering and observed specific differences in the preferences of different types of wallets, measures taken to secure their wallets, etc.

Building on this line of research, along with security issues, we investigated and uncovered different perceived usability aspects and how they affect the choice of threshold settings in multi-device wallets. For example, some participants preferred lower thresholds in multi-device wallets for lower transaction (submission) delay.

**Security and Privacy issues.** Frolich *et al.* [201] presented a systematic overview of different threats faced by cryptocurrency users—accidental, privacy, physical, financial fraud, social and technical threats. Among the physical threats they observe loss of cryptocurrency as a potential threat. As a possible counter measure they suggested backup mnemonics—divided and stored as separate parts on multiple devices. Furthermore, recently, Ghesmati *et al.* [202] studied the privacy perceptions of (12 cryptocurrency users and 58 non cryptocurrency users) about blockchains. They evaluated user-perceptions about anonymity, privacy and users' mitigation measures. They found that privacy concerns for few of 12 users were about exchanges having access to the secret information of the wallets and also exchange hacks. The authors observed that a majority of their participants preferred to use privacy coins with additional tools like CoinJoin, CoinSwap [203]–[206]. Our work builds on and is complementary to these prior works—instead of threats and privacy enhancing systems, our work focuses on (mis) conceptions about key-management as well as user preferences regarding key management for better perceived security of crypto-assets.

### 5.3.2 Key management in wallets

Passwords are still a popular form of authentication [207] and are even used by many cryptocurrency wallets [39], [41], [208]–[210]. However, the underlying authentication mechanism for cryptosystems is through public-key cryptography using secret-key, public-key pairs. Usability issues of public-key cryptography in encrypted e-mail have been studied [211]–[213] to report that key management by the end-users is indeed a complex task. To uncover usability issues in bitcoin key management, Eskandari *et al.* [48] conducted a cognitive walk-through of bitcoin applications, uncover shortcomings, and provided a framework to evaluate such key management systems.

**Vulnerabilities in wallets.** Single devices wallets are vulnerable to several attacks; Vasek *et al.* [50] study how brain wallets are prone to offline password guessing attacks. They show that most brain wallets are vulnerable and can be drained within a day of creation. Arapinis *et al.* [51] study the vulnerabilities of the hardware wallets by modeling their

security in the Universal Composability framework. They analyze a few well-known hardware wallets in their framework and show that they are vulnerable to payment, address generation, and chain attacks. Bui *et al.* [214] study how computer/desktop wallet applications are vulnerable; even without privileges, the attacker can impersonate the endpoints of remote procedure calls (RPC) and transfer funds. While multi-device wallets mitigate the risks of single-device wallets by distributing secret information among many devices, they still can be vulnerable to attacks. Aumasson and Shlomovits [190] show ways to attack the implementations of schemes like threshold-ECDSA [215], [216]; they also suggest ways to mitigate them.

Several works [181], [217], [218] studied the vulnerabilities in single device wallets and proposed various ways to mitigate them. Instead of storing the secret key in the memory, Dai *et al.* [217] suggest storing the seed of the secret key in a trusted part of the hardware such that no adversary can access it. Barber *et al.* [180] propose a super wallet - sub wallet mechanism where the currency is placed in the super wallet and transferred to sub wallets in smaller quantities as and when required; Rezaeighaleh and Zou [218] propose a deterministic sub wallet key generation from the super wallet seed. Marcedone *et al.* [181] proposed a two-factor signature generation mechanism in hardware wallets to be secure against malicious hardware vendors. He *et al.* [182] propose a distributed key management mechanism for better availability of keys in a multi-device wallet setting where the key is distributed among multiple servers; the proposed scheme provides high availability of the keys for the users. It is evident from the different approaches [49], [180]–[182] that multi-device wallets have been invariably proposed as schemes to mitigate the security risks of single-device wallets. This work contributes to understanding how the different users perceive the security of multi-device wallets and if there is a gap between offered and perceived security, thereby affecting their adoption.

## 5.4 Methodology

In this section, we discuss our survey-based study design to understand the wallets' usage and user preferences.

### 5.4.1 Survey instrument

Our survey instrument [1] had two parts. We asked questions regarding users' experiences with different crypto-wallets in part I. In part II, we probed users' preferences for two broad classes of wallets—single-device and multi-device wallets after grounding their understanding with videos discussing them. In our mixed-methods study, similar to Owens *et al.* [219], quantitative approaches uncovered user behavior, and qualitative methods uncovered mental models.

**Part I: Usage characteristics, experiences with current wallets, and factors responsible for choosing a wallet (RQ1).** We start part 1 of our study with a survey by asking which wallets are used by the participant and what factors impacted this choice. Specifically, we probed the impact of factors like wallets' interfaces, security guarantees, operation in multiple currencies, ease of recovery, as well as the relative importance of crowd-sourced ratings or reviews from famous personalities on the choice of a particular wallet. Next, to uncover experiences with their current wallets, we asked if our participants ever lost a key or password, resulting in the loss of crypto funds and their most significant security concern regarding crypto-wallets. We also adopted two sets of questions from earlier work to understand our participant attitudes better. These questions measured perceived vulnerability and perceived self-efficacy regarding safeguarding the funds and secret keys in crypto-wallet settings [200]. Finally, we asked the participant how familiar they were with each wallet– paper, exchange, desktop/mobile, threshold, and multisig wallets. These questions helped us estimate the user-familiarity levels with different wallets presented in the next part of our study.

**Part II: Users' preference for multi-device wallets and their default settings (RQ2, RQ3).** In the second part, we first educated the participants about different wallets using two short videos, each approximately 2 minutes long. The first video[2] discussed different single-device wallets and their pros and cons. The second video[3] showed how multi-device wallets mitigate the single-device wallets problems and discussed the two

[1]↑Can be found at https://eprint.iacr.org/2022/075
[2]↑Can be found at https://www.youtube.com/watch?v=rH1bcbeoPew
[3]↑Can be found at https://www.youtube.com/watch?v=LVa00yiOBjA

multi-device types —threshold and multisig wallets. Informing the participant using the videos helps us bring all participants to a similar understanding of wallets and helps us analyze their responses more confidently. To assess whether the participants have indeed watched and understood the content, we ask three knowledge-based questions (with justifications for their answers) after each video.

We first explain why multi-device wallets can be more secure (see Section 5.2.1) and survey if the participants are willing to shift to them. After inquiring the specific reasons for shifting (or not), we study their preferred different settings.

After showing the videos, first, we asked users' preference of the location for storing the key of an exchange wallet. This question helps us understand if the users trust the exchange and any single location among different client devices and remote servers. We further asked the participants about the vulnerability of various key storage locations of single-device wallets. Next, we inquired if the participants were willing to shift to multi-device wallet if the wallet developer provided it. We also asked which one they prefer between threshold and multisig wallets and why.

To understand the participants' preferred settings for the multi-device wallets, we asked them to choose among three different settings with a varied number of servers and threshold values. In this part, we essentially uncover participants' preferences regarding the reputation of the server hosts and the total number of servers. Furthermore, we explored the participants' preference regarding storing the secret keys for single-device wallets in the face of different attack scenarios and preference regarding the distribution of the shared keys among different devices for multi-device wallets.

Finally, we asked questions to investigate the participants' preferences regarding the key locations. Specifically, we showed users scenarios regarding different threat models (e.g., governments viewing and blocking access to the information hosted on servers in their jurisdiction). Then we asked where the participants preferred to store the key (share) *by default* among the options provided in the single device and multi-device wallet settings for these different threat models. These questions provide us with information regarding the desired settings of wallets under various threat models.

Essentially, we first educated the user regarding the advantages of multi-device wallets and checked if they were willing to shift to them. If they are not ready to shift even at the cost of security, we analyzed the reasons. We then studied the preferred settings for the multi-device wallets, including server setup under various government policies.

### 5.4.2 Pilot Studies

Before final deployment, we conducted two pilot studies for our survey. In the first, we piloted the survey using in-person interviews with six participants to test the comprehensibility of the questions and measure the average completion time.

Initially, the survey videos were shown to the participants consecutively, followed by four knowledge-check questions. However, during the first pilot, participants demonstrated a loss of attention, evident from the incorrect answers to our follow-up knowledge-check questions. However, when asked to explain the wrong answers, participants reevaluated and desired to change their responses, hinting at a cognitive overload. We divided the videos into two sections to address this problem and ask questions about each video separately. Responses from this first pilot also prompted us to simplify some questions which asked to rank provided options—we ended up converting them to equivalent Likert scale questions.

After incorporating the changes, we conducted a second pilot study using a crowdsourcing platform named Prolific.co, which is regularly used for academic advertising surveys. We recruited 20 (pre-screened) participants for further feedback. We asked additional follow-up questions to check the ambiguity of questions and answers in this pilot. 90% of the participants found no ambiguity in the survey. Additionally, we asked to explain the answers to knowledge-check questions to nudge participants to be attentive to our educational videos on wallets. We also increased the knowledge-check questions to three per video, totaling six instead of the earlier four for more stringent checking of the acquired knowledge.

### 5.4.3 Recruitment

Our online survey is scalable to a large number of participants. Consequently, we uncovered interesting user behaviors and attitude patterns using statistical analysis. However, one

key challenge of our recruitment was to target crypto-wallet users and enthusiasts. To that end, following the approach of Abramova et al., [200], we recruited participants from two sources—(i) The crowdsourcing platform Prolific.co, (ii) the social media platform Twitter to reach the broader cryptocurrency community. We recruited participants who have been using single-device and multi-device wallets; we do not restrict this study to only multi-device wallet users since we study the preferences of both single-device and multi-device wallet users and if single-device wallet users are willing to migrate to multi-device wallets. Restricting to only multi-device wallet users would not have been sufficient for our purpose.

**Recruitment from Prolific**: For Prolific, we chose participants both from the US and UK [4]. We ensured that they had not taken our pilot studies. We selected them using a screening survey conducted before the entire survey. This screening survey consisted of seven questions about the wallets they were using, for how long, and how frequently they used those wallets. To avoid irrelevant user responses, we made the question about their current wallet a text entry question. We removed all the participants who left the text field blank or entered an invalid wallet name. We also asked screening survey participants whether they were interested in a future longer survey.

We deployed the final survey in multiple batches of $30 - 50$ participants on various days and times over one week. We did this for the distribution to counter any anomalous time dependencies due to the effect of events occurring at a specific time [221]. The median time of completion of the survey was 21 minutes 52 seconds, and the compensation was 4$ for each participant (indicating an hourly wage of 10.88$, comparable to prior studies [200]). Furthermore, participant feedback from the pilot study on prolific showed that 95% of the participants were satisfied with the payment. We used additional stringent quality control criteria (Section 5.4.4) to ensure the quality of responses in our final dataset.

**Ethical Considerations.** For all participants, before the survey began, we informed the participants of the purpose of the study, its estimated duration, and the compensation. We further assured the participants that we would not collect any personally identifiable information (PII). Participants could abort and return the survey at any time during the study.

[4]↑Over 65% of the participants on Prolific are from the US and UK [220] who speak English and more than 18 years of age

Any identifying information like email ids, Twitter handles, etc., related to the participant is removed from the collected responses to preserve the anonymity of the participants. Our study was examined and approved by the lead author's Institutional Review Board (IRB).

### 5.4.4 Quality Control

To ensure the quality of responses, we randomly added an attention check question asking them to choose the current month of the year. Apart from that, we consider responses only from those participants who have answered our knowledge-based (Yes/No) questions satisfactorily (to check if they watched our videos). We consider only those participants who answered at least two out of three correctly in each subset. Furthermore, if a participant answered two or more questions wrong, one author manually checked the corresponding explanation to see if the participant was knowledgeable. For example, in one of the questions asked about the loss of funds upon an Exchange wallet compromise, participant $P25$ disagreed and responded —"*compromise of the server holding the keys does not necessarily mean my money is lost*". This participant has understood the question and has an idea about the correct answer but over-thought the questions. Correspondingly, they chose the wrong option; we included all such participants in our final study even when they answered more than two questions wrong in each set. There were 22 such responses. When watched at regular speed, the total length of videos was 4 minutes 35 seconds; hence we also exclude participants who finished the survey in less than 15 minutes, including watching the study. Since that would have implied they completed both parts of the study in around 10 minutes or less, signifying the poor quality of responses (also manually verified via checking qualitative responses).

**Knowledge-test after the videos.** Few participants were already familiar with multi-device wallets (evident from "familiarity"-related responses); However, to bring all participants to a similar level of understanding, we developed the two educational videos on different wallets (Section 4.1).

We were careful not to offer any views on the different settings of the different types of wallets and only *introduced* the single-device and multi-device wallet models. We were

mindful of any influence that the videos might exert on the users' choices. Note that there were significant number of participants who preferred single-device wallets (see Section 5.5.2) over multi-device wallets after watching the videos. Our approach, inspired by Ghorbani *et al.* [222], quantized the utility of the videos with a total of six knowledge-test questions (and accompanying free-text explanations). We included participants who answered most of the questions correctly, and an additional 22 participants who were considered knowledgeable from the manual evaluation (by authors) of free-text explanations.

### 5.4.5 Participant Demographics

A total of 334 participants responded to the survey on Prolific. We discarded the responses that did not meet the validity criterion and passed our quality control checks (Section 5.4.4). Finally, there were 210 valid responses from Prolific and 45 valid responses ( total 250) from the Twitter platform.

In total, 72.15% of the participants identified themselves as male and 27.45% as female, while one participant preferred not to answer, indicating a male bias in our sample. Among the different age groups, the $25-34$ age group dominated the total population with more than half of the total at 52.15% followed by the $18-24$ and $35-44$ age groups at 21.56% and 20%. Thus, our study has a larger younger population than older ($>35$). The participants in our survey are also more educated than the general US population [223], with 73.32% of the participants having a Bachelor's degree or higher. While one expects the participants from crowdsourcing platforms like Prolific to be tech-savvy [224], more than half of the participants (50.98%) of our participants reported that they do not have any experience in the information technology (IT) field.

Importantly, our participants are active users of different crypto-wallets, where they invest 29.56% of their savings on average across all participants. We provide our participants' crypto-wallet usage pattern in Figure 5.3. They follow different social media and reputed personalities for ratings and reviews in choosing their wallets, as shown in Figure 5.2a and Figure 5.6. Overall, a majority of our participants are young, well-educated and have invested in cryptocurrencies.

### 5.4.6 Analysis Method

**Coding free text answers.** We coded all the free text answers and explanations for questions from our survey to segregate and uncover different perceptions of the participants. Two researchers have independently coded all the free-text responses using a common codebook. Across the various questions, the inter-rater agreement – Cohen's $\kappa$ [225] was in the range $0.7 - 1$, indicating substantial agreement. The coders met and resolved the disagreements to arrive at the final codes.

**Statistical Analysis.** We used statistical hypothesis testing to uncover different correlations and identify the significant factors affecting the various preferences of the participants. We used Chi-Squared ($\chi^2$) test [226], [227] for the different responses to all the questions to uncover correlations between groups of participants and their preferences. We also used the Mann-Whitney U test [228] between participant groups to compare their characteristics. For our tests on the multi-answer questions, we treat each option as an independent question/answer. Our results for the $\chi^2$ tests have been presented in Table 5.1 and for Mann-Whitney U test are presented in Table 5.2. We used Mann-Whitney U (MWU) on Likert-scale responses and Chi-squared ($\chi^2$) test for checking correlation between user-groups and categorical preferences (Section 4.6). For all the tests, the significance level $\alpha$ was 0.05, which was further adjusted using Bonferroni multiple-testing correction[229] .

### 5.4.7 Limitations

We conducted the study to uncover factors affecting the users' preferences in choosing their wallets. While we tried to cover the aspects comprehensively, one should interpret our study in the context of limitations like all the previous studies. We collected 255 responses from platforms Prolific and Twitter. The responses from Prolific were limited to UK and US. Cryptocurrencies have proliferated and are used by various people varying in understanding, knowledge, and preferences. Restricting to two countries and online platform Twitter may be restricted in terms of the range of preferences uncovered, including any geographical or cultural influences on the choices made. However, we obtain interesting insights into the

mindset of the crypto-users in choosing the single-device and multi-device wallets. We bring out interesting observations regarding the participants' desire for control over their funds even in conditions of possible compromise.

Our survey and the two videos included in it have been in English. We required the participants to speak English which might have excluded any non-English speaking population and resulted in possible language and cultural bias. It would indeed be an interesting future work to identify and understand these biases and their effect on the choices made in choosing wallets. However, we believe our study covers a significant range of crypto users with varied experiences obtained while dealing with cryptocurrencies.

## 5.5 Results

### 5.5.1 Current usage-based groups and factors affecting users' choice of wallets - RQ1

We begin by categorizing our participants into two distinct usage-based groups: *Newbie* and *Non-newbie*. We report the usage and preferences of each group and compare them. We also analyzed the security-related preferences of these groups.

**Two different user groups exist with different familiarity and usage.** We first divided the users into usage-based groups to capture various behaviors and understand their preferences.

Recall that Abramova *et al.* [200] categorize participants into three categories using multiple factors, of which perceived vulnerability and self-efficacy are significant. However, a similar investigation of the perceived vulnerability and self-efficacy using the same set of questions has not resulted in any statistically significant clustering. Interestingly, the self-identified categories correlated well with the other independent survey responses regarding expertise and preferences. Specifically, we asked the participants to identify themselves among three types – (i) I use them solely for the interest in technology, (ii) I use them primarily as an avenue for trade, buying, and selling cryptocurrencies (iii) I am a newbie, started using them for fear of missing out the crypto boom.

The pairwise tests between responses from these three categories (for familiarity with wallets and usages) depicted a lack of statistically significant difference between the first and second categories (see Tables 5.4 and 5.5 for pairwise $p$-values for 3 group classification). Hence we group all the participants choosing the first two options as *Non-newbies* and the participants self-reporting as newbies under the *Newbies* group. These two groups significantly correlated with their responses to other questions, as indicated by the low $p$-values (see Table 5.1). We present the mean values for different responses among the two groups in Tables 5.1 and 5.2. We categorized the users based on self-identification, however, this division was supported by statistical tests from other independent survey responses—usage, investment, and familiarity.

Specifically, responses to the questions which significantly correlated with these two groups are in Tables 5.1 and 5.2 (p-values are with Bonferroni correction). The key differences occurred in the duration of usage of the crypto-wallets, % of savings invested in crypto assets, background knowledge in computer science/information technology (IT), and the purpose of use (trading). The other factors that differentiate the two groups are familiarity with different wallet types.

Our survey also reveals that participants consider ratings to be important in choosing wallets and the majority of current users are recent adopters (65% of the participants started using them only in the last two years); Loss of keys either through server compromise, by the user or compromise by the server and lack of recovery mechanisms are among the prominent security concerns of the participants. Social media is a source of knowledge for the participants regarding crypto-wallets where Twitter, YouTube and Reddit featuring in the prominent ones.

**Most users use single-device wallets.** Most of the participants use single device wallets, including hardware wallets like Trezor (presented in Figure 5.13). Coinbase and Binance seem to be popular among the Newbie and Non-newbie groups. 60% of all the participants use Coinbase, whereas 37.2% use Binance. The participants had a choice to enter up to 3 unlisted wallets in the 'Other' fields. The wallets listed by participants varied widely, including TrustWallet, Ziglu, Atomic, Dharma, and ZenGo.

**Table 5.1.** Chi-squared test results for different questions including demographics for Newbies and Non-newbies. The number of samples is 255. The table only shows the variables that have significant $p$-values. df is degrees of freedom.

| Variable | $\chi^2$ | df | p-value |
|---|---|---|---|
| Number of years | 39.1284 | 3 | 1.63e-08*** |
| % of savings invested | 27.1900 | 4 | 1.81e-05*** |
| Background in IT | 24.3844 | 2 | 5.06e-06*** |
| Usage - Trading | 17.10456 | 1 | 3.53e-05*** |

Significance codes: ***$p<$ 0.001, **$p<$0.01, *$p <$ 0.05

**Table 5.2.** $U, p$ values for the Mann-Whitney U test. In the table we only present the variables that have significant $p$-values. The mean values for Newbie and Non-newbie groups are also presented.

| Variable | U | p-value | $\mu$-Newbie | $\mu$-NonNewbie |
|---|---|---|---|---|
| **Familiarity- Wallet** | | | | |
| Paper | 7575 | 0.0001*** | 2.15 | 2.98 |
| Exchange | 8575 | 2.48e-09*** | 2.44 | 3.71 |
| Desktop/Mobile | 8161 | 2.61e-07*** | 2.93 | 3.90 |
| Hardware | 8962 | 1.78e-11*** | 1.87 | 3.24 |
| Multisig | 8121 | 2.73e-07*** | 1.44 | 2.38 |
| Threshold | 8033 | 4.46e-07*** | 1.34 | 2.14 |

Significance codes: ***$p <$ 0.001, **$p <$0.01, *$p <$ 0.05

See Figure 5.14 for the reasons for choosing the different wallets and the corresponding number of participants. Among the various reasons, the one with the highest number of participants among the Non-newbie group is the security guarantees offered by the wallet, with 75.6% opting for it. In contrast, among the Newbie group, it is the ease of usage of interface with 77.5%.

The most cited reasons for choosing the wallets among the participants across the two groups are the security guarantees they offer, the ease of interface, support for multiple currencies, and the popularity of the wallets.

**Users are less familiar with multi-device wallets.** The self-reported familiarity with the different wallet terms indicates that the users are unfamiliar with multi-device wal-

lets. On a Likert scale of $1-5$ with 1 being "Not-familiar at all", only 3.5% of all the participants claimed that they are 'very familiar' with the threshold wallet while 49.1% claimed to be "Not-familiar at all". The corresponding percentages for multisig wallets are $3.13\%, and 42.7\%$. The mean familiarity over all the wallet types for the Newbie group is 2.02 and for the Non-newbie group is 3.05. The familiarity of the groups with the multi-device wallets is lower at 1.39 and 2.26, respectively. This corroborates with the names of different wallets reported to be used by the participants (see Figure 5.13) where single-device wallets dominate and shows that the participants are less familiar with multi-device wallets.

To overcome this lack of familiarity in the latter part of the survey and to bring all the participants to a similar level of understanding of multi-device wallets, we designed and presented two short videos explaining the advantages and disadvantages of single and multi-device wallets. The videos are followed by two sets of 3 questions each for the explanations are sought. 120 participants got all the 6 correct, showing increased knowledge and familiarity after the videos.

**Ratings and reviews in crowd-sourced platforms heavily affect users' choice of wallets.** Ratings of the wallets seem to affect the users' choice to a great extent – 34.9% of total participants have claimed to choose their wallets *solely* based on ratings of wallets on crowd-sourced platforms like Play Store and App Store. 56.8% of all have reported that ratings and reviews by famous personalities are 'very important' compared to 38.8% who mentioned that they are 'slightly important'. Only 4.3% have claimed ratings and reviews are not important at all. In both the Newbie and Non-newbie groups, at least 29% of each group have claimed to have chosen the wallets solely based on ratings showing their significance (see Figure 5.2a and Figure 5.2b). This shows that ratings and reviews influence the choice of wallets; however, exact such influence on the mental models needs to be studied in the future.

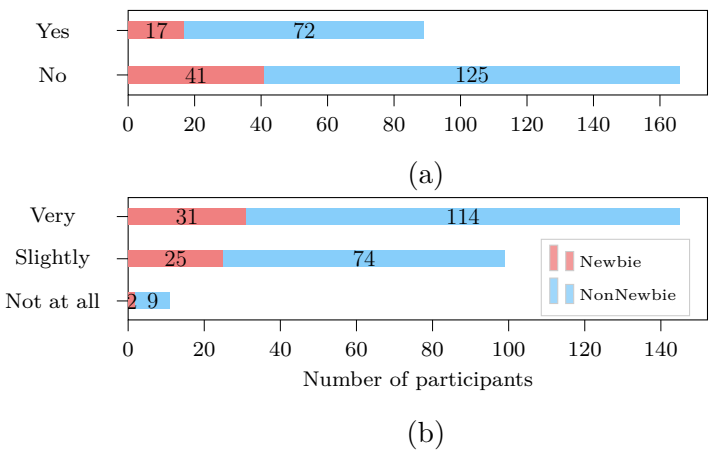



**Figure 5.2.** Response to the questions (a) Was the wallet chosen solely based on ratings (b) Importance of ratings and reviews while choosing a wallet. They show that ratings are reviews are important to many participants.

**The current majority of users are recent adopters and use them for long-term investment**

**Majority of current crypto users are 'recent' adopters.** Of the total participants surveyed, a total of 65% have reported having started using crypto-wallets only in the last two years, and 33% have started using less than a year ago. This shows the rapidly expanding nature of cryptocurrencies in recent years. Among the Non-newbie group, 59.8% of participants have been using it for the last two years. All the participants using the wallets for more than four years are in the Non-newbie group, accounting for 17.25% of that group. Among the Newbie group, 63.7% have started using only in the last year while 93% have reported using them for the previous two years. This is expected as the group identifies itself as one adopting cryptocurrencies for fear of missing out on the crypto boom. Figure 5.3 shows the total number of users for each time period[5]. It indicates that the majority of our participants are recent adopters using them for less than two years.

*. Users employ cryptocurrencies as long-term investment far more than as an alternative to fiat currency 80.3% of the participants reported using cryptocurrencies as a long-term investment, and only 20.8% use them as an alternative for fiat currency. 81.02% of newbies use

[5]↑Questions with a single answer are displayed in red-blue, and ones with multiple answers are shown in green-grey bar graphs.

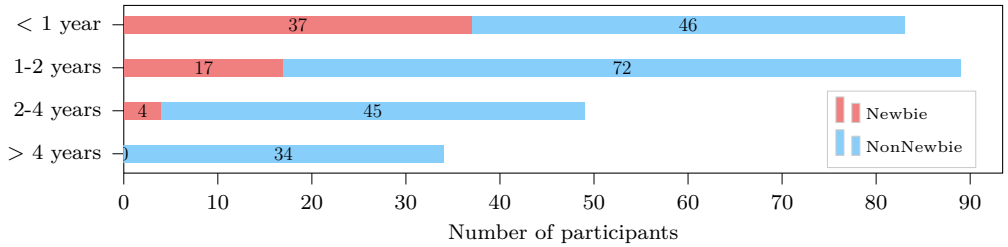



**Figure 5.3.** Duration of crypto-wallet usage by the participants. Most recent-adopters were using them for the last two years.

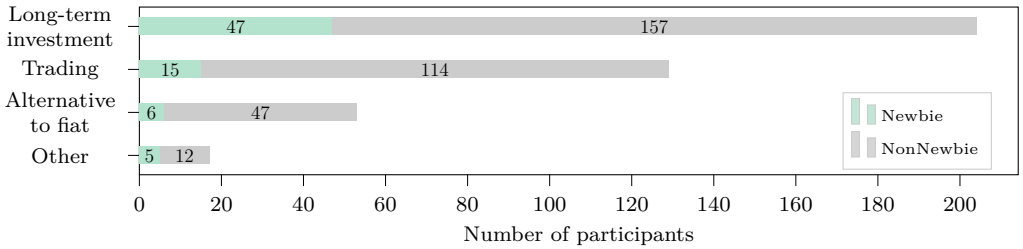



**Figure 5.4.** Purpose of using crypto-wallets; Majority of participants use them for long-term investments and trading.

them for long-term investment, whereas the corresponding percentage among non-newbies is 79.69%. Among non-newbies, 23.8% report using cryptocurrencies as an alternative to fiat currency, and only 10.34% of newbies use it for the same.

**Type of security concerns for user groups.** Different participants have different security concerns regarding the wallets. However, the biggest concern is the loss of funds by compromise of server and secret key hosted by the remote server (see Figure 5.5). The next biggest concern for both groups is the lack of proper recovery mechanisms for the secret key. These are followed closely by the other concerns of compromise and loss of key at the user device and server and the loss of secret key by the user in both groups.

**Social Media as a source of knowledge.** Among the social media followed by the participants for learning about wallets, Twitter and Youtube occupy the top positions, followed by Reddit and Facebook. We asked the participants to choose (or add) all the social media they followed in the survey. The percentage of participants using Twitter, YouTube, Facebook and Reddit among Non-newbies are $59.89\%, 36.54\%, 27.91\%, 27.91\%$ and the corresponding

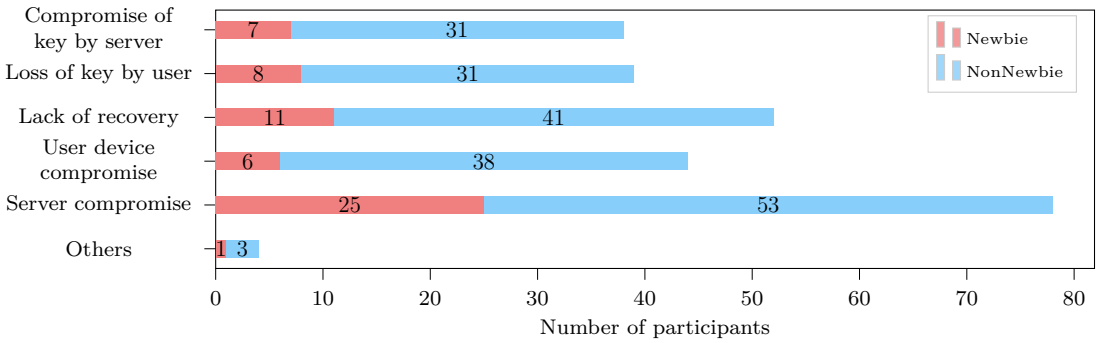



**Figure 5.5.** Biggest security concern of the participants when using a crypto-wallet.

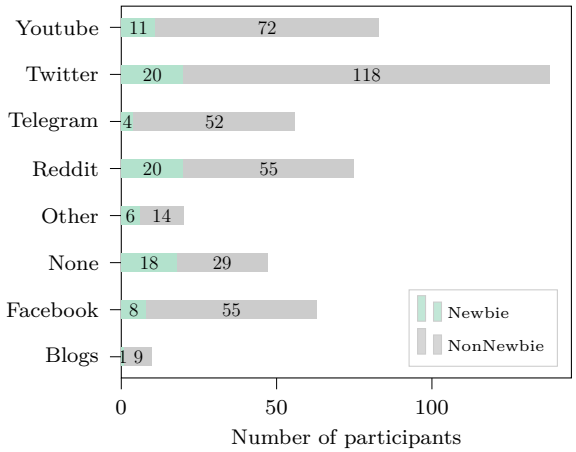



**Figure 5.6.** Followed social media for knowledge and information regarding crypto-wallets.

numbers for newbies are $34.48\%, 18.96\%, 13.79\%, 34.48\%$ (see Figure 5.6). The number of Reddit followers among Newbies is higher than the other social media except for Twitter, showing its increasing popularity among the recent users.

**Users wish to distribute trust for a fixed total number of devices.** To understand if the participants were willing to distribute trust among more entities, we asked if they were willing to increase the value of $T$ (threshold) for a given $N$ (total servers) in a multi-device wallet. 60.4% of the Non-newbie group opted to increase the value of $T$ while 36.54% chose not to. In the Newbie group, these percentages stood at 53.44% and 44.82% (see Figure 5.12). Increasing $T$ would imply distributing the trust among more devices/people. However, since signatures from any $T$ or more are required to authenticate a transaction,

**Table 5.3.** Reasons from our open coding and % of participants for their willingness (non-willingness) to shift from single-device wallet to multi-device wallet.

| | Reasons | % |
|---|---|---|
| Not willing | Single-device wallets are more secure | 37.5% |
| | Single-device wallets are simple to use | 25% |
| | I do not want to lose control of keys | 20.8% |
| | Other reasons | 16.6% |
| Willing | Multi-device wallets are more secure | 79.2% |
| | Other reasons including availability | 20.78% |

it would mean higher communication overhead and possible delays if that person/device does not respond as expected. Thus the trade-off would be between distributing the trust against higher availability of the keys.

### 5.5.2 Users' willingness to shift towards multi-device wallets - RQ2

**The majority of users are willing to shift to multi-device wallets, but few are not**

After learning about multi-device wallets, when asked which wallet they prefer, 67% of all the participants chose multi-device wallets (see Figure 5.7a). The majority of participants wished to shift to them if their current firm offers it; at least 70% of each group wanted to shift (see Figure 5.7b). However, the remaining – slightly $<$ 30% of each group were unwilling to use multi-device wallets. Table 5.3 shows the percentage of participants, the reason for retaining single-device wallets, and shifting to multi-device wallets. Believing that single-device wallets are more secure, simple to use, and retaining control of the secret key are the main motives across the users for remaining with single-device wallets. There is no correlation between the Newbie and Non-newbie groups and their choices of shifting to multi-device wallets (indicated by high $p$ values in the $\chi^2$ analysis).

**Reasons for shifting to multi-device wallets..** Most participants who chose 'Yes', opted for it because multi-device wallets offer better security features like overcoming a single point of failure— P53 explained *"Better security because you need multiple devices to gain access.*

*This also means that even if one device is compromised the attacker can't gain access".* 79.2% of participants who chose to shift opted multi-device wallets for better security features (see Table 5.3).

Around 20.8% participants (of the ones choosing to shift) wanted to shift to multi-device wallets for reasons including ease of access from any device of their choice, better availability in case of loss of a device, and ease of recovery. P202 wrote, *"I can access my wallet from several devices, that's better as I don't have to depend on one device".* In the case of multi-device wallets, the other parties can refresh the shares when a device is compromised. Some participants realized this and chose to shift to multi-device wallets for the easier key recovery; P213 opined *"It is much easier to recover your keys in a multi-device wallet than a single-device wallet."*

**Reasons for not shifting to multi-device wallets.** Among those who opted not to shift to multi-device wallets, when asked to explain, the responses included a few factors — believing that the single-device wallets are more secure and preferring the simplicity to place the trust only on the self to safeguard the keys.

37.5% of the participants who stick to single-device wallets believe they are more secure than the multi-device wallets. They wrote *"Personal hardware keys should be secure enough" (P18), "I will still stick to my single wallet device because it is difficult to compromise."(P99).* However, this is a flawed mental model of security since it is shown [49] that multi-device wallets are more secure than single-device wallets. 20.8% participants wish to use single-device wallets since they want to hold on to the key themselves. Few answered, *"I prefer to be responsible for my keys. If I lose them, that is my fault"(P1), "Id rather keep the key on me at all times so I know where it is and who has it, I only trust myself"(P25).* Another participant, P38 preferred single-device wallets for their simplicity; they said *"I'm happy with the simplicity and current security available with a single-device wallet".*

In multi-device wallets, multiple devices need to communicate and aggregate the signatures collected to compute the final signature. This may induce some delays and also affect the availability. Few participants preferred to stick to single-device wallets for the availability of the keys. P101 mentioned *"I prefer that I try my best to keep the single key safe than*

*run into server down-times. If I decide to use a Multi-device scheme, the devices might have downtime".*

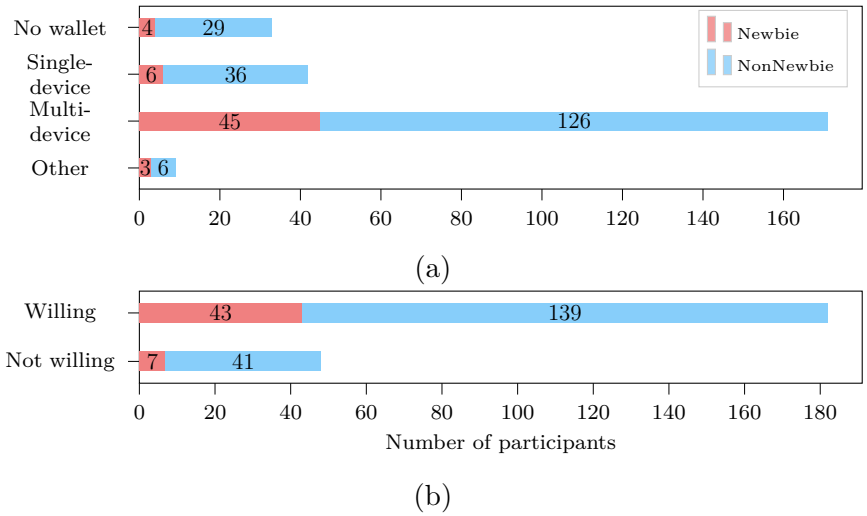



**Figure 5.7.** (a) Preference among single-device and multi-device wallets. (b) Willingness to shift to a multi-device wallet from the employed single-device wallet.

### Among multi-device wallets, users prefer threshold wallets for their privacy

In a threshold wallet, the threshold signature [188] generated to authenticate a transaction does not reveal the access structure, i.e., does not reveal the (N, T) values. In a multisig wallet, the access structure and T (minimum number of required signatures) are revealed. When asked to choose among multi-device wallets, 63.95% of the Non-newbie group and 68.96% of the Newbie group participants chose the threshold wallet over the multisig wallet as shown in Figure 5.8.

These participants opted for threshold wallet for its privacy properties, like not revealing the access structure. On these lines, P158 commented, *"threshold wallet withholds a little more information like the N, T values and this provides more security"*. Some participants realized not knowing the $N$ and $T$ values makes it difficult for the adversary to decide on how many devices to compromise. This provides better security apart from the privacy offered by the threshold signature.

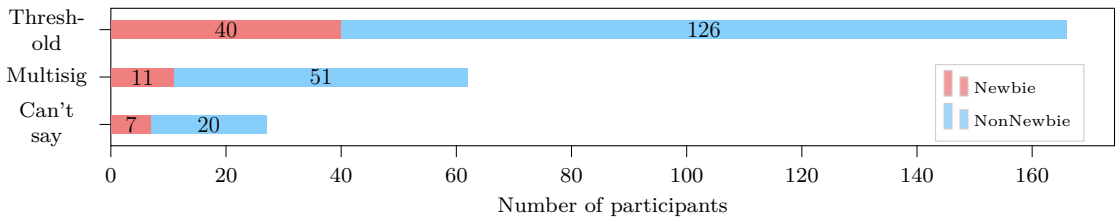



**Figure 5.8.** Preference among the multi-device wallets types – threshold and multisig wallets. The majority choose threshold wallets for offered privacy.

In a multisig wallet, several signatures are collected and aggregated by concatenation, whereas, in a threshold wallet, the threshold signature appears similar to a single device signature. Hence the total data needed to represent the threshold signature is lesser than the multisig signature, making it more space-efficient. While this is a technical aspect to grasp, few participants understand this and have opted for threshold wallet. P246 commented *"Multi-sig wallets are inefficient - requiring several signatures wasting gas. Better implement a threshold wallet with MPC to reduce inefficiencies"*.

Participants chose multisig wallets for their simplicity and *because* it reveals the access structure $(N, T)$. P146 commented *"It is easier for me to know how many devices and threshold I will require to be able to authenticate a transaction. It is easy to use too"*. Multisig wallet signature reveals which parties have provided the signature; if the signature is generated by any collusion, the colluding parties are revealed in the signature. Some participants prefer this over not knowing who signed. Participant P240 who chose the option 'Can not say' wrote, *"The Multisig system clearly labels who the bad nodes are in a collusion attack, which information is missing from the threshold. OTOH, Multisig adds more load on the transactions, as more sigs is more data"*.

After familiarizing themselves through the presented videos, more than two-thirds of participants were willing to shift to multi-device wallets. Among those who wish to use only single-device wallets, 37.5% (wrongly) believe that they are more secure. 20.8% of them choose so because they do not want to lose control over the keys. It should be noted here that, multi-device wallets can indeed provide control over the keys to users. For example, if one share among the two total shares of the key is placed on the user's device, no entity

can access the key and funds without the user's approval and authentication. Among the multi-device wallets, the participants prefer the threshold wallets for the privacy properties they offer. A smaller set of participants prefer the multisig wallet for the simplicity and accountability they impose on the signers. We further investigate the participants' attitude in terms of security by studying the default security settings they prefer for the different wallets.

### 5.5.3 Preferred default settings for crypto-wallets - RQ3

**In single-device wallets retaining agency over the key is preferred over the account compromise risk**

It is natural to choose a particular location for a secret key depending on the risk perception of certain attacks on the system. Hence to understand the participants' risk perception, we investigated their preferred key-storage location for a single-device wallet under different attack scenarios. When asked to choose a location of secret key storage under the specific threat of client-device compromise, the choice of a maximum number of participants of each group is "Multiple remote servers (each storing the key)". This can be expected as one would expect users to opt for remote servers under the client compromise scenario. Hosting the key on multiple remote servers increases the availability of the key while also increasing the risk of being compromised. Many participants in both groups opted for client devices, including desktop/mobile, paper, and hardware tokens as the preferred location for client storage (see Figure 5.15). This indicates that even under vulnerabilities and client device compromise, many wish to retain control over the secret key and thereby the agency over the funds. In the remote server compromise scenario (see Figure 5.16) the three key storage locations chosen by the highest number of participants are paper, client desktop/mobile, and hardware token.

**For multi-device wallets, users weigh reputation over distributing the attack surface**

To understand the settings that the users prefer for multi-device wallets, we asked the participants to pick among three choices — (i) a smaller number of reputed servers, (ii) a

large number of servers with a much lower threshold, (iii) a large number of servers with a high threshold. In the case of a smaller number of reputed servers, they would provide higher availability with very few servers needing to respond; however, the attacker just needs to compromise those few servers. In the second option, the servers are randomly chosen (with a certain criterion) among many servers across the globe but with a lower threshold. Here the attacker is not readily sure of which servers to attack even though the threshold is small. The last option has a higher threshold indicating that the attacker needs to compromise a high number of servers. We deliberately provided options with numbers starkly showing these differences to bring them to participants' attention.

More than half of the participants placed their trust in reputation rather than the inability of attackers to compromise a large number of servers distributed across the globe. 65.48% of Non-newbies and 55.17% of Newbies chose a small number of servers ($(10, 5)$ in Figure 5.9) hosted by reputed firms. Participants seem to trust the reputed servers to take good security measures as their reputation is at stake in case of compromise. P38 wrote, *"I prefer servers hosted by well-known reputed firms as they are likely to have stringent security measures to stop any breaches."* Few chose a smaller number of servers since maintaining and keeping track of a large number of them can be a complex task; they wrote, *"Keep it simple. More servers, more things to go wrong" (P244).*

Among the parties who opted for choices with more servers, increasing the number of servers for the adversary to attack is the most quoted reason. P25 said *"The more there are, the harder it will be to be compromised. Being random servers, it is also harder to track them down".* Another interesting aspect is that reputed firms can become centers for targeted attacks by adversaries. Given this one participant, P76 said *"I would prefer randomly chosen servers as they are less likely to be targeted than established companies"* while choosing the $(100, 50)$ setting. The participants who chose a larger number of servers and a low threshold $(100, 5)$ opted for high availability of keys; even if many servers are down, the secret information is available to the clients.

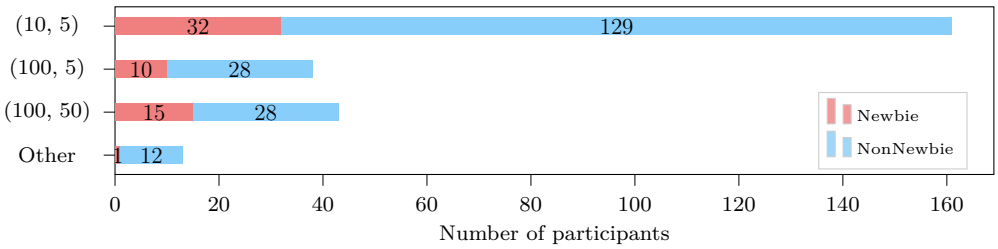



**Figure 5.9.** Preferred settings for $(N, T)$ multi-device wallet. $N$ is the total number of devices, and $T$ is the minimum number of devices needed to generate the signature. $(10, 5)$ is with servers hosted by reputed firms. In other settings, servers are chosen randomly across the globe.

### Users wish to distribute trust for a fixed total number of devices

When asked if the participants were willing to increase the value of $T$ (implying distributing the trust among more devices/people), majority of them opted for it (see **??** 11 for details). The participants were allowed to choose the 'Other' option followed by a text response which we analyzed; Few of them who chose 'Other' indicated that they did not wish to simply distribute the secret key among more parties but carefully tailor the threshold for the scenario.

### The government policies affect the preference for share-distribution

Any server hosted in a particular country is subject to the local government privacy policies. Depending on the policy, few governments may be able to view or even block access to any cryptocurrency server data if they wish (here, we assume a setting where the governments do not share data with each other). Thus, the location of the hosted server is significant in terms of privacy and availability of keys to the users. Our survey explores users' preferences for the location of these servers for different secret-key distributions among client devices and remote servers. We investigate these preferences for both single-device and multi-device wallet scenarios under different government characteristic settings. For Threshold wallets, the participants were allowed to choose from (i) sharing the key among servers, (ii) dividing the key in two parts ($Share_1$ and $Share_2$), placing one part $Share_1$ on the client, and sharing the second part $Share_2$ among all the servers.

**Users do not prefer server locations where governments can block data access.** For threshold wallets, whenever the government can not block access, irrespective of the government can view the data, at least 50% of the Newbie group are willing to opt for sharing the key only among servers. When the government can block access, less than 37.93% chose to share among servers, with 63.79% choosing to share between the client device and the servers. In the Non-newbie group, more than 55% always wanted to place a share on the client device, which went up to 69% when the government could deny access to the data (see Figure 5.11). The responses are significantly correlated against the different government characteristics with $\chi^2$ test $p$-value of 0.001. Thus government policies greatly affect the choice of location for the secret and majority of users wish to have a share of the key on their devices when the government can deny access.

In short, in our study, most participants wish to retain control over the secret key despite its vulnerabilities. They prefer the keys distributed on a smaller number of servers hosted by reputed firms; they also like to increase the threshold for a fixed number of servers to distribute the trust further. The governments' ability to block access to their secret information affects their choices of key locations.

**Settings for single-device wallets.** For the single-device scenario, as long as the government can not block access to the server data, more than 56% of the Newbie group was willing to place the secret information on multiple servers. However, when the government can deny data, this percentage is less than 36%. Among the Non-newbie group, when the government cannot view and deny data, 60.9% of participants were willing to place the secret shares on multiple servers while this drops to 41.6% when the government can view and deny data to the clients (see Figure 5.10). The $p$-value of 0.013 in the $\chi^2$ test shows significant correlation of responses against the government characteristics.

## 5.6 Implications

Our study offers the developers specific insights into the settings and architectures for their wallets. We also observe few interesting threshold cryptographic research problems.

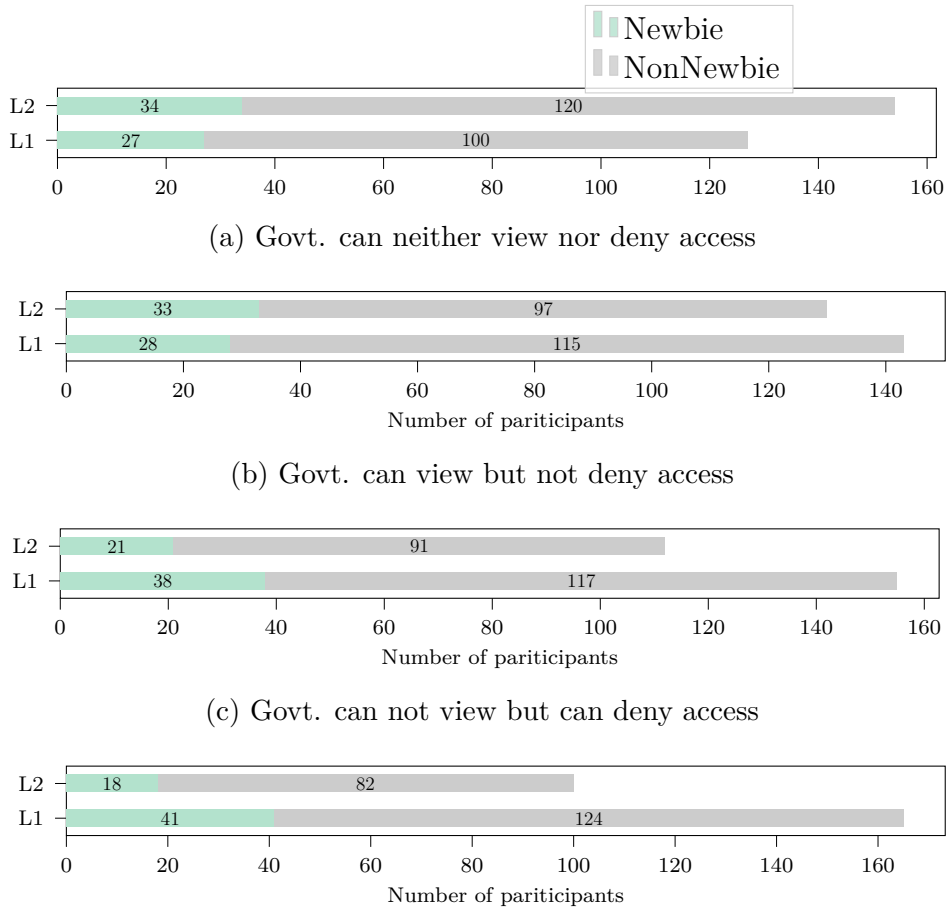
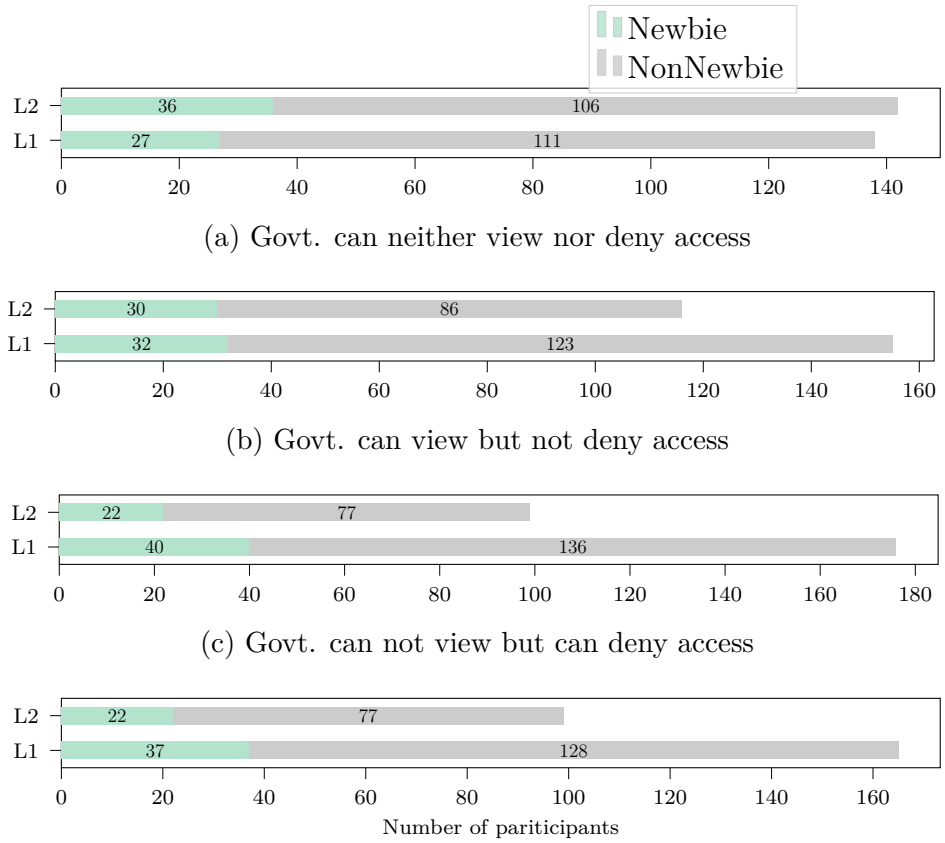



**Figure 5.10.** Single device wallet – user key location preference under different government characteristics L2 - The key is on multiple remote servers across different countries (these countries do not share data). L1 -The key is on client desktop/mobile/hardware token.

### Educating the users

As participants prefer multi-device wallets for better security, this study encourages developers who have developed or are considering a multi-device version of their wallets. However, about 37% of participants who were unwilling to shift to multi-device wallets believed that single-device wallets are more secure. This flawed mental model needs to be addressed by educating the users about the security features of multi-device wallets. It can be achieved by nudging the users towards better practices [230], [231]. About 20.5% of them wanted to stick to single-device wallets because they did not want to lose control over the




(a) Govt. can neither view nor deny access




(b) Govt. can view but not deny access




(c) Govt. can not view but can deny access




(d) Govt. can both view and deny access

**Figure 5.11.** Threshold Wallet – user key location preference under different government characteristics. L2 - Threshold-share the key among multiple servers. L1 - Divide the key into two parts. Place one part on the client-device. Threshold-share the other part among multiple-servers.

keys. This should encourage the multi-device wallet developers to choose settings where they provide control of the keys to the user and convince them of the same.

### Distributed server setup for multi-device wallets

While choosing a distributed server setup to host the shared keys, our study can significantly help developers arrive at a setting. We learn that the majority of the participants prefer a smaller set of reputed servers in locations where the governments cannot deny access to the data (see 5.5.3). Among the different share distributions (or access structures), as chosen by the participants, the developers should consider always placing a share on the

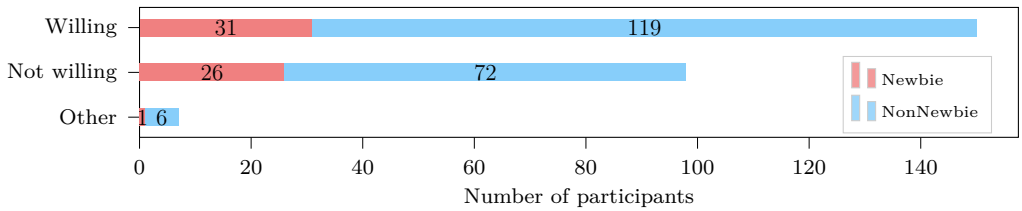



**Figure 5.12.** Willingness to increase $T$ for a fixed $N$ in a $(N, T)$ multi-device wallet. It shows willingness of participants to distribute trust among higher (threshold) number of nodes.

**Table 5.4.** $p$-values for the Chi-squared test for 3 group classification — Newbie, Trader and Techie. Here we present the variables that had significant $p$-values for Newbie-Non newbie classification (in Table 5.1). **df** is the degrees of freedom.

| Variable | $\chi^2$ | df | p-value |
|---|---|---|---|
| Number of years | 2.21 | 3 | 0.33 |
| % of savings invested | 6.523 | 4 | 0.58 |
| Background in IT | 9.534 | 2 | 0.049* |
| Usage - Trading | 0.252 | 1 | 0.881 |

Significance codes: ***$p<$ 0.001, **$p<$0.01, *$p <$ 0.05

client device to give them control (see Section 5.5.3). This can be achieved by generating two shares of the secret key $Share_1$ and $Share_2$, placing, say $Share_1$ on the client device, and dividing $Share_2$ among multiple servers. Note that threshold wallet ZenGo [56] already follows this pattern with only one server share, while Torus [55] wallet offers no such control to the users. It is important for the wallet firms to choose the proper default settings the covers most user preferences as most users may just choose them [232]–[235]. The users consider the location of the servers (see Section 5.5.3), so the firms offering the wallets must also allow the users choose the servers on which to host their secret key shares.

**General adversary access structure.** In fact, the developers can consider general adversary structure secret sharing (GASS) [236]–[239] for their wallets. In a typical threshold cryptographic setting, the adversary can corrupt up to a fixed fraction of players. However, GASS considers more general adversary corruption patterns, in which the adversary is allowed to corrupt any set of players in some pre-defined collection of sets (or access structure).

**Table 5.5.** $p$-values for the Mann-Whitney U test for 3 group classification — Newbie, Trader and Techie. In the table we only present the variables that had significant $p$-values for Newbie-Non newbie classification (in Table 5.2).

| Variable | Newbie-Trader | Trader-Techie | Techie-Newbie |
|---|---|---|---|
| **Familiarity- Wallet** | | | |
| Paper | 0.00016*** | 0.8471 | 0.0093** |
| Exchange | 2.21e-09*** | 0.8114 | 0.00068*** |
| Desktop/Mobile | 7.40e-08*** | 0.0766 | 0.0370* |
| Hardware | 3.22e-11*** | 0.9900 | 1.69e-05*** |
| Multisig | 1.39e-07*** | 0.3980 | 0.0059** |
| Threshold | 2.51e-07*** | 0.2068 | 0.0043** |

Significance codes: ***$p$< 0.001, **$p$<0.01, *$p$ < 0.05

**Table 5.6.** Percentage of participants who answered correct (and wrong) in the 6 knowledge-test questions after the videos – Questions 19 and 25 of the survey instrument. The comparison is between two groups of participants who have been using multi-device wallets and single-device wallets.

| Question | multi-device users | single-device users |
|---|---|---|
| Q19(a) | 92% (8%) | 97.9% (2.1%) |
| Q19(b) | 92% (8%) | 94.8% (5.2%) |
| Q19(c) | 92% (8%) | 95.3% (4.7%) |
| Q25(a) | 92% (8%) | 86% (14%) |
| Q25(b) | 76% (24%) | 80% (20%) |
| Q25(c) | 68% (32%) | 63% (37%) |

Developing personalized threshold wallets based on GASS enables the developer to realize individual users' adversary mental models better and be more realistic for a wallet design. General adversary structure secret sharing and threshold signing for multi-device wallets can be an interesting design and implementation target for the research community and the wallet developers.

We note that implications of the user control go beyond the multi-device wallets setting and are also highly relevant for NIST threshold cryptography efforts [58] as well as fast-growing multi-party computation (MPC) based privacy-preserving machine learning (PPML) [240], [241]. For example, in the context of NIST's threshold cryptography initiative, it will be an interesting research problem to design a secure threshold ECDSA protocol

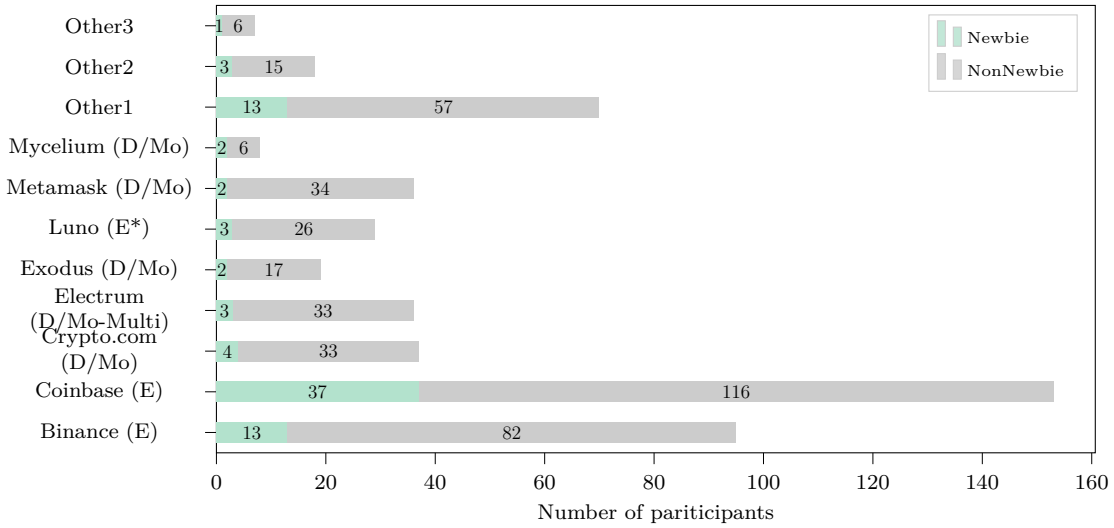



**Figure 5.13.** Currently used crypto-wallets. D/Mo- Desktop/Mobile, H - Hardware, Multi - Supports MultiSig, E* - Behaves like exchange wallet by maintaining keys.

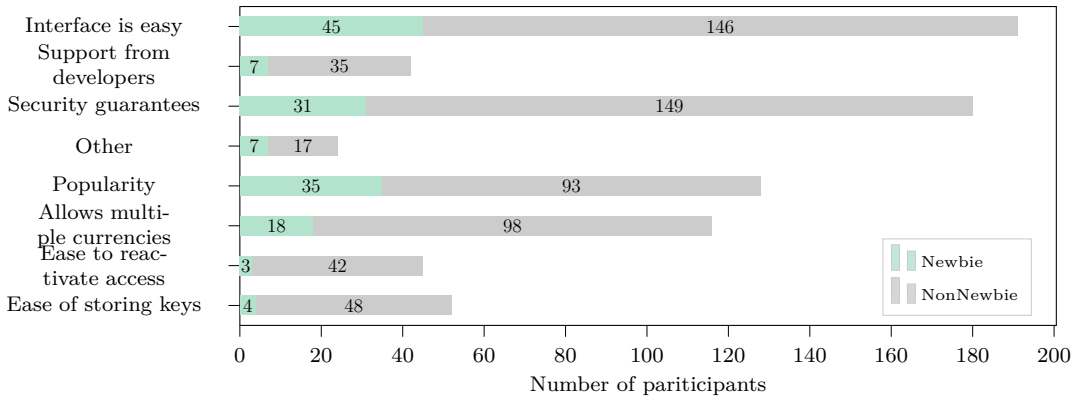



**Figure 5.14.** Reasons for choosing the most used crypto-wallets by the participants.

that maintains the users' control over the keys in the wallets. The current threshold ECDSA protocols [193], [215], [216], [242] cannot securely realize users' control in the above-described setting where one of the two shares is re-shared among the servers.

**Threshold vs. Multisig wallets.** Threshold and multisig wallets offer interesting trade-offs concerning accountability and privacy. While many participants prefer the privacy provided by the threshold wallet, some do not wish to use them for the exact reason that they do not reveal enough information (see Section 5.5.2). For example, if a signature is generated

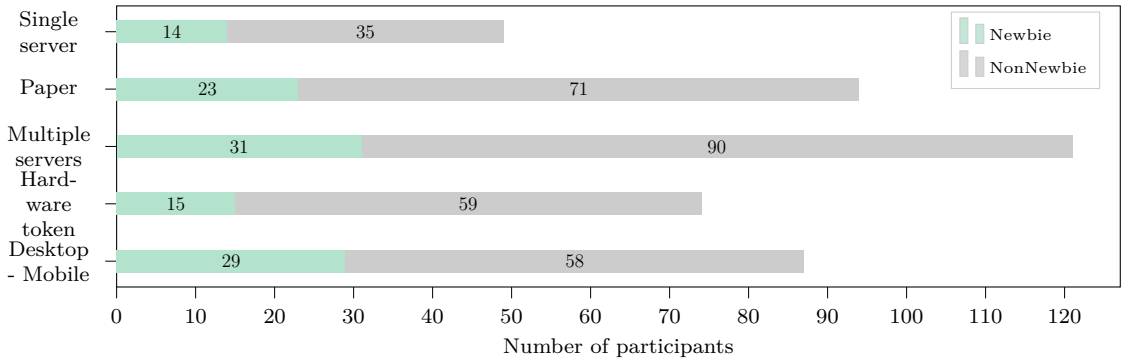



**Figure 5.15.** Key storage location preference under client device compromise scenario.

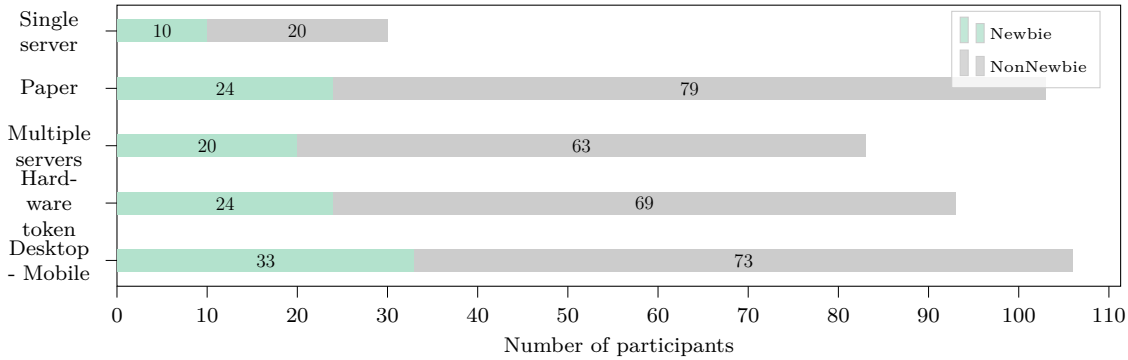



**Figure 5.16.** Key storage location preference under remote server compromise scenario.

under collusion, the information of who is involved is not revealed in a threshold signature but is revealed under multi-signatures.

This study shows that users understand and consider the trade-offs in two types of multi-device wallets. This motivates security research toward signature generation and wallet design to offer the best of both worlds, including privacy and accountability. Furthermore, since participants are concerned about the space requirements of multi-signatures in wallets (see Section 5.5.2), developing space-efficient multi-signature schemes is an interesting problem to consider.

# 6. D-KODE: KEY-DISTRIBUTION USING A LATTICE BASED KEY-HOMOMORPHIC PRF

## 6.1 System Setup and Solution Overview

### 6.1.1 System Setup

Consider a system of $n$ servers $\{P_1, P_2, \cdots, P_n\}$ that share a master secret (vector) $\mathsf{k}$[1] through a $(n, t)$-threshold scheme. The servers interact with clients who join and leave the network anytime. All the servers have access to a broadcast channel and the network is synchronous. We consider a $t-$bounded static adversary that corrupts up-to $t$ servers at the start of the protocol. Corrupted servers remain so through-out the protocol run. Each pair of servers is connected through a secure channel that provides secrecy and authenticity; this is typically achieved through TLS channels [243] which mitigate any man-in-the-middle attacks. While we consider a static adversary model for the distributed key generation mechanism, we extend it to a mobile adversary model for the proactive secret sharing mechanism discussed in Section 6.7. The secrecy/confidentiality of the secret key in the D-KODE-protocol is based on the discrete logarithm (DLog) and Learning-with-rounding assumptions:

**Definition 6.1.1.** *The Discrete Logarithm (DLog) assumption [244]: For a generator $g \in \mathbb{G}$ and a $a \xleftarrow{\$} \mathbb{Z}_q$, given the value $g^a$, the probability of a ppt algorithm $\mathcal{A}_{DLog}$ to output the value $a$, $Pr[\mathcal{A}_{DLog}(g, g^a) = a]$ is negligible.*

**Definition 6.1.2.** *The Learning-with-rounding (LWR) [245] problem consists of distinguishing the distribution $(\boldsymbol{A}, \lfloor \boldsymbol{As} \rfloor_p)$ where $\boldsymbol{A} \sim U(\mathbb{Z}_q^{m \times n}), \boldsymbol{s} \sim U(\mathbb{Z}_q^n)$ and the uniform distribution $U(\mathbb{Z}_q^{m \times n} \times \mathbb{Z}_p^m)$; $q \geq 2$. We say that the $\mathsf{LWR}_{(q,m,n)}$ is hard if for all ppt algorithm $\mathcal{A}$, the advantage $\boldsymbol{Adv}_{q,m,n}^{\mathsf{LWR}}(\mathcal{A}) = \|Pr[\mathcal{A}(\boldsymbol{A}, \lfloor \boldsymbol{As} \rfloor_p) = 1] - Pr[\mathcal{A}(\boldsymbol{A}, \boldsymbol{u}) = 1]\|$ is negligible, with the probabilities taken over $\boldsymbol{A} \sim U(\mathbb{Z}_q^{m \times n}), \boldsymbol{s} \sim U(\mathbb{Z}_q^n)$, and $\boldsymbol{u} \sim U(\mathbb{Z}_p^m)$.*

[1] ↑We denote all vectors and matrices in bold font.

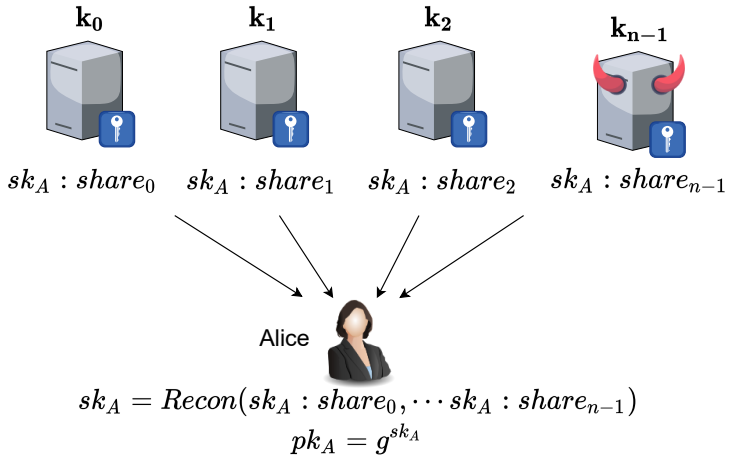



**Figure 6.1.** Scenario 1a: Alice uses her public string $\mathsf{ID}_A$, obtains evaluations and reconstructs private key $sk_A$ after authentication

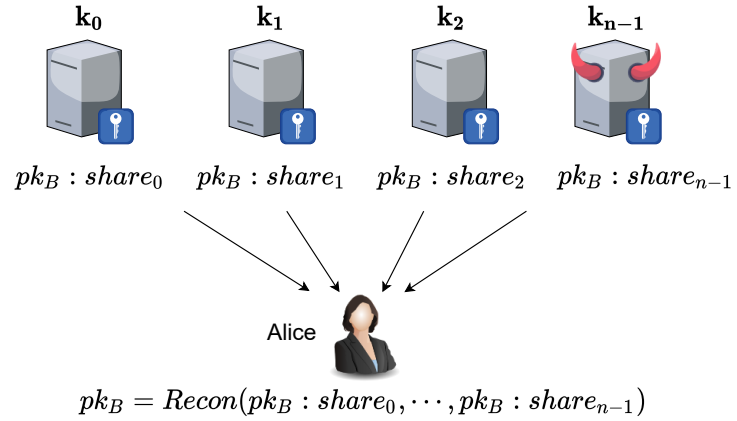



**Figure 6.2.** Scenario 2: Alice uses Bob's public string $\mathsf{ID}_B$ to obtain his public key shares and compute the public key $pk_B$

### 6.1.2 Design Overview

In the D-KODE protocol, a master key $\mathsf{k}$ is $(n, t)$-threshold secret-shared among $n$ servers and the client private key is computed as the PRF [71] evaluation $F(X, \mathsf{k}) = \left\lfloor H(X) \cdot \mathsf{k} \right\rfloor_p \in \mathbb{Z}_p$, for $X \in \mathcal{X}$ where $\mathcal{X}$ is the client-input space, $\boldsymbol{K} \in \mathbb{Z}_q^u$ the server key and $H : \{0,1\}^* \to \mathbb{Z}_q^u$ a cryptographic hash function. $(\cdot)$ indicates the vector dot product computation. The master key vector $\mathsf{k}$ is shared among the servers with each server $P_{\mathrm{i}}$ obtaining the share matrix $\boldsymbol{K}_{\mathrm{i}}$, the PRF evaluation of each server is $\lfloor H(X) \cdot \boldsymbol{K}_{\mathrm{i}} \rfloor_p$. The shares $\boldsymbol{K}_{\mathrm{i}}$ are generated in a

distributed manner using distributed key generation (DKG) involving verifiable black box secret sharing (BBSS) scheme (elaborated in Section 2.1.2). The BBSS scheme involves a *distribution matrix* which is constructed such that the reconstruction coefficients for the shares are in the set $\{-1, 0, 1\}$. It is done by realizing the $(n, t)$-threshold access structure as a threshold circuit and expressing it as a monotone boolean function. This function is then converted to a distribution matrix using the Benaloh and Leichter [94] construction (recalled in 2.6).

Let each server $P_{\mathsf{i}}$ be associated with a set $T_{\mathsf{i}}$ such that $P_{\mathsf{i}}$ receives the matrix $\boldsymbol{K}_{\mathsf{i}} = \{\boldsymbol{k}_{\mathsf{j}}, \mathsf{j} \in T_{\mathsf{i}}\}, \boldsymbol{k}_{\mathsf{j}} \in \mathbb{Z}_q^u$. The partial evaluations of server $P_{\mathsf{i}}$ upon client input $X$ is a vector of evaluations $\{F(X, \boldsymbol{k}_{\mathsf{j}}), \mathsf{j} \in T_{\mathsf{i}}\}$. To compute the required keys, the client forwards the public string $X$, obtains partial evaluations and reconstructs the corresponding keys. Let $y = F(X, \mathsf{k})$ and $y_\ell = F(X, \boldsymbol{k}_\ell), \ell \in \cup_{\mathsf{i}} T_{\mathsf{i}}$ be the set of all partial evaluations received from the servers. To generate the private key the client obtains a linear combination $\tilde{y} = \sum_{\mathsf{i} \in S} \lambda_{\mathsf{i}} \cdot y_{\mathsf{i}}$ where each $\lambda_{\mathsf{i}} \in \{0, 1, -1\}$. $\tilde{y}$ differs from $y$ by an error $\mathsf{e} = \pm t'$ when $t' > t$ evaluations have been used for the computation.

**(Scenario 1) Private key of Alice, the online client.** Alice securely authenticates herself to the servers (using email-login, OAuth tokens etc.) and forwards her public string $\mathsf{ID}_A$ (for example, her email ID), obtains the partial evaluations $y_\ell = F(\mathsf{ID}_A, \boldsymbol{k}_\ell)$ from servers and computes the private key as $sk_A = \sum_{\mathsf{i}} \lambda_{\mathsf{i}} \cdot y_{\mathsf{i}}$ as depicted in Figure 6.1. The values of $\lambda_{\mathsf{i}}$ are determined by the qualified set of servers whose evaluations are utilized in the reconstruction (refer Section 2.1.2). From the private key $sk_A$, she can compute the public key as $pk_A = g^{sk_A}$. With the key pair $(sk_A, pk_A)$ she can perform any required transaction.

**(Scenario 2) Public key of Bob, the offline client.** When Alice tries to pay Bob, she forwards Bob's public string $\mathsf{ID}_B$ to the servers and obtains the evaluations $z_\ell = g^{y'_\ell}$ where $y'_\ell = F(\mathsf{ID}_B, \boldsymbol{k}_\ell)$ as depicted in Figure 1.2. She computes a public key of Bob as $pk_B = \prod_{\mathsf{i}} (z_{\mathsf{i}})^{\lambda_{\mathsf{i}}}$ and proceeds to pay Bob using the computed public key $pk_B$.

When Bob tries to compute his private key later corresponding to this public key $pk_B$, he authenticates to the servers and obtains a private key $sk'_B$ which differs from the private key $sk_B$ (corresponding to the public key $pk_B$), by a maximum of $n$. He simply computes

all the private keys in the range $[sk'_B - n, sk'_B + n]$, obtains the corresponding public keys $[g^{sk'_B - n}, g^{sk'_B + n}]$ and queries for balance on the $2n$ (where $n \in [5, 50]$) public keys in this set from the crypto-currency system. $pk_{\mathrm{B}}$ will be in that set of $2n$ keys and since he has private keys corresponding to all of them, he can utilize the funds transferred by Alice to $pk_{\mathrm{B}}$. Note that only Bob owns these secret keys.

Thus Alice can pay Bob by computing $sk_A$ in scenario 1 and pay him by computing $pk_B$ in scenario 2. Bob can later compute the corresponding key $sk_B$ and retrieve the funds whenever necessary. This solution does not involve any interaction between the servers for the computation of client keys, since they just evaluate $y'_{\mathrm{i}} = F(\mathsf{ID}_B, \boldsymbol{k}_{\mathrm{i}})$ and forward $g^{y'_{\mathrm{i}}}$ to the client non-interactively. In summary, the proposed solution for the two scenarios consists of following steps:

- The servers $P_{\mathrm{i}}, \mathrm{i} \in [n]$ participate in DKG involving BBSS and obtain shares $\boldsymbol{K}_{\mathrm{i}} = \{\boldsymbol{k}_{\mathrm{j}}, \mathrm{j} \in T_{\mathrm{i}}\}$ of a master key $\mathsf{k}$
- For Scenario 1: The servers generate partial evaluations $y_\ell = F(X, \boldsymbol{k}_\ell)$ using the server key shares $\boldsymbol{K}_{\mathrm{i}}$ and public input string input $X$ from the client. The client combines the shares to compute the private key evaluation $y = F(X, \mathsf{k})$.
- For Scenario 2: The servers evaluate $y'_{\mathrm{i}} = F(X', \boldsymbol{k}_{\mathrm{i}})$ and forward $g^{y'_{\mathrm{i}}}$ for the evaluation of public key $z = g^{y'}$ for the input $X'$ from any client.

Since we envisage a full-fledged deployment where the servers are used for the evaluation of keys for a very large number of clients over a long period of time, we propose proactive secret sharing mechanism for BBSS. The servers store only one set of key shares corresponding to the master key $\mathsf{k}$ and perform share-refreshing periodically using the proposed Proactive BBSS scheme (refer Section 6.7). For share refreshing, the servers re-share each of their share elements to the set of servers in the next time period. The servers then compute the new shares from the shares of the share-elements.

We implement the full protocol and extract many interesting aspects of BBSS scheme in the practical regime. While the existing works discussing BBSS and the related Linear Integer secret sharing (LISS) scheme [93], [245] have shown that the circuit size for the construction of distribution matrix varies from $O(n^{5.3}) - O(n^{2.414})$, we show that for certain threshold access structure regimes, this size is much smaller than what can be theoretically

surmised from the literature, making the sharing mechanisms very efficient in the practical setting.

## 6.2 Verifiable BBSS (V-BBSS)

Verifiability of a secret sharing scheme is the property which lets the parties receiving the shares from a dealer to verify the validity of the shares. Several verifiability techniques [246]–[248] have been proposed for different secret sharing schemes, here we discuss the verifiability of the BBSS scheme.

After generating the share elements by performing $\boldsymbol{s} = \boldsymbol{M} \cdot \boldsymbol{\rho}$, for a distribution matrix $\boldsymbol{M}$ and a random vector $\boldsymbol{\rho} = \{\rho_1, \rho_2, \cdots, \rho_{\mathrm{e}}\} \in \mathbb{Z}_q^{\mathrm{e}}$, the dealer commits to each element of the vector $\boldsymbol{\rho}$ and forwards the commitments to all the parties receiving the shares. The matrix $\boldsymbol{M} = m_{\mathrm{i,j}}, \mathrm{i} \in [d], \mathrm{j} \in [\mathrm{e}]$ is public and is known to all the parties.

We briefly sketch the different steps of the Verifiable-BBSS scheme [249]:

**Share Generation.** : The dealer samples a random vector $\boldsymbol{\rho} = \{\rho_1, \rho_2, \cdots, \rho_{\mathrm{e}}\} \in \mathbb{Z}_q^{\mathrm{e}}$ and sets the element $\rho_1$ to the desired secret value $s$ to be shared. For a $(n, t)$ threshold sharing, he computes the distribution matrix $(M)$ and generates share element vector $\boldsymbol{s} = \boldsymbol{M} \cdot \boldsymbol{\rho}, \boldsymbol{s} = \{s_{\mathrm{i}}\}, \mathrm{i} \in [d]$. The dealer generates a commitment vector $C$ consisting of commitments $C_{\mathrm{i}}$ to each element of the vector $\boldsymbol{\rho}$. The element $\rho_{\mathrm{i}}$ is committed using Pedersen commitment as $C_{\mathrm{i}} = g^{\rho_{\mathrm{i}}} h^{\rho'_{\mathrm{i}}}$ using random $\rho'_{\mathrm{i}} \in \mathbb{Z}_q$. The dealer also computes the vector $\boldsymbol{s}' = \boldsymbol{M} \cdot \boldsymbol{\rho}'$ where $\boldsymbol{\rho}' = (\rho'_1, \rho'_2, \cdots, \rho'_{\mathrm{e}})$ and $\boldsymbol{s}' = \{s'_{\mathrm{i}}\}, \mathrm{i} \in [d]$

The dealer forwards the share vectors $\boldsymbol{s}_{\mathrm{i}} = \{s_{\mathrm{j}}\}, \boldsymbol{s}'_{\mathrm{i}} = \{s'_{\mathrm{j}}\}, \mathrm{j} \in T_{\mathrm{i}}$ to party $P_{\mathrm{i}}$ where $T_{\mathrm{i}}$ is the set of all row indices owned by party $P_{\mathrm{i}}$. The dealer also *broadcasts* the commitment vector $C$ to all the parties.

## 6.3 Distributed Key Generation using BBSS

A distributed key generation (DKG) [30] protocol allows a set of nodes to share a secret among themselves without a trusted third party such that any qualified subset of nodes can use/reveal their shares to compute the secret. However, any subset of nodes outside the set of qualified sets has no information about the shared secret. For a $(n, t)-$DKG, any

**BBSS-DKG**

pp: $\{n, t, q, p, \boldsymbol{M} \in \{0,1\}^{d \times \mathrm{e}}, \psi(\cdot)\}$
Phase 1: Generating shares of $sk \in \mathbb{Z}_q$ :

1. Each party $P_\mathrm{i}$ performs a Verifiable BBSS of a random value $z_\mathrm{i} \in \mathbb{Z}_q$ :

   (a) $P_\mathrm{i}$ chooses two random vectors $\boldsymbol{\rho}_\mathrm{i} = \{\rho_{\mathrm{i},1}, \rho_{\mathrm{i},2}, \cdots, \rho_{\mathrm{i},\mathrm{e}}\}$ and $\boldsymbol{\rho}'_\mathrm{i} = \{\rho'_{\mathrm{i},1}, \rho'_{\mathrm{i},2}, \cdots, \rho'_{\mathrm{i},\mathrm{e}}\}; \boldsymbol{\rho}_\mathrm{i}, \boldsymbol{\rho}'_\mathrm{i} \in \mathbb{Z}_q^\mathrm{e}$. Sets the first element $\boldsymbol{\rho}_{\mathrm{i},1} = z_\mathrm{i}$.

   (b) $P_\mathrm{i}$ computes two vectors $\boldsymbol{S}_\mathrm{i} = M \cdot \boldsymbol{\rho}_\mathrm{i}$ and $\boldsymbol{S}'_\mathrm{i} = M \cdot \boldsymbol{\rho}'_\mathrm{i}$, generates commitment vector $\boldsymbol{C}_\mathrm{i}$ consisting of commitments to each of the elements of the vector $\boldsymbol{\rho}_\mathrm{i}$ as $C_{\mathrm{i},l} = g^{\rho_{\mathrm{i},l}} h^{\rho'_{\mathrm{i},l}}; l \in [\mathrm{e}]$ where $g, h$ are generators of a multiplicative group $\mathbb{G}$. Let the computed vectors be $\boldsymbol{S}_\mathrm{i} = \{s_{\mathrm{i},1}, s_{\mathrm{i},2}, \cdots, s_{\mathrm{i},\mathrm{e}}\}, \boldsymbol{S}'_\mathrm{i} = \{s'_{\mathrm{i},1}, s'_{\mathrm{i},2}, \cdots, s'_{\mathrm{i},\mathrm{e}}\}$.

   (c) $P_\mathrm{i}$ forwards the shares $\boldsymbol{s}_{\mathrm{i},\mathrm{j}}$, a subset of the vector $\boldsymbol{S}_\mathrm{i}$ to $P_\mathrm{j}$ consisting of share elements $s_{\mathrm{i},k}, k \in \{T_\mathrm{j} = \psi^{-1}(\mathrm{j})\}$ and it also forwards the corresponding $\boldsymbol{s}'_{\mathrm{i},\mathrm{j}}$, a subset of the vector $\boldsymbol{S}'_\mathrm{i}$ to the $P_\mathrm{j}, \mathrm{j} \in [n]$.

   (d) $P_\mathrm{i}$ broadcasts its commitment vector $\boldsymbol{C}_\mathrm{i}$ with elements $C_{\mathrm{i},l}, l \in [\mathrm{e}]$ to every other party $P_\mathrm{j}$, $\mathrm{j} \in [n]$.

   (e) $P_\mathrm{j}$ verifies the shares it received from the other parties using the specified verification procedure. $s_{\mathrm{i},k}$ (corresponding to the row $k$ of the vector $\boldsymbol{S}_\mathrm{i}$ of $P_\mathrm{i}$) received by $P_\mathrm{j}$ from $P_\mathrm{i}$ is verified as: $g^{s_{\mathrm{i},k}} h^{s'_{\mathrm{i},k}} = \prod_{l=1}^{\mathrm{e}} C_{\mathrm{i},l}^{m_{k,l}} \bmod p$. (Here row $k$ is held by $P_\mathrm{j}$, $k \in T_\mathrm{j}$).
   If any verification fails, party $P_\mathrm{j}$ broadcasts a complaint against party $P_\mathrm{i}$ by broadcasting the shares $(s_{\mathrm{i},k}, s'_{\mathrm{i},k})$.

   (f) On receiving a compliant against self from $P_\mathrm{j}$ for any row $k$, $P_\mathrm{i}$ reveals the shares by broadcasting $\boldsymbol{s}_{\mathrm{i},k}, \boldsymbol{s}'_{\mathrm{i},k}$.

2. Every party maintains a set of parties *Qualified* $\mathcal{Q}$, any party excluded from the set is disqualified by that particular party. Every party $P_\mathrm{j}$ excludes a party $P_\mathrm{i}$ if $P_\mathrm{i}$ either receives more that $t$ complaints or the broadcasted shares after complaint do not pass the verification. At the end of the complaint and verification phase, every honest party will have the same qualified set $\mathcal{Q}$.

3. Every party $P_\mathrm{j}$ locally forms its shares of the secret key $sk$ by adding element-wise, the shares of the vectors $\boldsymbol{s}_{\mathrm{i},\mathrm{j}}$ received from every other party $P_\mathrm{i}, \mathrm{i} \in [n]$ i.e., each $P_\mathrm{j}$ computes its share as $\boldsymbol{sk}_\mathrm{j} = \{\hat{s}_k \| k \in T_\mathrm{j}\} = \sum_\mathrm{i} s_{\mathrm{i},k}$ for each $k \in T_\mathrm{j}$. Share of each party $P_\mathrm{j}$ is a vector $\boldsymbol{sk}_\mathrm{j}$ of share elements whose cardinality is $d_\mathrm{j} = \|T_\mathrm{j}\|$.

Phase 2: Computing the public key $g^{sk}$:

1. Each $P_\mathrm{i}, \mathrm{i} \in [n]$ broadcasts the values $A_{\mathrm{i},1} = g^{\rho_{\mathrm{i},1}}$ and a NIZKPoK $\pi_\mathrm{i}$ proving that the value committed $z_\mathrm{i} = \rho_{\mathrm{i},1}$ is same value in both $A_{\mathrm{i},1}, C_{\mathrm{i},1}$ broadcast earlier to every other party $P_\mathrm{j}$, $\mathrm{j} \in [n]$.

2. Each party verifies the broadcast NIZKPoK of every other party and anyone failing verification is disqualified and removed from $\mathcal{Q}$.

3. Finally the public key is computed as $pk = \prod_{\mathrm{i} \in \mathcal{Q}} g^{\rho_{\mathrm{i},1}}$.



**Figure 6.3.** BBSS-DKG Protocol

subset of $t+1$ or more nodes constitutes the qualified subset. At the heart of any DKG is a verifiable secret sharing (VSS) scheme. To achieve a $(n,t)$-DKG protocol, we consider a $(n,t)$-VSS scheme; unlike a VSS scheme which requires a trusted dealer, the DKG mechanism distributes the trust among the nodes removing the requirement of a trusted party. In this work, we consider a DKG protocol resistant to $f$ malicious nodes with the total number of nodes $n = 3f+1$ in the network.

Using the verifiable BBSS scheme (refer 6.2), we obtain a DKG on the lines of the scheme by Gennaro *et al.* [30]. The protocol proceeds in two phases, in phase 1, each party $P_{\mathrm{i}}$ performs a verifiable secret sharing of a random value $z_{\mathrm{i}}$ and every party verifies the received shares using the broadcast commitments. After this, every party $P_{\mathrm{j}}$ forms the qualified set of parties $\mathcal{Q}$ whose shares are verified and compute its share $\boldsymbol{sk}_{\mathrm{j}}$ by locally adding the verified shares. The computed shares correspond to shares of a random secret key $sk \in \mathbb{Z}_q$. In Phase 2, the parties of the qualified set forward the exponentiation of their shared secret $z_{\mathrm{i}}$ and a zero-knowledge proof that the forwarded Pedersen commitment in Phase 1 corresponds to the same. Every party computes the public key $pk = g^{sk}$ after verifying the zero-knowledge proofs. The complete DKG protocol based on BBSS sharing is described in Figure 6.3.

The proposed DKG mechanism offers the following properties:

- *Correctness:* All qualified subsets of shares provided by honest parties define the same unique secret key $sk$; All honest parties compute the same public key $pk = g^{sk}$ value corresponding to the secret key $sk$
- *Secrecy:* No information on $sk$ can be obtained by the $t-$limited adversary except what can be inferred from the public information.

**Theorem 6.3.1.** *Given a correct and secure $(n,t)$-verifiable BBSS scheme, the DKG protocol of Figure 6.3 satisfies correctness and secrecy properties under the Dlog assumption (Definition 6.1.1)*

*Proof. Proof.* **Correctness.** In Phase 1 of the BBSS-DKG protocol from Figure 6.3, all honest parties compute the same qualified set $\mathcal{Q}$ as the complaint and disqualification information is broadcast to all parties. Any party $P_{\mathrm{i}} \in \mathcal{Q}$, which shared its value $z_{\mathrm{i}}$ successfully and any set $\mathcal{T}$ of $t+1$ or more honest parties can reconstruct the secret key

value, owing to the threshold structure of the BBSS performed. Let $\mathcal{R} = \bigcup_{\mathrm{i}} T_{\mathrm{i}}, \mathrm{i} \in \mathcal{T}$ be the set of all row indices of $\boldsymbol{M}$ held by the parties of $\mathcal{T}$. Each $z_{\mathrm{i}} = \sum_{k \in \mathcal{R}} s_{\mathrm{i},k} \cdot \lambda_k$ $\boldsymbol{\lambda}_{\mathcal{T}} = \{\lambda_k, k \in \mathcal{R}\}$ such that $\boldsymbol{M}_{\mathcal{T}}^{\top} \cdot \boldsymbol{\lambda}_{\mathcal{T}} = \boldsymbol{\varepsilon}$ and $z_{\mathrm{i}} = \boldsymbol{s}_{\mathcal{T}}^{\top} \cdot \boldsymbol{\lambda}_{\mathcal{T}}$, where $\boldsymbol{s}_{\mathcal{T}}$ is the vector of all share elements held by all the parties in $\mathcal{T}$. Every honest party computes its share vector $\boldsymbol{sk}_{\mathrm{j}} = \{\hat{s}_k \| \hat{s}_k = \sum_{\mathrm{i} \in \mathcal{Q}} s_{\mathrm{i},k}, k \in T_{\mathrm{j}}\}$ element-wise for each $k$. Thus we have,

$$sk = \sum_{\mathrm{i} \in \mathcal{Q}} z_{\mathrm{i}} = \sum_{\mathrm{i} \in \mathcal{Q}} \Big( \sum_{k \in \mathcal{R}} s_{\mathrm{i},k} \cdot \lambda_k \Big)$$
$$\implies sk = \sum_{k \in \mathcal{R}} \lambda_k \cdot \Big( \sum_{\mathrm{i} \in \mathcal{Q}} s_{\mathrm{i},k} \Big) = \sum_{k \in \mathcal{R}} \lambda_k \cdot \hat{s}_k$$

This holds for any set qualified set $\mathcal{T}$ (and hence the corresponding set of rows $\mathcal{R}$), thus giving a unique $sk$ for all such sets with $t+1$ or more parties.

Also, each share element $\hat{s}_k, k \in T_{\mathrm{j}}$ of a party $P_{\mathrm{j}}$, can be computed and verified from the publicly available values $g^{s_{\mathrm{i},k}}$.

$$g^{\hat{s}_k} = g^{\sum_{\mathrm{i} \in \mathcal{Q}} s_{\mathrm{i},k}} = \prod_{\mathrm{i} \in \mathcal{Q}} g^{s_{\mathrm{i},k}} = \prod_{\mathrm{i} \in \mathcal{Q}} \Big( \prod_{l=1}^{\mathrm{e}} A_{\mathrm{i},l}^{m_{k,l}} \Big)$$

which is available from Phase 2 of the protocol of Figure 6.3. Thus each share (and share element) can be verified for correctness at the time of reconstruction.

The public key $pk = \prod_{\mathrm{i} \in \mathcal{Q}} g^{\rho_{\mathrm{i},1}}$ is computed from values broadcast in the protocol, hence the value can be obtained by all the honest parties. It remains to be shown that $pk = g^{sk}$ such that $sk = \sum_{\mathrm{i} \in \mathcal{Q}} z_{\mathrm{i}}$. For the parties against whom a complaint is generated, the value $z_{\mathrm{i}}$ is reconstructed publicly. For the other parties against whom there was no complaint, all their values $A_{\mathrm{i},l}, l \in [\mathrm{e}]$ have been verified using the verification step in Phase 2 of the protocol. Since all such parties constitute the qualified set $\mathcal{Q}$ which is computed by all the honest parties, the value $A_{\mathrm{i},1} = g^{\rho_{\mathrm{i},l}} = g^{z_{\mathrm{i}}}$. The value $pk$ is computed by honest parties as $pk = \prod_{\mathrm{i} \in \mathcal{Q}} g^{z_{\mathrm{i}}} = g^{\sum_{\mathrm{i} \in \mathcal{Q}} z_{\mathrm{i}}} = g^{sk}$. Hence all the honest parties compute the same public key $pk$ corresponding to $sk$. Also since the qualified set of parties $\mathcal{Q}$ computed in the phase 1 of the protocol consists of at least one honest party who shares the value $z_{\mathrm{i}}$ which is chosen randomly, the secret key $sk = \sum_{\mathrm{i} \in \mathcal{Q}} z_{\mathrm{i}}$ is uniformly random.

□

**Secrecy.** We provide a simulator $\mathcal{S}$ in Figure 6.4 on the lines of [30], [250] which simulates the adversary view of the BBSS-DKG protocol of Figure 6.3. With out loss of generality we assume that the set of parties $\mathcal{C} = \{P_1, \cdots, P_{t'}\}$ are corrupted and set of rest of the parties $\mathcal{H} = \{P_{t'+1}, \cdots, P_n\}$ are honest. The simulator controls all the honest parties $\mathcal{H}$ and performs all computations and communications with the corrupt parties on behalf of them.

The simulator follows the Phase 1 of the protocol as shown in Figure 6.3 and generates share vectors $\boldsymbol{s}_{\mathrm{i,j}}$ using random $\boldsymbol{\rho}_{\mathrm{i}}$ for $P_{\mathrm{i}} \in \mathcal{H}, P_{\mathrm{j}} \in \mathcal{C}$. Similarly it generates and forwards the vectors $\boldsymbol{s}'_{\mathrm{i,j}}$ using random $\boldsymbol{\rho}'_{\mathrm{i}}$. Simfollows the protocol including the computation of qualified set $\mathcal{Q}$. However, in the second phase of the protocol, it computes and broadcasts all the $A_{\mathrm{i},l}$ for all the honest parties except one party $P_n$. For the party $P_n$ it sets the secret value $A_{\mathrm{i},0}$ such that the public key obtained as $\prod_{\mathrm{i}\in\mathcal{Q}} A_{\mathrm{i},l}, l \in [\mathrm{e}]$ is the desired value $y$. The simulator $\mathcal{S}$ will be able to reconstruct the vector $\boldsymbol{\rho}_k$ for any party $P_k$ which is present in the qualified set $\mathcal{Q}$ but not in the set $\mathcal{H}$. Whenever a valid complaint is broadcast from any party controlled by adversary, Sim constructs the secret value and opens it. □

## 6.4 D-KODE Protocol

By D-KODE protocol we refer to set of all algorithms for generating client keys in a distributed fashion. These algorithms include generation of shares of master key $\mathsf{k}$ at the servers using BBSS-DKG, PRF evaluation upon user input and algorithms to combine the partial evaluations to compute keys at the client. Since BBSS-DKG and PRF are run on the server, we refer to them as server-side algorithms and the algorithms for combining the partial evaluations for computing keys at the client as client-side algorithms. On the client side, we have two different versions corresponding to offline or online client. Offline client refers to the one to whose public key the payment has been made and wishes to retrieve the funds by generating the corresponding secret key. Online client computes the private key of self and public key of another client to process payment etc. For the ease of exposition, we present the verifiability of the PRF evaluation as a separate subsection. The D-KODE protocol consists of following algorithms.

**Simulator $\mathcal{S}$**

Let $\mathcal{C} = \{P_{\mathrm{i}}, \mathrm{i} \in \{1, \cdots, t'\}\}$ denote the parties controlled by the adversary and $\mathcal{H} = \{P_{\mathrm{j}}, \mathrm{j} \in \{t'+1, \cdots, n\}\}$ denote the set of honest parties in the protocol. $t' \leq t$. Simtakes the public key $y$ as input.

1. The simulator $\mathcal{S}$ performs all the steps in the Phase 1 of the BBSS-DKG on behalf of the parties of set $\mathcal{H}$ including generating and forwarding shares and commitments, verifications of the received shares and handling all communications with the corrupted parties such that the following hold:
   (a) The values $\boldsymbol{\rho}_{\mathrm{i}}, \boldsymbol{\rho}'_{\mathrm{i}}$ for $P_{\mathrm{i}} \in \mathcal{H}$ are chosen at random by $\mathcal{S}$.
   (b) The set $\mathcal{Q}$ is well defined with $\mathcal{H} \subset \mathcal{Q}$
   (c) The adversary's view consists of $(\boldsymbol{\rho}_{\mathrm{j}}, \boldsymbol{\rho}'_{\mathrm{j}})$ for $P_{\mathrm{j}} \in \mathcal{C}$, shares $(\boldsymbol{s}_{\mathrm{i,j}}, \boldsymbol{s}'_{\mathrm{i,j}})$ for $P_{\mathrm{i}} \in \mathcal{Q}$ and $P_{\mathrm{j}} \in \mathcal{C}$ and commitments $C_{\mathrm{i},k}, P_{\mathrm{i}} \in \mathcal{Q}, k \in [t]$
   (d) $\mathcal{S}$ has all shares and commitments of the parties in $\mathcal{Q}$. For $\mathrm{j} \in \mathcal{Q} \backslash \mathcal{H}$, Sim has enough valid shares to reconstruct the vector $\boldsymbol{\rho}_{\mathrm{j}}, \boldsymbol{\rho}'_{\mathrm{j}}$.
2. Perform:
   (a) Compute $A_{\mathrm{i},l}, l \in [\mathrm{e}] = g^{\rho_{\mathrm{i},l}}$ for $\mathrm{i} \in \mathcal{Q} \backslash n, l \in [\mathrm{e}]$
   (b) Set $A^*_{n,0} = y \prod_{\mathrm{i} \in \mathcal{Q} \backslash n} (A_{\mathrm{i},0})^{-1}$ and $\boldsymbol{s}^*_{n,k} = \boldsymbol{s}_{n,k} = \{s_{n,k}, k \in T_n\}$ where $s_{n,l}, l \in [\mathrm{e}]$ is an element of the vector $M \cdot \boldsymbol{\rho}_n$ item Broadcast the values $A_{\mathrm{i},l}$ for $\mathrm{i} \in \mathcal{H} \backslash n$ and $A^*_{n,l}$ with $l \in [\mathrm{e}]$ along with the corresponding NIZKPoK $\pi_{\mathrm{i}}$

**Figure 6.4.** Simulator for BBSS-DKG

### 6.4.1 Server Side Algorithms

**Cryptographic Setup.** $\mathsf{Setup}(\lambda, n, t)$: It takes as input the security parameter $\lambda$, the threshold $t$ and the number of servers $n$ and outputs the public parameters $\mathsf{pp} := \{H(\cdot), p, q, q', u, \mathbb{G}, g, \mathsf{G}, \mathsf{g}, \mathsf{h}, \boldsymbol{M}, \psi(\cdot)\}$.

**Distributed Key Generation.** $\mathsf{DKG}(n, t, q, u)$: The servers run the BBSS-DKG mechanism among themselves using $(n, t)$-BBSS to generate shares of a master key $\mathsf{k} \in \mathbb{Z}_q^u$.

The BBSS-DKG mechanism presented in Section 6.3 (Figure 6.3) provides shares corresponding to a *single element* $sk \in \mathbb{Z}_q$ to all the servers. However, for the PRF evaluation, $F(X, \mathsf{k}) = \lfloor H(X) \cdot \mathsf{k} \rfloor_p$ introduced in Section 2.2, the key $\mathsf{k}$ is a vector of length $u$. Hence,

**Algorithm 5** ParSecretKeyEval $(X, \boldsymbol{E}_{\mathrm{i}}, \mathsf{pp})$

Parse the matrix $\boldsymbol{E}_{\mathrm{i}}^{\top} \sim \mathbb{Z}_q^{u \times d_{\mathrm{i}}}$ as $[\boldsymbol{k}_{\mathrm{i},1} \| \boldsymbol{k}_{\mathrm{i},2} \| \cdots \| \boldsymbol{k}_{\mathrm{i},d_{\mathrm{i}}}]$
**for** $1 \leq \mathrm{j} \leq d_{\mathrm{i}}$ **do**
    $z_{\mathrm{i,j}} = \left\lfloor H(X) \cdot \boldsymbol{k}_{\mathrm{i,j}} \right\rfloor_p \in \mathbb{Z}_p$
**return** $\boldsymbol{z}_{\mathrm{i}} = \{z_{\mathrm{i},1}, z_{\mathrm{i},2}, \cdots, z_{\mathrm{i},d_{\mathrm{i}}}\} \in \mathbb{Z}_p^{d_{\mathrm{i}}}$

**Algorithm 6** ParPubKeyEval$(X', \boldsymbol{E}_{\mathrm{i}}, \mathsf{pp})$

Parse the matrix $\boldsymbol{E}_{\mathrm{i}}^{\top} \sim \mathbb{Z}_q^{u \times d_{\mathrm{i}}}$ as $[\boldsymbol{k}_{\mathrm{i},1} \| \boldsymbol{k}_{\mathrm{i},2} \| \cdots \| \boldsymbol{k}_{\mathrm{i},d_{\mathrm{i}}}]$
**for** $1 \leq \mathrm{j} \leq d_{\mathrm{i}}$ **do**
    $z_{\mathrm{i,j}} = \left\lfloor H(X') \cdot \boldsymbol{k}_{\mathrm{i,j}} \right\rfloor_p \in \mathbb{Z}_p$
**return** $\boldsymbol{y}_{\mathrm{i}} = \{g^{z_{\mathrm{i},1}}, g^{z_{\mathrm{i},2}}, \cdots, g^{z_{\mathrm{i},d_{\mathrm{i}}}}\} \in \mathbb{G}^{d_{\mathrm{i}}}$

initially, the servers run $u$ instances of DKG to generate shares of elements of vector in $\mathbb{Z}_q^u$. Let the share element matrix obtained by each server $P_{\mathrm{i}}$ be $\boldsymbol{E}_{\mathrm{i}}$.

**PRF evaluation.** The servers run the PRF service through the ParSecretKeyEval and ParPubKeyEval algorithms to compute private key or public key shares respectively for an identity forwarded by the client.

ParSecretKeyEval$(X, \boldsymbol{E}_{\mathrm{i}}, \mathsf{pp})$: Sever $P_{\mathrm{i}}$ takes the client input string $X$, share matrix $\boldsymbol{E}_{\mathrm{i}}$, the public parameters pp and returns the evaluation of the PRF as the vector $\boldsymbol{z}_{\mathrm{i}}$. The matrix $\boldsymbol{E}_{\mathrm{i}}^{\top}$ is parsed into $d_{\mathrm{i}}$ columns of $u$ length each while input $X$ is hashed to a vector of length $u$ using the hash $H : \{0,1\}^* \rightarrow \mathbb{Z}_q^u$. $d_{\mathrm{i}}$ is the number of rows of matrix $\boldsymbol{M}$ owned by $P_{\mathrm{i}}$. ParSecretKeyEval is shown in Algorithm 5.

ParPubKeyEval$(X', \boldsymbol{E}_{\mathrm{i}}, \mathsf{pp})$: Partial evaluation for public key generation is similar to that of the secret key except that the final vector is the exponentiated version of partial secret key evaluation. Server $P_{\mathrm{i}}$ takes the client input string $X'$, share matrix $\boldsymbol{E}_{\mathrm{i}}$, the public parameters pp and returns a vector $\boldsymbol{y}_{\mathrm{i}}$. The matrix $\boldsymbol{E}_{\mathrm{i}}^{\top}$ is parsed into $d_{\mathrm{i}}$ columns of $u$ length each while input $X$ is hashed to a vector of length $u$ using $H : \{0,1\}^* \rightarrow \mathbb{Z}_q^u$. $d_{\mathrm{i}}$ is the number of rows of matrix $\boldsymbol{M}$ owned by $P_{\mathrm{i}}$. Each of the elements of the PRF evaluation is exponentiated resulting in a vector of elements of group $\mathbb{G}$ and of length $d_{\mathrm{i}}$. ParPubKeyEval is shown in Algorithm 6.

### 6.4.2 Client Side Algorithms

The client computes the private key by combining the partial evaluations using the CombSecKey algorithm and computes the public key of identity $X'$ by using the CombPubKey algorithm. The offline client after generating private key of his identity searches for the appropriate secret key - public key pair to which payment has been made.

**Private key generation.** CombSecKey(pp, $\{\boldsymbol{z}_1, \boldsymbol{z}_2, \cdots, \boldsymbol{z}_{\|\mathcal{T}\|}\}$): Let $\mathcal{T}$ with $\|\mathcal{T}\| \geq t+1$ be the set of parties whose evaluations are used for reconstruction. CombSecKey() takes-in the partial evaluation vectors $\boldsymbol{z}_{\mathrm{i}}$ received from the servers $P_{\mathrm{i}}$ of the set $\mathcal{T}$ and concatenates them to form $\boldsymbol{Z} = \{\boldsymbol{z}_1 \| \| \boldsymbol{z}_2 \| \| \cdots \| \| \boldsymbol{z}_{\|\mathcal{T}\|}\}$ . Let the set of all the row indices of matrix $\boldsymbol{M}$ held by the parties in $\mathcal{T}$ be $\mathcal{R} = \bigcup_{\mathrm{i}} T_{\mathrm{i}}, P_{\mathrm{i}} \in \mathcal{T}$. $\boldsymbol{Z}$ is a vector of length $\|\mathcal{R}\|$. The private key is computed as the linear combination of the vector elements. The reconstruction coefficient vector $\lambda_{\mathcal{T}}$ is computed by solving $\boldsymbol{M}_{\mathcal{T}}^{\top} \cdot \boldsymbol{\lambda}_{\boldsymbol{\mathcal{T}}} = \boldsymbol{\varepsilon}$. $\boldsymbol{M}_{\mathcal{T}}^{\top}$ is the set of all rows of matrix $\boldsymbol{M}$ held by the parties in $\mathcal{T}$. $\boldsymbol{\varepsilon} = \{1, 0, \cdots, 0\}$.

*Online client* : The online client computes the private key $sk$ and the corresponding public key as $pk = g^{sk}$ and uses the key-pair $(sk, pk)$ to perform different transactions as needed.

*Offline client*: Once the offline client computes the private key $sk$ corresponding to his identity, he computes $2\|\mathcal{T}\|$ secret keys as $[sk - \|\mathcal{T}\|, \cdots, sk + \|\mathcal{T}\|]$ and obtains the corresponding public keys $[g^{sk-\|\mathcal{T}\|}, \cdots, g^{sk+\|\mathcal{T}\|}]$. He then queries the blockchain based system for the $2\|\mathcal{T}\|$ public keys (addresses) and uses the secret key $sk'$ corresponding to the public key to which funds have been transferred. CombSecKey is shown in the Algorithm 7.

**Public key generation.** CombPubKey(pp, $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_{\|\mathcal{T}\|}\}$): Let $\mathcal{T}$ with $\|\mathcal{T}\| \geq t+1$ be the set of servers whose evaluations are used for reconstruction. CombPubKey takes-in the vector of partial evaluations $\boldsymbol{y}_{\mathrm{i}}$ received from the servers $P_{\mathrm{i}}$ of the set $\mathcal{T}$ and concatenates them to form $\boldsymbol{Y} = \{\boldsymbol{y}_1 \| \| \boldsymbol{y}_2 \| \| \cdots \| \| \boldsymbol{y}_{\|\mathcal{T}\|}\}$. The set of all the row indices (of matrix $\boldsymbol{M}$) held by the parties in $\mathcal{T}$ is $\mathcal{R} = \bigcup_{\mathrm{i}} T_{\mathrm{i}}, P_{\mathrm{i}} \in \mathcal{T}$. $\boldsymbol{Z}$ is a vector of length $\|\mathcal{R}\|$. Compute the public key as $pk = \prod_{1 \leq \mathrm{j} \leq \|\mathcal{R}\|} y_{\mathrm{j}}^{\lambda_{\mathrm{j}}}$, where $\boldsymbol{M}_{\mathcal{T}}^{\top} \cdot \boldsymbol{\lambda}_{\boldsymbol{\mathcal{T}}} = \boldsymbol{\varepsilon}$, $\boldsymbol{M}_{\mathcal{T}}^{\top}$ is the set of all rows of matrix $\boldsymbol{M}$ held by the parties in $\mathcal{T}$, $\boldsymbol{\lambda}_{\mathcal{T}} = \{\lambda_{\mathrm{j}}, 1 \leq \mathrm{j} \leq \|\mathcal{R}\|\}$, $\boldsymbol{Y} = \{y_{\mathrm{j}}, 1 \leq \mathrm{j} \leq \|\mathcal{R}\|\}$.

**Algorithm 7** CombSecKey $(\mathsf{pp}, \{\boldsymbol{z}_1, \boldsymbol{z}_2, \cdots, \boldsymbol{z}_{\|\mathcal{T}\|}\})$

Compute $\boldsymbol{Z} = \{\boldsymbol{z}_1 \| \|\boldsymbol{z}_2\| \| \cdots \| \|\boldsymbol{z}_{\|\mathcal{T}\|}\} \in \mathbb{Z}_p^{\|\mathcal{R}\|}$
Compute $\boldsymbol{\lambda}_{\boldsymbol{\mathcal{T}}} \in \{-1, 0, 1\}^{\|\mathcal{R}\|}$ such that $\boldsymbol{M}_{\mathcal{T}}^{\top} \cdot \boldsymbol{\lambda}_{\boldsymbol{\mathcal{T}}} = \boldsymbol{\varepsilon}$
Compute $sk = \boldsymbol{\lambda}_{\mathcal{T}}^{\top} \cdot \boldsymbol{Z} \in \mathbb{Z}_p$
**if** Online client **then**
    **return** $sk$
**if** Offline client **then**
    Compute $[sk - \|\mathcal{T}\|, \cdots, sk + \|\mathcal{T}\|]$
    Compute public keys $\vec{pk} = [g^{sk-\|\mathcal{T}\|}, \cdots, g^{sk+\|\mathcal{T}\|}]$
    Query public keys $\vec{pk}$ and find corresponding $sk'$
    **return** $sk'$

**Algorithm 8** CombPubKey $(\mathsf{pp}, \{\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_{\|\mathcal{T}\|}\})$

Compute $\boldsymbol{Y} = \{\boldsymbol{y}_1 \| \|\boldsymbol{y}_2\| \| \cdots \| \|\boldsymbol{y}_{\|\mathcal{T}\|}\} \in \mathbb{G}^{\|\mathcal{R}\|}$
Compute $\boldsymbol{\lambda}_{\boldsymbol{\mathcal{T}}} \in \{-1, 0, 1\}^{\|\mathcal{R}\|}$ such that $\boldsymbol{M}_{\mathcal{T}}^{\top} \cdot \boldsymbol{\lambda}_{\boldsymbol{\mathcal{T}}} = \boldsymbol{\varepsilon}$
$\boldsymbol{\lambda}_{\mathcal{T}} = \{\lambda_{\mathsf{j}}\}$, $\boldsymbol{Y} = \{y_{\mathsf{j}}\}, 1 \leq \mathsf{j} \leq \|\mathcal{R}\|$
Compute $pk = \prod_{1 \leq \mathsf{j} \leq \|\mathcal{R}\|} y_{\mathsf{j}}^{\lambda_{\mathsf{j}}} \in \mathbb{G}$
**return** $pk$

Any client can forward the public identity of another client and compute the public key from the obtained partial evaluations using CombPubKey which shown as Algorithm 8.

### 6.4.3 Verifying the evaluation of the PRF

While the clients obtain shares as the PRF evaluations presented in Section 6.4.1, it is imperative for the clients to verify if the values received were generated correctly. The servers after evaluating the PRF, forward a commitment and a zero-knowledge proof proving that the values have been computed according to the protocol. For ease of exposition, we present here the verifiability for *one* PRF evaluation.

The PRF function employed by D-KODE protocol is

$F(X, \boldsymbol{K}) = \left\lfloor H(X) \cdot \boldsymbol{K} \right\rfloor_p \in \mathbb{Z}_p$ with $H : \mathcal{X} \to \mathbb{Z}_q^u$, $\boldsymbol{K} \in \mathbb{Z}_q^u$, $F : \mathcal{X} \times \mathbb{Z}_q^u \to \mathbb{Z}_p$ and $p < q$. Let $\boldsymbol{K} = \{\alpha_1, \alpha_2, \cdots, \alpha_u\}$.

**Verification of the private key evaluation.** Let $z = F(X, \mathsf{k})$ for $\mathsf{k}$ defined as above. To compute $z$, the servers compute the inner product $\hat{z} = (H(X) \cdot \mathsf{k}) \in \mathbb{Z}_q$ and perform the

operation $z = \lfloor \hat{z} \rfloor_p \in \mathbb{Z}_p$. Hence we have, $z = \left\lfloor \hat{z} \cdot \frac{p}{q} \right\rfloor \implies p\hat{z} = zq + r$ where the value $r < q$. To provide verifiability, it is enough for the server to prove that the above equation has been evaluated correctly and that the value $r < q$. The server uses commitments and zero-knowledge range proof to do the same.

*Using Pedersen commitments*[251]: For a key $\boldsymbol{K} = \{\alpha_1, \alpha_2, \cdots, \alpha_u\}$, the server initially publishes the commitments $c_\mathsf{i} = \mathsf{g}^{\alpha_\mathsf{i}}\mathsf{h}^{\beta_\mathsf{i}}, \mathsf{i} \in [u]$, $\mathsf{g}, \mathsf{h} \in \mathsf{G}$ are generators of multiplicative group of order $q' > pq$. While forwarding the value $z = F(X, \mathsf{k})$, the server forwards the value $z'$ and commitment to the value $r$, $c = \mathsf{g}^r\mathsf{h}^{r'}$ such that $p\hat{z} = zq + r; \quad p\hat{z}' = z'q + r'$ where $z' = F(X, \boldsymbol{K}'), \boldsymbol{K}' = \{\beta_1, \beta_2, \cdots, \beta_n\}$.

The server also provides zero-knowledge range proof $\boldsymbol{\pi}_r$ [252] proving that $r < q$. Thus when evaluating the PRF for an input $X$, the server replies with the following: $\{z, z', c, \boldsymbol{\pi}_r\}$ As the server publishes the $c_\mathsf{i}$ values, they are available to the client before the evaluation.

*Verification*: After verifying the zero-knowledge range proof [252], the client verifies the obtained value $z$ by checking if $\prod_{\mathsf{i}=1}^{u}(c_\mathsf{i})^{h_\mathsf{i}p} = (\mathsf{g}^z \cdot \mathsf{h}^{z'})^q \cdot c$ where $H(X) = \{h_1, h_2, \cdots, h_n\}$. We have, $\prod_{\mathsf{i}=1}^{u}(c_\mathsf{i})^{h_\mathsf{i}p} = \prod_{\mathsf{i}=1}^{u}\left(\mathsf{g}^{\alpha_\mathsf{i}}\mathsf{h}^{\beta_\mathsf{i}}\right)^{h_\mathsf{i}p} = \mathsf{g}^{\sum_\mathsf{i}\alpha_\mathsf{i}h_\mathsf{i}\cdot p}\mathsf{h}^{\sum_\mathsf{i}\beta_\mathsf{i}h_\mathsf{i}\cdot p} = \mathsf{g}^{\hat{z}p}\mathsf{h}^{\hat{z}'p}$
$= \mathsf{g}^{zq+r}\mathsf{h}^{z'q+r'} = \left(\mathsf{g}^z\mathsf{h}^{z'}\right)^q \cdot \mathsf{g}^r\mathsf{h}^{r'} = \left(\mathsf{g}^z\mathsf{h}^{z'}\right)^q \cdot c$

**Verification of the public key evaluation.** When the client requests for the public key corresponding to identity $X$, the server computes and forwards the value $g^z, z = F(X, \mathsf{k})$, for $g \in \mathbb{G}$ a generator of a multiplicative group of order $p$.

For verifiability, the server forwards values $\mathsf{g}^z$ and commitments $\mathsf{g}^z\mathsf{h}^{z'}$, $\mathsf{g}, \mathsf{h} \in \mathsf{G}$ are generators of multiplicative group of order $q' > pq$. The server also forwards a commitment $c = \mathsf{g}^r\mathsf{h}^{r'}$ such that $p\hat{z} = zq + r; \quad p\hat{z}' = z'q + r'$ where $\hat{z}' = F(X, \boldsymbol{K}'), \boldsymbol{K}' = \{\beta_1, \beta_2, \cdots, \beta_n\}$. The commitments $c_\mathsf{i} = \mathsf{g}^{\alpha_\mathsf{i}}\mathsf{h}^{\beta_\mathsf{i}}, \mathsf{i} \in [u]$ are published by the server earlier.

The server also forwards the value $z'$, two zero-knowledge proofs, one $\boldsymbol{\pi}_{\mathsf{Equ}}(g^z, \mathsf{g}^z\mathsf{h}^{z'})$ proving that the value $z$ in both the exponents $(g^z, \mathsf{g}^z\mathsf{h}^{z'})$ is equal for the zero knowledge proof used) and $\boldsymbol{\pi}_r$, proving that the value $r < q$. Thus the server forwards the values $\{g^z, \mathsf{g}^z\mathsf{h}^{z'}, c, \boldsymbol{\pi}_r, \boldsymbol{\pi}_{\mathsf{Equ}}(g^z, \mathsf{g}^z\mathsf{h}^{z'})\}$. After verifying the zero knowledge proofs, the client verifies if: $\prod_{\mathsf{i}=1}^{u}(c_\mathsf{i})^{h_\mathsf{i}p} = (\mathsf{g}^z \cdot \mathsf{h}^{z'})^q \cdot c$

**Theorem 6.4.1.** *If the* $\mathsf{LWR}_{(q,m,n)}$ *assumption holds,* $\mathsf{ParSecretKeyEval}(X, \boldsymbol{E}, pp)$ *is a pseudo-random function.*

*Proof.* Let $\mathsf{ParSecretKeyEval}(X, \boldsymbol{E}, \mathsf{pp})$ be $f_{\boldsymbol{E}}(X)$, we show that $f_{\boldsymbol{E}}$ is a family of pseudo-random functions. Let $\mathcal{D}$ be an efficient algorithm that gets the value of $f_{\boldsymbol{E}}$ on $\ell-1$ uniformly chosen inputs $X_1, X_2, \cdots, X_{\ell-1}$ and distinguishes $f_{\boldsymbol{E}}(X_\ell)$ from random with a non-negligible advantage $\epsilon$. We construct an algorithm $\mathcal{A}$ that breaks the $\mathsf{LWR}$ assumption:

On input $(\boldsymbol{A}, \lfloor \boldsymbol{A}\boldsymbol{s} \rfloor_p)$ where $\boldsymbol{A} \sim U(\mathbb{Z}_q^{\mathsf{m}\times\mathsf{n}}), \boldsymbol{s} \sim U(\mathbb{Z}_q^{\mathsf{n}})$. $\mathcal{A}$ parses the matrix $\boldsymbol{A}$ as rows $\boldsymbol{a}_1, \boldsymbol{a}_2, \cdots, \boldsymbol{a}_{\mathsf{m}}$ and vector $\lfloor \boldsymbol{A}\boldsymbol{s} \rfloor_p$ as $\boldsymbol{z'}_1, \boldsymbol{z'}_2, \cdots, \boldsymbol{z'}_{\mathsf{m}}$. For each $\boldsymbol{z'}_{\mathrm{i}}, \mathrm{i} \leq \mathsf{m}$, sample $d-1$ uniformly random values $s_{\mathrm{i},2}, s_{\mathrm{i},3}, \cdots s_{\mathrm{i},d} \in \mathbb{Z}_p$. Let $\boldsymbol{z}_{\mathrm{i,j}} = \boldsymbol{a}_{\mathrm{i}} \cdot s_{\mathrm{i,j}}$ for $\mathrm{i} \leq \mathsf{m}; 2 \leq \mathrm{j} \leq d$. Now $\mathcal{A}$ invokes $\mathsf{m}$ instances of algorithm $\mathcal{D}_{\mathrm{i}}$ each with the $\ell-1$ pairs of values $\{\langle H(X_{\mathrm{j}}), f_{\boldsymbol{E}}(X_{\mathrm{j}})\rangle\}_{\mathrm{j}=1}^{\ell-1}$ and a pair $\langle a_{\mathrm{i}}, [\boldsymbol{z'}_{\mathrm{i}}, \boldsymbol{z}_{\mathrm{i},2}, \boldsymbol{z}_{\mathrm{i},3}, \cdots, \boldsymbol{z}_{\mathrm{i},d}]\rangle$ for $\mathrm{i} \leq \mathsf{m}$. $\mathcal{D}_{\mathrm{i}}$ distinguishes $[\boldsymbol{z'}_{\mathrm{i}}, \boldsymbol{z}_{\mathrm{i},2}, \boldsymbol{z}_{\mathrm{i},3}, \cdots, \boldsymbol{z}_{\mathrm{i},d}]$ from a uniformly random vector with advantage $\epsilon$. Algorithm $\mathcal{A}$ distinguishes the $\mathsf{LWR}$ instance from a uniformly random vector $U(\mathbb{Z}_q^d)$ with an advantage at-least $\epsilon$. □

**Theorem 6.4.2.** *If the* $\mathsf{LWR}_{(q,m,n)}$ *assumption holds,* $\mathsf{CombSecKey}$ *is a* $(n,t)$*-threshold evaluation of a pseudo-random function.*

*Proof.* Let $\mathcal{D}'$ be an efficient algorithm that differentiates an evaluation of $\mathsf{CombSecKey}$ from a uniformly random vector with a non-negligible advantage $\epsilon$ after $\ell-1$ queries. It takes the vectors $[\boldsymbol{z}_1, \boldsymbol{z}_2, \cdots, \boldsymbol{z}_n]$, computes $\lambda_{\mathrm{i}} \cdot \boldsymbol{z}_{\mathrm{i}}$ such that the elements of the vector $\lambda_{\mathrm{i}} \in \{-1, 0, 1\}$ and differentiates the resultant vector $\boldsymbol{sk}$ from the uniform vector $U(\mathbb{Z}_q^n)$ with an advantage $\epsilon$.

We first consider the case when all the $n$ servers are honest and then consider the case when $t$ of them are corrupt. We build an algorithm $\mathcal{A}'$ with uses $\mathcal{D}'$ to solve the $\mathsf{LWR}$ instance. On input $(\boldsymbol{A}, \lfloor \boldsymbol{A}\boldsymbol{s} \rfloor_p)$ where $\boldsymbol{A} \sim U(\mathbb{Z}_q^{\mathsf{m}\times\mathsf{n}}), \boldsymbol{s} \sim U(\mathbb{Z}_q^n)$. $\mathcal{A}$ parses the matrix $\boldsymbol{A}$ as rows $\boldsymbol{a}_1, \boldsymbol{a}_2, \cdots, \boldsymbol{a}_{\mathsf{m}}$ and vector $\lfloor \boldsymbol{A}\boldsymbol{s} \rfloor_p$ as $\boldsymbol{z}'_1, \boldsymbol{z}'_2, \cdots, \boldsymbol{z}'_{\mathsf{m}}$. For each $\boldsymbol{z}'_{\mathrm{i}}, \mathrm{i} \leq \mathsf{m}$, sample $d-1$ uniformly random values $s_{\mathrm{i},2}, s_{\mathrm{i},3}, \cdots, s_{\mathrm{i},d} \in \mathbb{Z}_p$. Let $\boldsymbol{z}_{\mathrm{i,j}} = \boldsymbol{a}_{\mathrm{i}} \cdot s_{\mathrm{i,j}}$ for $\mathrm{i} \leq \mathsf{m}; 2 \leq \mathrm{j} \leq d_{\mathrm{i}}$, $Z_{\mathrm{i}} = [\boldsymbol{z}'_{\mathrm{i}}, \boldsymbol{z}_{\mathrm{i},2}, \boldsymbol{z}_{\mathrm{i},3}, \cdots, \boldsymbol{z}_{\mathrm{i},d_{\mathrm{i}}}]$. Now $\mathcal{A}'$ invokes j instances of algorithm $\mathcal{D}'$ each with $\ell-1$ vectors $\hat{Z}_{\mathrm{i,j}}, \mathrm{i} \leq \ell-1$ and an additional input a vector $Z'_{\mathrm{j}} = [Z_{\mathrm{j}}, Z_{\mathrm{j}+1}, \cdots Z_{\mathrm{j}+n}]$ for $1 \leq \mathrm{j} \leq \lceil \frac{\mathsf{m}}{n} \rceil$. Each instance of $\mathcal{D}'$ distinguishes the input vector from uniformly random vector $U(\mathbb{Z}_p^n)$ with an

advantage $\epsilon$, thus algorithm $\mathcal{A}'$ distinguishes an $\mathsf{LWR}$ instance from a random vector with an advantage at-least $\epsilon$.

In the case where $t'$ servers are corrupt, the adversary has access to the secret key shares of the $t'$ servers. In such a case, the algorithm $\mathcal{A}'$ supplies only $n-t$ element vectors to each instance of the algorithm $\mathcal{D}'$ through the vector $[Z_{\mathrm{j}}, Z_{\mathrm{j}+1}, \cdots Z_{\mathrm{j}+n-t}]$. Each $\mathcal{D}'$ simulates the $t$ servers by sampling $t$ values $Z_{\mathrm{j}+n-t}, \cdots, Z_{\mathrm{j}+n} \in \mathbb{Z}_p^{d_{\mathrm{i}}}$. It constructs the vector $Z'_{\mathrm{j}} = [Z_{\mathrm{j}}, Z_{\mathrm{j}+1}, \cdots, Z_{\mathrm{j}+n}]$, computes $\boldsymbol{sk}_{\mathrm{j}} = \lambda_{\mathrm{i}} \cdot Z_{\mathrm{j}}$ for each element of $\lambda_{\mathrm{i}} \in \{-1, 0, 1\}$. The algorithm $\mathcal{D}'$ differentiates the vector from uniform random vector with an advantage $\epsilon$. The algorithm $\mathcal{A}'$ differentiates the $\mathsf{LWR}$ instance from random vector with an advantage of at-least $\epsilon$.

□

### 6.5 Distribution Matrix from Threshold Function

To generate the distribution matrix $\boldsymbol{M}$ for a $(n,t)$-threshold BBSS scheme used in the DKG mechanism, we realize the $(n,t)$ threshold access structure as a *threshold circuit* of *sufficient* depth. We convert the monotone boolean function representation of the circuit to the distribution matrix using the Benaloh-Leichter (BL) [93], [94] construction. Much of the previous works [93], [110], [112] suggest realizing the threshold access structure using a majority function [110]. Valiant [110] first proved that a polynomial size monotone circuit is realizable for majority circuit and provided a construction of size $O(n^{5.3})$, while the work by Hooray *et al.* [111] further improved the size of the circuit to $O(n^{1+\sqrt{2}})$. Valiant[110] suggested realizing threshold function using majority circuit of $2n$ variables [2] which was adapted by other works like Damgard *et al.* [93] following similar approach. Also, the proposed constructions [110], [111] are probabilistic in nature and depth of the circuits is such that the probability with which the circuits outputs 1, on majority in the $n$ input variables, is $1-\mathsf{e}$ where $\mathsf{e} = 2^{-n}$. In this work, instead of realizing threshold circuit using majority circuit, we compute the required threshold circuit directly and also report that choosing $\mathsf{e} = 2^{-n}$ is indeed an overkill increasing the depth of circuit. Larger $\mathsf{e} > 2^{-n}$ is

[2] ↑For $(n,t)$ threshold function, take $n$ extra variables (total $2n$ variables), fix $n-t$ of them to be 1 and the rest $t$ to 0; whenever there are more than $t$ 1s in the original $n$ variables, the majority function outputs 1.

**Table 6.1.** $m$ values obtained through threshold circuit for different $n, \mathtt{p}$ values and error margins

| $n$ | $\mathtt{e} = 2^{-n}$ | | $\mathtt{e} = 2^{-\frac{n}{4}}$ | |
|---|---|---|---|---|
| | $\mathtt{p} = 0.5$ | $\mathtt{p} = 0.66$ | $\mathtt{p} = 0.5$ | $\mathtt{p} = 0.66$ |
| 5 | 81 | 9 | 9 | 9 |
| 10 | 2187 | 81 | 81 | 27 |
| 20 | 59049 | 729 | 2187 | 27 |
| 30 | 177147 | 2187 | 19683 | 81 |

**Table 6.2.** Dimensions of Distribution matrix $\boldsymbol{M}$ for different $m$

| $m$ | Rows | Columns |
|---|---|---|
| 3 | 6 | 4 |
| 9 | 36 | 22 |
| 27 | 216 | 130 |
| 81 | 1296 | 778 |
| 243 | 7776 | 4666 |

sufficient to realize the required access structure in the practical system profiles considered. Essentially, we relax the error to achieve efficient implementation while still reconstructing the secret for all the qualified sets of the access structure.

We adapt the construction provided by Goldreich [113] for the majority circuit construction that uses a MAJ3 probability amplifier node[3] (Refer 2.5 for a brief description of Goldreich's [113] construction and analysis of the majority circuit). The construction as depicted in Figure 6.5 consists of $n$ variables $x_{\mathrm{i}}, \mathrm{i} \in [n]$ and $m$ variables $y_{\mathrm{j}}, \mathrm{j} \in [m]$ are assigned as follows: choose random indices i uniformly between 1 and $n$ and assign the corresponding $x_{\mathrm{i}}$ to each $y_{\mathrm{j}}, \mathrm{j} \in [m]$ sequentially. Construct a 3-ary tree of MAJ3 nodes with $y_{\mathrm{j}}$ as leaves. The probability $\mathtt{p} = \Pr(y_{\mathrm{j}} = 1)$ is taken as 0.5 for designing a majority circuit. However, we choose the value of $\mathtt{p}$ as $\frac{t}{n}$ for the threshold access structure $(n, t)$, we also compute depth with $\mathtt{e}= 2^{-\frac{n}{4}}$. To see why this is significant, we first present how the dimensions of the distribution matrix $\boldsymbol{M}$ are related to the value $m$, the number of leaves in the circuit.

[3]↑The MAJ3 node realizes majority of 3 variables $(x_1, x_2, x_3)$ as $x_1x_2 + x_2x_3 + x_1x_3$

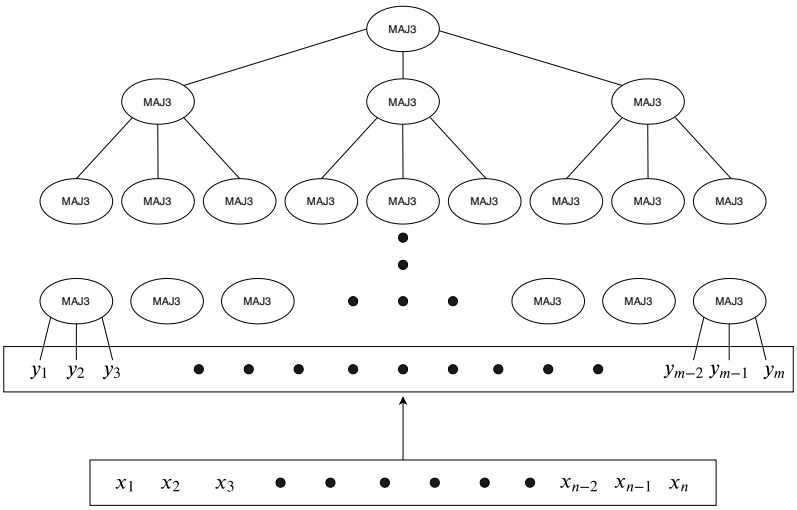



**Figure 6.5.** Majority circuit realization using MAJ3 nodes. The variables $x_{\mathrm{i}}, \mathrm{i} \leq n$ are mapped to $y_{\mathrm{j}}, \mathrm{j} \leq m$ uniformly randomly. MAJ3 tree is formed from $y_{\mathrm{j}}$.

Table 6.2 presents $m$ values and the dimensions of $\boldsymbol{M}$ when the circuit is constructed using MAJ3 nodes and the distribution matrix is constructed by BL construction [93], [94] from the monotone boolean formula representation of the circuit. With the above construction, the number of rows of matrix $\boldsymbol{M}$ grow as $6^{\log_3(m)}$. Table 6.1 depicts the value of $m$ needed to represent the threshold access structure for different values of $\mathtt{p}$ and $\mathtt{e}$. For instance, from Table 6.2 for $m = 243$, the number of rows of $\boldsymbol{M}$ is 7776. Observe from Table 6.1 that for $(n, \mathtt{p}, \mathtt{e}) = (20, 0.5, 2^{-n})$, the value $m = 59049$. For $m = 243$ itself, the number of rows is 7776, for $m = 59049$ the number of rows make it extremely difficult (almost impossible) to perform the secret sharing on a laptop or a phone using a majority circuit implementation ($\mathtt{p} = 0.5$) with $\mathtt{e} = 2^{-n}$. However, through implementation (by computing different threshold combinations) we find that $\mathtt{e} = 2^{-\frac{n}{4}}$ is indeed sufficient to successfully reconstruct the secret for the qualified sets for number of servers $n$ up to 50.

In this work we consider the $(n, \lfloor \frac{2n}{3} \rfloor)$ access structure and generate the matrix $\boldsymbol{M}$ with depth analysed using $\mathtt{p} = \frac{2}{3}$. It must be noted that the distribution matrix size is dependent on the computed $m$ value rather than directly on the value $n$ that is to say, multiple $n$ values may result in similar $m$ value computed and hence will have similar distribution matrices. Since the designed circuit is a 3-ary tree, the $m$ value chosen will be a power of 3 for any

**Table 6.3.** $m$ values when using majority and threshold circuits for different $n$ values for $\mathtt{p} = 0.5, 0.66, \mathtt{e} = 2^{-\frac{n}{4}}$

| $n$ | Majority Circuit | | Threshold Circuit | |
|---|---|---|---|---|
| | $\mathtt{p} = 0.5$ | $\mathtt{p} = 0.66$ | $\mathtt{p} = 0.5$ | $\mathtt{p} = 0.66$ |
| 5 | 9 | 81 | 9 | 9 |
| 10 | 81 | 2187 | 81 | 27 |
| 20 | 2187 | 59049 | 2187 | 81 |
| 30 | 19683 | 531441 | 19683 | 243 |

given $n$. Table 6.3 compares the value of $m$ needed for different $n, \mathtt{p}$ values using majority circuit and threshold circuits to achieve error margin $\mathtt{e} = 2^{\frac{-n}{4}}$.

## 6.6 Search for Distribution Matrix

We realize the threshold circuit using $MAJ_3$ internal nodes and compute the distribution matrix for different values of $n$. To generate the matrix, different random instances of assignment of $y_{\mathrm{i}}$ values of Figure 6.5 from $x_{\mathrm{i}}$ values are considered. A distribution matrix is taken as the matrix $\boldsymbol{M}$ for the access structure if any secret shared using the matrix $\boldsymbol{M}$ can be successfully reconstructed by any set that belongs to the set of qualified subsets of the total number of nodes.

We consider a $(n, \lfloor \frac{2n}{3} \rfloor)$ access structure and compute the distribution matrix $\boldsymbol{M}$ for different number of nodes. A random instance of mapping from literals $x_{\mathrm{i}}, \mathrm{i} \in [n]$ to literals $y_{\mathrm{j}}, \mathrm{j} \in [m]$ needs to be fixed for the computation, to do so one needs to search across the possible random instances of mapping when each $y_{\mathrm{j}}$ is assigned a uniformly sampled $x_{\mathrm{i}}$. Since for each $y_{\mathrm{j}}$, any of the $x_{\mathrm{i}}$ values can be assigned, the size of the assignment space is $n^m$, however the search space can be drastically reduced when considering the number of occurrences of each literal among $x_{\mathrm{i}}$s. Each literal $x_{\mathrm{i}}$ corresponds to the node with index i, hence in an ideal scenario, all the nodes need to occur "uniformly" among the literals $y_{\mathrm{j}}$, that is to say, the number of occurrences/assignments of each $x_{\mathrm{i}}$ to certain $y_{\mathrm{j}}$ should be almost equal. Thus we look at only those random instances where each literal $x_{\mathrm{i}}$ occurs $\sim \frac{m}{n}$ times, so we restrict ourselves to those instance where each literal is assigned literals

between $\left[\lfloor \frac{m}{n} \rfloor, \lceil \frac{m}{n} \rceil + 1\right]$, for each of the instance of random mapping, the distribution matrix is constructed and checked against all the possible threshold combinations.

For an access structure $(n, t)$, there are $\sum_{k=t+1}^{n} \binom{n}{k}$ qualified sets that can reconstruct the secret value, however if the reconstruction is successful for all the $t+1$ element subsets, it will successful for any of the subsets with more than $t+1$ elements. Thus a distribution matrix is declared to be valid if all the $t+1$ element subsets result in correct reconstruction.

**Reconstruction.** When a subset $\mathcal{T}$ of nodes come together to reconstruct a secret, they first compute the vector $\boldsymbol{\lambda}_{\mathcal{T}}$ such that $M_{\mathcal{T}}^{\top}\boldsymbol{\lambda}_{\mathcal{T}} = (1, 0, \cdots, 0)^{\top}$. As can be observed from the dimensions of the matrix $\boldsymbol{M}$, for a threshold access structure $(n, \frac{2n}{3})$, $\boldsymbol{\lambda}_{\mathcal{T}}$ is a solution for under-determined system of linear equations with solution $\{\lambda_i\} \in \{0, 1, -1\}$.

### 6.7 Proactive BBSS Mechanism

System attacks are common as flaws in the software realization of the protocols are ubiquitous. While cryptographic secrecy protects again break-ins, its effect is limited over a longer time. This is especially true in-case of a *mobile* attacker [65], [66] who can break into systems one-by-one over a long time. Proactive secret sharing (PSS) guards against these gradual attacks by combining distributed trust with periodic share renewing. When systems store keys for a long time, even when the secret information is threshold-shared, it is imperative to refresh the shares such that the adversary does not eventually gain all the information. In proactive security [65], [66], [253], the nodes modify their secret shares periodically such that the adversary's knowledge of secret information from any previous period is not useful in the next. For the D-KODE protocol, we propose proactive secret sharing for the BBSS scheme.

**Adversary.** We consider a computationally bounded *mobile* adversary [65] that can corrupt any server any point of time, however, the adversary can corrupt no more than $t$ servers at any instant of time. The adversary after compromising the server has full access to the server's secret information and communication. We consider malicious corruption in which the adversary makes the server deviate arbitrarily from the protocol. The adversary has access to the complete view of the corrupted server's communication, however, he can nei-

ther inject, access or deny messages between any two non-compromised nodes nor affect the broadcast channel. The adversary corrupting the servers is removable by a reboot mechanism [253], which is handled by the system management interacting with the servers. The defined protocol provides explicit mechanism to detect malicious behaviour, we assume a reboot is triggered as soon as malicious behaviour is detected which is completed with in that epoch. The system management initializes the system by establishing server to server communication and no secret information of the protocols is available to it.

The aim of the adversary corrupting the servers is to learn the secret information or the secret key shares involved in the protocol. The user or clients interacts with the servers to obtain partial evaluations of the keys. He may try to attack the system by either predicting the server secret key or the evaluations for other clients. At the end of each refresh phase, the servers erase the old information of the previous epochs. This process is a assumed reliable such that when the server is compromised, the adversary does not have access to the secret information of the previous epochs. If a server is compromised in the refresh phase, the server is assumed to be compromised in both the phases adjacent to that phase.

**Protocol.** We propose a proactive secret sharing scheme [65] for the black box secret sharing mechanism where the size of share-elements does not increase with each refresh. The protocol proceeds in intervals of time called e*pochs*, which are synchronized by the common global clock. The parties participate in a share *refresh* phase at the beginning of each epoch after which every party in the system has access to the new shares. The adversary can corrupt up-to $t$ parties, if it is detected that a certain party is corrupted in an epoch, its shares are renewed in the *share renewal phase* phase of the next epoch, similarly if a node crashes during an epoch, its shares are reconstructed in the *reconstruction phase* of the next epoch. Share renewal and reconstruction are a part of the refresh phase of each epoch.

Without loss of generality, let $(n, t)$ be the access structure of epoch $\mathsf{e}$ and $(n', t')$ be the access structure of the epoch $\mathsf{e}+1$. Let the access structure of epoch $\mathsf{e}$ correspond to the share distribution matrix $\boldsymbol{M}$ and $\boldsymbol{M}'$ for the epoch $\mathsf{e}+1$. Let $\boldsymbol{sk}_{\mathrm{i}}$ be the set of share elements held by the party $P_{\mathrm{i}}$ for the epoch $\mathsf{e}$. In our proactive protocol, each party re-shares

**Proactive BBSS**

The public parameters $\mathsf{pp} = \{n, t, q, p, \boldsymbol{M}, \boldsymbol{M'}, \psi(\cdot), \psi(\cdot)'\}$. Each party $P_\mathrm{i}$ begins with an initial verified share $\boldsymbol{sk}_\mathrm{i}$( and $\boldsymbol{sk}'_\mathrm{i}$) consisting of elements $\hat{s}_{\mathrm{i},k'}$( and $\hat{s'}_{\mathrm{i},k'}$) $\in \mathbb{Z}_q, 0 \leq k' \leq \|\psi^{-1}(\mathrm{i})\|$. $\boldsymbol{M} \in \{0,1\}^{d \times \mathrm{e}}, \boldsymbol{M'} \in \{0,1\}^{d' \times \mathrm{e}'}$. All the honest parties begin with a commitment vector $\boldsymbol{V} = (v_1, v_2 \cdots v_\mathrm{e})$. Share renewal:
For each $k'$ from above party $P_\mathrm{i}$ performs the following:

1. Performs a Verifiable-LISS of each of the share elements among all the parties. Samples random vectors $\boldsymbol{\rho}_\mathrm{i}, \boldsymbol{\rho'}_\mathrm{i} \in \mathbb{Z}_p^{\mathrm{e}'}$ with elements $\rho_{\mathrm{i},l}, \rho'_{\mathrm{i},l}, l \in [\mathrm{e}']$ and computes $\boldsymbol{S}_\mathrm{i} = \boldsymbol{M'} \cdot \boldsymbol{\rho}_\mathrm{i}$ and $\boldsymbol{S'}_\mathrm{i} = \boldsymbol{M'} \cdot \boldsymbol{\rho'}_\mathrm{i}$ with $\rho_{\mathrm{i},1} = \hat{s}_{\mathrm{i},k'}$ and $\rho'_{\mathrm{i},1} = \hat{s'}_{\mathrm{i},k'}$

2. Let the share elements of $\boldsymbol{S}_\mathrm{i}$ and $\boldsymbol{S'}_\mathrm{i}$ be $s_{\mathrm{i},l}$ and $s'_{\mathrm{i},l}, l \in [\mathrm{e}']$. Forward the share elements $s_{\mathrm{i},k}, s'_{\mathrm{i},k}$ to party $P_\mathrm{j}, k \in T_\mathrm{j} = \psi^{-1}(\mathrm{j})$ and commitments $C_{\mathrm{i},l} = g^{\rho_{\mathrm{i},l}} h^{s'_{\mathrm{i},l}}, l \in [\mathrm{e}']$ to all the parties.

3. $P_\mathrm{i}$ verifies the shares and the corresponding commitments received from party $P_\mathrm{j}$ and broadcasts a complaint against $P_\mathrm{j}$ if the verification fails.

4. $P_\mathrm{i}$ computes the qualified set $\mathcal{Q}'$ as in Phase 1 of BBSS-DKG, at the end of which all honest parties compute the same set $\mathcal{Q}'$.

5. $P_\mathrm{i}$ computes the new share as follows: Let $\boldsymbol{M'}_{\mathcal{Q}'}$ be the set of rows held by the parties in the set $\mathcal{Q}'$. Each party computes the vector $\boldsymbol{\lambda}_{\mathcal{Q}'} \in \{0, 1, -1\}^{d_{Q'}}$ such that $\boldsymbol{M'}^{\top}_{\mathcal{Q}'} \cdot \boldsymbol{\lambda}_{\mathcal{Q}'} = \boldsymbol{\varepsilon}$. The new share of $P_\mathrm{i}$ is $\boldsymbol{sk}'_\mathrm{i} = \tilde{\boldsymbol{S}}^{\top}_{\mathrm{i},\mathcal{Q}'} \cdot \boldsymbol{\lambda}_{\mathcal{Q}'}$, where $\tilde{\boldsymbol{S}}_{\mathrm{i},\mathcal{Q}'}$ is the set of all share elements received by party $P_\mathrm{i}$ from the parties in the set $\mathcal{Q}'$.

**Figure 6.6.** Proactive BBSS Scheme

every share element held by the party to all other parties of the next epoch. The Proactive BBSS scheme is presented in Figure 6.6.

Proactive BBSS offers the following properties [253]:

- *Robustness/Correctness*: The new shares computed at the end of the share renewal phase correspond to the original secret $sk$ shared among the parties i.e., any qualified set of parties - $t+1$ or more number of parties can reconstruct the secret $sk$.

- *Secrecy:* No information about the secret $sk$ is obtained by the $t$-limited adversary in any epoch. The adversary who obtains shares of no more than $t$ parties has no information about the secret $sk$ in any epoch.

- *Liveness*: All honest parties complete the refresh of shares (at the beginning) in each epoch.

The proactive BBSS mechanism works mainly in two steps:

- Each party $P_{\mathrm{i}}, \mathrm{i} \in [n]$ does verified secret sharing of each of its shares $\boldsymbol{sk}_{\mathrm{i}}$ among all the parties
- From the obtained verified shares, each party *reconstructs* their new shares $\boldsymbol{sk}'_{\mathrm{i}}$.

Let $C_{\mathrm{i}}$ be the vector of commitments to the vector $\boldsymbol{\rho}_{\mathrm{i}}$ by each party $P_{\mathrm{i}}$ in the previous epoch and $\mathcal{Q}$ be the qualified set computed during that epoch. Each party stores a vector $V$ of commitments from the parties of qualified set computed during the re-sharing from the previous epoch for the verifiability of shares for the next epoch. All the honest parties update the commitment vector $\boldsymbol{V}$ with elements $V_l = \prod_{P_{\mathrm{i}} \in \mathcal{Q}} C_{\mathrm{i},\ell}^{\lambda_{\mathrm{i}}}, \ell \in [\mathrm{e}]$. When party $P_{\mathrm{i}}$ shares $\hat{s}_{\mathrm{i},k}$ (while using $\hat{s'}_{\mathrm{i},k}$), each party $P_{\mathrm{j}}$ checks if $g^{\hat{s}_{\mathrm{i},k}} h^{\hat{s'}_{\mathrm{i},k}} = \prod_k (V_k)^{m_{\mathrm{i},k}}$ where $\boldsymbol{M}_{\mathcal{Q}}^{\top} \boldsymbol{\lambda}_{\mathcal{Q}} = \varepsilon, \boldsymbol{\lambda} = \{\lambda_k, k \in \bigcup_{\mathrm{i}} T_{\mathrm{i}}, P_{\mathrm{i}} \in \mathcal{Q}\}$. Let $s_{\mathrm{i},k}, k \in T_{\mathrm{j}}$ be the shares received by $P_{\mathrm{j}}$ from party $P_{\mathrm{i}} \in \mathcal{Q}'$. $\mathcal{R}' = \{\bigcup_{\mathrm{i}} T_{\mathrm{i}}, P_{\mathrm{i}} \in \mathcal{Q}'\}$ is the set of all rows held by $\mathcal{Q}'$. $P_{\mathrm{j}}$ computes the new share element $s_k = \sum_{\mathrm{i} \in \mathcal{Q}'} \lambda_{\mathrm{i}} s_{\mathrm{i},k}, k \in T_{\mathrm{j}}$.

**Theorem 6.7.1.** *Given a correct and secure $(n,t)$-verifiable BBSS scheme, the Proactive BBSS protocol of Figure 6.6 satisfies correctness and secrecy properties under the discrete logarithm assumption.*

*Proof.* **Correctness.** Let $(n,t),(n',t')$ be access structures in the epochs $\mathsf{e}$ and $\mathsf{e}+1$. Without loss of generality let $\boldsymbol{sk}_{\mathrm{i}}, \mathrm{i} \in [n]$ be shares of secret key $sk$ of the $n$ parties in epoch $\mathsf{e}$ and $\boldsymbol{sk}'_{\mathrm{i}}, \mathrm{i} \in [n']$ be shares of the $n'$ parties in epoch $\mathsf{e}+1$. We need to show that any set of $t'+1$ or more parties in epoch $\mathsf{e}+1$ reconstruct the secret key $sk$.

For epoch e, the share elements held by parties in qualified set $\mathcal{Q}$ are $\hat{s}_k, k \in \mathcal{R} = \{\bigcup_{\mathrm{i}} T_{\mathrm{i}}, P_{\mathrm{i}} \in \mathcal{Q}\}$. $\mathcal{R}$ is the set of all rows held by the parties in $\mathcal{Q}$. We know, $sk = \sum_{k \in \mathcal{R}} \lambda_k \hat{s}_k$

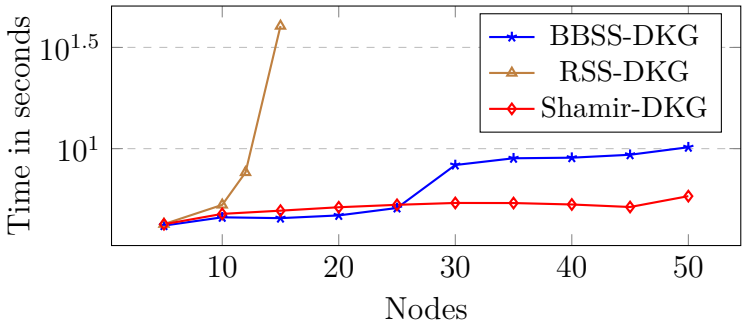



**Figure 6.7.** Time taken to perform DKG to generate shares of a 256-bit key for Shamir-DKG and 283-bit value for RSS and BBSS-DKG. The values show the mean of values across nodes for 10 runs of the protocol.

However, each share element $\hat{s}_k$ is verifiable secret shared in the next epoch $e+1$. Thus any qualified set $\mathcal{Q}'$ of $t'+1$ parties can construct the share element $\hat{s}_k$. Let $\mathcal{R}'$ be the rows held by the parties in $\mathcal{Q}'$. Then,

$$sk = \sum_{i \in \mathcal{R}} \lambda_i \hat{s}_i = \sum_{i \in \mathcal{R}} \lambda_i \Big( \sum_{j \in \mathcal{R}'} \lambda_j s_{i,j} \Big)$$
$$= \sum_{j \in \mathcal{R}'} \lambda_j \Big( \sum_{i \in \mathcal{R}} \lambda_i s_{i,j} \Big) = \sum_{j \in \mathcal{R}'} \lambda_j s_j = sk$$

**Secrecy.** The secrecy of the secret in each phase follows from the security properties of Verifiable BBSS scheme. Let $\mathcal{B}, \mathcal{B}', \|\mathcal{B}\|, \mathcal{B}'\| < t$ be the set of servers corrupted in an epoch $e$ and $e+1$. W.l.o.g let $\mathcal{B} \cap \mathcal{B}' = \phi$, from the correctness principle above, we know that any $t'+1$ or more parties can construct the secret key in the epoch $e+1$. From the security of the BBSS scheme we know what no set of $t'$ or less number of parties has any information about the secret, hence maintaining the secrecy property. □

### 6.8 Performance Analysis

We evaluate the performance of D-KODE protocol, using AWS EC2 t3a.2xlarge instances with 8 cores. Our prototype Python implementation includes BBSS, BBSS-DKG, BBSS-PSS and the corresponding reference implementations of New-JF-DKG [30] instantiated with Shamir secret sharing and replicated secret sharing (RSS).

**Distributed Key Generation (DKG).** We implement the DKG protocols using Tendermint [254] as a broadcast channel for verifiable secret sharing. Figure 6.7 provides a logarithmic plot comparing time taken to run DKG to generate shares of a 256-bit key using Shamir, replicated (RSS) [82] and black-box (BBSS) secret sharing schemes for up-to 50 nodes. RSS is a well-known scheme [82] to share secrets in $\mathbb{Z}_q$ in an additive form. The access structure for the verifiable sharing corresponds to $(n, \lfloor \frac{2n}{3} \rfloor)$ threshold in all the protocols analysed through Figure 6.7.

Shamir secret sharing allocates one share element per node while BBSS and RSS allocate share *vectors*. The vector length for RSS grows exponentially as $\binom{n-1}{t}$ for $(n, t)$-sharing. The share vector length for a node in BBSS is determined by the distribution matrix and the share allocation function $\psi(\cdot)$. While BBSS allocates more than one share element per user, the verification of shares is efficient involving only multiplications instead of exponentiations since the distribution matrix is a sparse binary matrix. This is reflected in the slightly lesser times recorded compared to Shamir-DKG for up-to 27 nodes. The distribution matrix is of dimension $36 \times 22$ (with different $\psi(\cdot)$ function) when the number of nodes $n \in [4, 9]$; it is $216 \times 130$ and $1296 \times 778$ for $n \in [11, 27]$ and $n \in [28, 50]$ respectively. Beyond 28 nodes, the time to perform BBSS-DKG shows a jump owing to the change in the distribution matrix size. Such a change in matrix occurs at 10 nodes as well, however the change in the time taken is not too significant. The higher variance beyond $n = 28$ for BBSS-DKG results from the difference in the number of shares obtained by each node. While using RSS, the time taken for DKG grows exponentially owing to exponential increase in number of shares per node with the $n$, the scheme becomes un viable beyond 12-15 nodes. In Shamir-DKG, since each node provides only one share element for every other node, the time taken is the lowest for higher $n$. Though the time taken to perform BBSS-DKG can be higher than Shamir-DKG, it is the *number of instances* of the DKG that is significantly lesser while employing D-KODE protocol when compared to the Plain-DKG [4].

**Distributed PRF.** D-KODE provides key-shares using PRF $F(X, \mathsf{k})$ where $\mathsf{k}$ is a vector. Each element of the vector $\mathsf{k}$ at the server is a share generated using BBSS-DKG. The

[4] ↑The basic differences between Plain-DKG and D-KODE are recalled in Table 6.4

**Table 6.4.** Comparison of Plain-DKG and D-KODE with BBSS for $n$ servers, $c$ keys and a constant $k$. Refer Table 6.5 for the exact value of $k$ for different $n$.

| | Plain-DKG | D-KODE (with BBSS) |
|---|---|---|
| Key-generation | DKG, Consensus for every new-id | Non-interactive PRF |
| Storage - shares | $c$ | $k$ |
| Communication complexity (PSS) | $c \cdot O(n^3)$ | $k \cdot O(n^3)$ |
| Security assumptions | DLog | DLog and LWR |
| Preferable for keys range | $< 100$K | $> 100$K |

parameters (LWR) for computing the PRF are chosen as following: $\mathsf{n} = 8192$, $q$ : 256-bit, $p$ : 32-bit. The parameter $q' > pq$ used for commitments is 571-bit with commitments on the curve `secp571r1`. The servers run 8192 instances of BBSS-DKG to generate shares for the key k. The PRF output is a $32-$bit key; for the scenario where user generates the private key, for each input $X$, the servers compute 8 instances of the PRF by deriving 8 inputs from $X$ and finally forward the resulting 32 bit keys to the user. The user concatenates them to generate a $256-$bit key. The corresponding public key is computed on the `secp256k1` curve. In the case of computing public key of another party, the servers use 8 instances of 32-bit key shares to generate the public key share (on the curve `secp256k1`) and forward it to the user. Each server takes $< 300$ msec to generate shares for a user per thread, for $n \in [5, 50]$; AWS EC2 t3a.2xlarge on an average (over 10 runs) supports 35 such threads per second.

Decreasing the value of $p$ increases the bit-security and can reduce the vector length n for a fixed bit-security, however it increases the number of instances of PRF required to generate a 256-bit key and vice versa. The chosen parameters offer more than 128bits of LWR security (for solving the LWR instance) as estimated using the LWE-estimator [255]. Since the secret keys are 256-bit with public keys on the `secp256k1` curve, the bit-security offered by our system is 128-bits which is similar to the Plain-DKG approach.

**D-KODE vs Plain-DKG.** D-KODE allows clients to generate private and public keys using partial share-evaluations from different servers. Plain-DKG approach is another way to provide such key shares where one instance of DKG is run *per user* to provide the shares (private or public key shares) whenever requested. In this, for every new user the servers

perform consensus on index of pre-shared keys and offer the key shares to the user. As Shamir-DKG is efficient even for higher number of servers as shown in Figure 6.7, we consider Shamir-DKG for Plain-DKG approach. We compare D-KODE with Plain-DKG as it is the only other major approach available currently in the industry (Torus[60], Sepior [64] etc).

When the servers store keys, either own or user's secret keys for a long-time, proactively refreshing the shares is inevitable. This is one of key phases where D-KODE offers advantage. To bring this out, we compare the different number of shares and commitments stored at each server when using different schemes to provide the key shares in Table 6.5. The table compares D-KODE where the master key between servers is shared using RSS and BBSS and Plain-DKG for different number of servers and clients present in the system. For share refreshing, each share value stored at the server is re-shared in the next epoch. Hence number of shares stored at each server is the same as the number of verifiable secret sharings to be performed in the next round. Table 6.5 also provides the number of commitments to be stored at each server for the pro-active secret sharing, thus providing the total storage requirement. Each share value is a $\mathbb{Z}_q$ element where $q$ is 256-bit and the commitments are on the curve `secp256k1` where each commitment is of size 56 bytes. Plain-DKG stores $t+1$ commitments for each $(n, t)$ DKG, hence for $c$ number of client keys, stores $c \cdot (t+1)$ commitments per server. For BBSS with distribution matrix of size $d \times \mathrm{e}$, each server stores e commitments per shared value. Hence for 8192-element master key, stores $8192 \cdot \mathrm{e}$ commitments. For RSS, each server forwards commitments to each of the share, hence the number of commitments is $8192 \cdot \binom{n}{t}$.

For Plain-DKG, since the number of shares is same as the number of keys and hence linear with, increasing the share-refresh time with higher number of keys. D-KODE uses a fixed 256-element long master key vector shared among the servers. Only shares corresponding to the master key vector need to refreshed at each round and does not change with the number of keys. For D-KODE with RSS, the number of shares is constant with respect to the users but increases exponentially with the number of servers. The number of shares stored at the server when D-KODE is used with BBSS is dependent on the distribution vector. Since the actual number of share elements per server may vary depending on the share distribution function, we provide the average number of share elements per server. For

**Table 6.5.** Number of shares per server while using Plain-DKG[30] and D-KODE with either RSS or BBSS for $c$ keys. Here, $c$ can be as large as 1 billion. Number of verifiable secret sharing instances for share refreshing is same as the average number of shares stored. The shares are 256-bit for Plain-DKG and 283-bit for BBSS.

| No. of keys (c) | No. of servers ($n$) | Average number of shares per server | | |
|---|---|---|---|---|
| | | Plain DKG | D-KODE | |
| | | | With RSS | With BBSS |
| $c$ | 5 | $c$ | 32768 | 58982.4 |
| | 10 | $c$ | 688,128 | 176,947.2 |
| | 20 | $c$ | 22.224e+7 | 88,473.6 |
| | 30 | $c$ | 82.016e+9 | 353,894.4 |
| | 40 | $c$ | 66.528e+12 | 265,420.8 |
| | 50 | $c$ | 27.424e+15 | 212,336.64 |

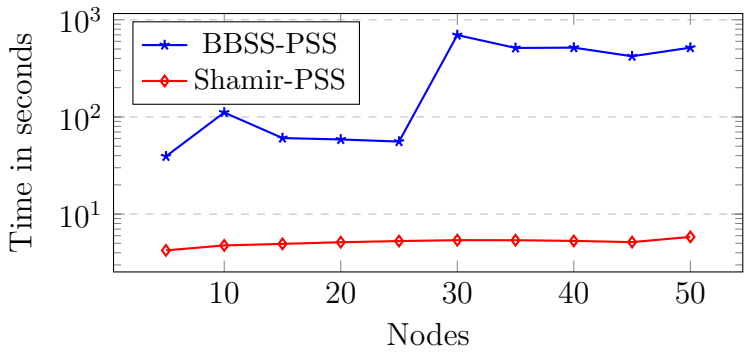



**Figure 6.8.** Time taken to refresh shares corresponding to one scalar value using PSS. For BBSS, it corresponds to re-sharing a total of 216 values for 10-27 nodes and 1296 283-bit values for $28-50$ node network. For Shamir secret sharing, each node re-shares just one 256-bit element per key. The values show the mean of values across nodes for 10 runs of the protocol.

the ranges $n \in [4, 9], [10, 27], [28, 50]$, the distribution matrix would be the same within each range. Hence with increasing $n$ in those ranges, the average number of shares per server decreases. In fact, the distribution matrix would again change at $n = 82$.

Figure 6.8 shows the time to refresh one share through proactive secret sharing (PSS). BBSS-PSS takes longer as the number of share elements per server is higher whereas it is just one element for Shamir secret sharing while sharing a single secret value. The increase

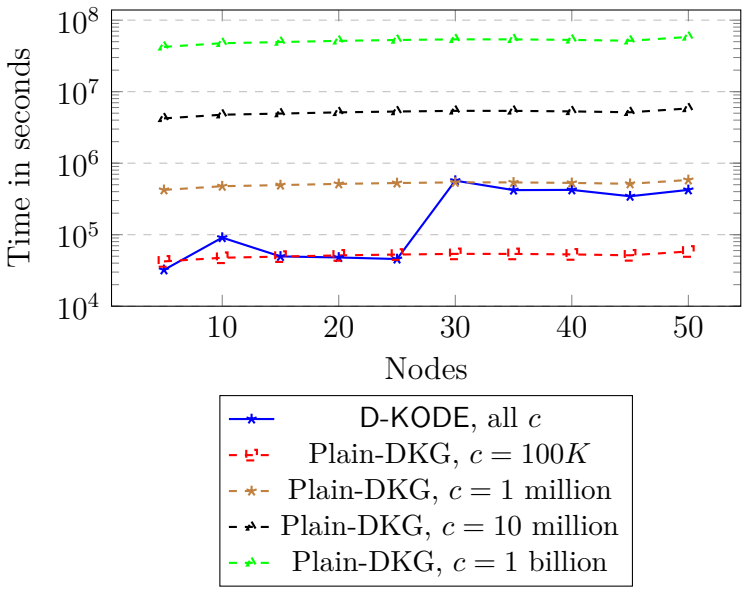



**Figure 6.9.** Estimated time to refresh shares through proactive secret sharing for D-KODE and Plain-DKG for number of keys $c = 100K$, 1million and 10million for 10 parallel instances. D-KODE re-shares shares of a fixed number of 8192 values and hence takes the same time even for a billion keys ($c = 1$ billion); Plain-DKG re-shares values equal to the number of keys.

in time at $n = 10$ and $n = 28$ for BBSS-PSS is due to the change in distribution matrix size. Figure 6.9 shows the estimated time to refresh shares using D-KODE and Plain-DKG for increasing number of keys. We note that any parallelization applied to speed-up can be applied to both schemes. Hence, we provide an estimate of times taken by appropriately scaling the timing values obtained for re-sharing of single share value. D-KODE out-performs Plain-DKG for 94K and higher keys when the number of servers used is below 27. In the range of 28-50 servers, D-KODE out-performs Plain-DKG from 1 million keys. D-KODE protocol also offers the non-trivial advantages of storing shares of 8192-element key vector versus millions of key-shares and the servers being essentially non-interactive except during the share-refreshing phase. D-KODE is particularly suitable for large-scale service-offering scenarios involving millions of keys.

# 7. CONCLUSION AND FUTURE WORK

In this work, we have addressed two applications of threshold cryptography, realizing information escrows and key maintenance using threshold cryptography. We first realize a two-party escrow mechanism in which any dishonest behavior is automatically penalized by loss of the claim-or-refund deposit made prior to the escrow mechanism operations. The proposed Pepal protocol which employs the proposed doubly-oblivious transfer primitive allows the sender to penalize even partial leakages. We extend the two-party escrow scenario to a multi-party escrow scenario where a group of agents offers the service such that the information of the sender is distributed among the agents. Any qualified set of agents (set greater than a certain threshold size) may collude and decrypt the sender's information prematurely. To prevent this we propose the CDE protocol which ensures that any such collusion results in a subset of agents being able to transfer the claim-or-refund deposits of all the other agents. The enforced policy of banning colluding agents ensures that the non-collusion is the best response strategy for the agents. In the future, we aim to realize partial leakage prevention in the multi-agent collusion scenario. For this, a multi-party version of doubly-oblivious transfer or an efficient cryptographic primitive which realizes a similar functionality may be proposed. Alternative mechanisms of collusion prevention including complaints on other agents which may remove the necessity of claim-or-refund deposits would be a good future direction to explore in realizing escrows.

Towards understanding the mental models of the users in adopting different cryptocurrency wallets, we analyzed a total of 255 valid responses of participants through an online survey. We analyzed why the adoption of the more secure multi-device wallets is not high compared to the single-device wallets and if the participants are willing to shift to multi-device wallets after learning their security properties. We identified that the users behave as two specific groups of newbies and non-newbies and the majority of them are willing to shift to multi-device wallets but few are not. I would be interesting to see how these groups evolve with advancements in the settings are properties offered by the multi-device wallets in the future. To propose a mechanism to achieve efficient threshold key-management, we design the D-KODE protocol which uses practical black-box secret sharing to distribute a

master secret key among a set of servers. The servers compute a lattice-based PRF on the user's input as a partial evaluation of the user key. The user combines the partial evaluations from the servers to compute the required keys. Alternatively, the user can request the servers to generate threshold signatures on behalf of the users. In the D-KODE setup it is assumed that up to a certain threshold $t$ number of servers can be corrupted and the rest are honest and hence will not collude. It would be interesting in the future to consider rationality in such a key-management protocol that can deter collusion. The D-KODE protocol can also be extended to consider different communication models including the asynchronous model.

# REFERENCES


[1] NSW, *Nsw data and information custodianship policy*, https://data.nsw.gov.au/sites/default/files/inline-files/NSW%20Data%20and%20Information%20Custodianship%20Policy%20v1-0_0.pdf, n.d.

[2] I. Ayres and C. Unkovic, "Information escrows," *Mich. L. Rev.*, vol. 111, p. 145, 2012.

[3] V. Arun, A. Kate, D. Garg, P. Druschel, and B. Bhattacharjee, "Finding safety in numbers with secure allegation escrows," in *NDSS 2020*, 2020.

[4] B. Kuykendall, H. Krawczyk, and T. Rabin, "Cryptography for #metoo," *Proc. Priv. Enhancing Technol.*, vol. 2019, no. 3, pp. 409–429, 2019.

[5] A. Rajan, L. Qin, D. W. Archer, D. Boneh, T. Lepoint, and M. Varia, "Callisto: A cryptographic approach to detecting serial perpetrators of sexual misconduct," in *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies, COMPASS 2018*, E. W. Zegura, Ed., 2018, 49:1–49:4.

[6] T. C. May, *Timed-release crypto*, http://cypherpunks.venona.com/archive/1993/02/msg00129.html, 1993.

[7] G. Di Crescenzo, R. Ostrovsky, and S. Rajagopalan, "Conditional oblivious transfer and timed-release encryption," English, in *Advances in Cryptology â EUROCRYPT â 99*, ser. Lecture Notes in Computer Science, J. Stern, Ed., vol. 1592, Springer Berlin Heidelberg, 1999, pp. 74–89, ISBN: 978-3-540-65889-4.

[8] J. H. Cheon, N. Hopper, Y. Kim, and I. Osipkov, "Provably secure timed-release public key encryption.," *ACM Trans. Inf. Syst. Secur.*, vol. 11, no. 2, Feb. 29, 2008. [Online]. Available: http://dblp.uni-trier.de/db/journals/tissec/tissec11.html#CheonHKO08.

[9] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," Massachusetts Institute of Technology, Cambridge, MA, USA, Tech. Rep., 1996.

[10] *Escrow tech*, https://www.escrowtech.com/.

[11] *Iron mountain*, https://www.ironmountain.com/information-management/software-escrow.

[12] C. Hourihan and B. Cline, *A look back: U.s. healthcare data breach trends"*, https://hitrustalliance.net/content/uploads/2014/05/HITRUST-Report-U.S.-Healthcare-Data-Breach-Trends.pdf, 2008.

[13] *Man in the cloud (mitc) attacks*, https://www.imperva.com/docs/HII_Man_In_The_Cloud_Attacks.pdf, 2015.

[14] Y. Rahulamathavan, M. Rajarajan, O. F. Rana, M. S. Awan, P. Burnap, and S. K. Das, "Assessing data breach risk in cloud systems," in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, Nov. 2015, pp. 363–370.

[15] T. Floyd, M. Grieco, and E. F. Reid, "Mining hospital data breach records: Cyber threats to u.s. hospitals," in *2016 IEEE Conference on Intelligence and Security Informatics (ISI)*, Sep. 2016, pp. 43–48.

[16] I. F. Blake and A. C.-F. Chan, "Scalable, server-passive, user-anonymous timed release public key encryption from bilinear pairing," in *In Proc. IJCAI '01*, 2004, pp. 504–513.

[17] M. O. Rabin and C. Thorpe, "Time-lapse cryptography," Harvard University School of Engineering and Applied Sciences, Tech. Rep., 2006.

[18] J. H. Cheon, N. Hopper, Y. Kim, and I. Osipkov, "Timed-release and key-insulated public key encryption," *IACR Cryptology ePrint Archive*, vol. 2004, p. 15, 2004. [Online]. Available: http://eprint.iacr.org/2004/231.

[19] Y. Watanabe and J. Shikata, "Timed-release computational secret sharing and threshold encryption," *Designs, Codes and Cryptography*, vol. 86, no. 1, pp. 17–54, 2018.

[20] Y. G. Desmedt and Y. Frankel, "Threshold cryptosystems," in *Proceedings on Advances in Cryptology*, ser. CRYPTO '89, Santa Barbara, California, USA: Springer-Verlag New York, Inc., 1989, pp. 307–315, ISBN: 0-387-97317-6. [Online]. Available: http://dl.acm.org/citation.cfm?id=118209.118237.

[21] D. Boneh, X. Boyen, and S. Halevi, "Chosen ciphertext secure public key threshold encryption without random oracles," English, in *Topics in Cryptology â CT-RSA 2006*, ser. Lecture Notes in Computer Science, D. Pointcheval, Ed., vol. 3860, Springer Berlin Heidelberg, 2006, pp. 226–243, ISBN: 978-3-540-31033-4.

[22] J. Doerner, Y. Kondi, E. Lee, and a. shelat, "Secure two-party threshold ecdsa from ecdsa assumptions," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 595–612. DOI: 10.1109/SP.2018.00036. [Online]. Available: doi.ieeecomputersociety.org/10.1109/SP.2018.00036.

[23] *Data protection and breach*, https://otalliance.org/system/files/files/resource/documents/dpd_2015_guide.pdf, 2015.

[24] *Data breaches*, https://www.privacyrights.org/data-breaches?title=&breach_type%5B%5D=267, n.d.

[25] C. M. Bast, "At what price silence: Are confidentiality agreements enforceable?" *William Mitchell Law Review*, vol. 25, no. 2, 1999.

[26] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," in *ICC*, 2014.

[27] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Secure multiparty computations on bitcoin," in *IEEE Symposium on Security and Privacy*, 2014.

[28] T. Ruffing, A. Kate, and D. Schröder, "Liar, liar, coins on fire!: Penalizing equivocation by loss of bitcoins," in *ACM CCS*, 2015.

[29] R. Gennaro, M. O. Rabin, and T. Rabin, "Simplified vss and fast-track multiparty computations with applications to threshold cryptography," in *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '98, Puerto Vallarta, Mexico: ACM, 1998, pp. 101–111, ISBN: 0-89791-977-7. DOI: 10.1145/277697.277716. [Online]. Available: http://doi.acm.org/10.1145/277697.277716.

[30] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *J. Cryptol.*, vol. 20, no. 1, pp. 51–83, Jan. 2007, ISSN: 0933-2790. DOI: 10.1007/s00145-006-0347-3. [Online]. Available: http://dx.doi.org/10.1007/s00145-006-0347-3.

[31] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '91, London, UK, UK: Springer-Verlag, 1992, pp. 129–140, ISBN: 3-540-55188-3. [Online]. Available: http://dl.acm.org/citation.cfm?id=646756.705507.

[32] O. Catrina and A. Saxena, "Secure computation with fixed-point numbers.," in *Financial Cryptography*, R. Sion, Ed., ser. Lecture Notes in Computer Science, vol. 6052, Springer, 2010, pp. 35–50, ISBN: 978-3-642-14576-6. [Online]. Available: http://dblp.uni-trier.de/db/conf/fc/fc2010.html#CatrinaS10.

[33] T. Veugen, "Linear round bit-decomposition of secret-shared values," *Information Forensics and Security, IEEE Transactions on*, vol. 10, no. 3, pp. 498–506, 2015, ISSN: 1556-6013.

[34] T. Toft, "Topics in cryptology – ct-rsa 2009: The cryptographers' track at the rsa conference 2009, san francisco, ca, usa, april 20-24, 2009. proceedings," in Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, ch. Constant-Rounds, Almost-Linear Bit-Decomposition of Secret Shared Values, pp. 357–371, ISBN: 978-3-642-00862-7.

[35] A. Lysyanskaya and N. Triandopoulos, "Rationality and adversarial behavior in multi-party computation," in *Annual International Cryptology Conference*, Springer, 2006, pp. 180–197.

[36] *Bitcoin price history*, https://www.investopedia.com/articles/forex/121815/bitcoins-price-history.asp, 2021.

[37] *Total cryptocurrency market cap, 2021*, https://coinmarketcap.com/charts/, 2021.

[38] *Crypto: A new asset class*, https://www.goldmansachs.com/insights/pages/crypto-a-new-asset-class-f/report.pdf, 2021.

[39] *Coinbase wallet*, https://wallet.coinbase.com/.

[40] *Coinbase revenue and usage statistics (2021)*, https://www.businessofapps.com/data/coinbase-statistics/.

[41] *Binance*, https://www.binance.com/en/.

[42] *Coin ranking - binance exchange*, https://coinranking.com/exchange/-zdvbieRdZ+Bbinance.

[43] A. Voskobojnikov, O. Wiese, M. Mehrabi Koushki, V. Roth, and K. ( Beznosov, "The u in crypto stands for usable: An empirical study of user experience with mobile cryptocurrency wallets," ser. CHI '21, Yokohama, Japan: Association for Computing Machinery, 2021, ISBN: 9781450380966. DOI: 10.1145/3411764.3445407. [Online]. Available: https://doi.org/10.1145/3411764.3445407.

[44] M. Fröhlich, F. Gutjahr, and F. Alt, "Don't lose your coin! investigating security practices of cryptocurrency users," in *Proceedings of the 2020 ACM Designing Interactive Systems Conference*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1751–1763, ISBN: 9781450369749. [Online]. Available: https://doi.org/10.1145/3357236.3395535.

[45] X. Gao, G. D. Clark, and J. Lindqvist, "Of two minds, multiple addresses, and one ledger: Characterizing opinions, knowledge, and perceptions of bitcoin across users and non-users," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2016, pp. 1656–1668, ISBN: 9781450333627. [Online]. Available: https://doi.org/10.1145/2858036.2858049.

[46] L. Glomann, M. Schmid, and N. Kitajewa, "Improving the blockchain user experience - an approach to address blockchain mass adoption issues from a human-centred perspective," in *Advances in Artificial Intelligence, Software and Systems Engineering*, T. Ahram, Ed., Cham: Springer International Publishing, 2020, pp. 608–616, ISBN: 978-3-030-20454-9.

[47] K. Krombholz, A. Judmayer, M. Gusenbauer, and E. Weippl, "The other side of the coin: User experiences with bitcoin security and privacy," in *Financial Cryptography and Data Security*, J. Grossklags and B. Preneel, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 555–580, ISBN: 978-3-662-54970-4.

[48] S. Eskandari, J. Clark, D. Barrera, and E. Stobert, "A first look at the usability of bitcoin key management," *arXiv preprint arXiv:1802.04351*, 2018.

[49] I. Eyal, *On cryptocurrency wallet design*, Cryptology ePrint Archive, Report 2021/1522, https://ia.cr/2021/1522, 2021.

[50] M. Vasek, J. Bonneau, R. Castellucci, C. Keith, and T. Moore, "The bitcoin brain drain: A short paper on the use and abuse of bitcoin brain wallets," *Financial Cryptography and Data Security, Lecture Notes in Computer Science. Springer*, 2016.

[51] M. Arapinis, A. Gkaniatsou, D. Karakostas, and A. Kiayias, "A formal treatment of hardware wallets," in *Financial Cryptography and Data Security*, I. Goldberg and T. Moore, Eds., Cham: Springer International Publishing, 2019, pp. 426–445, ISBN: 978-3-030-32101-7.

[52] *Hackers move 760 million from the 2016 bitfinex hack*, https://therecord.media/hackers-move-760-million-from-the-2016-bitfinex-hack/.

[53] *The complete list of crypto exchange hacks*, https://www.hedgewithcrypto.com/cryptocurrency-exchange-hacks/, 2021.

[54] *A comprehensive list of cryptocurrency exchange hacks*, https://selfkey.org/list-of-cryptocurrency-exchange-hacks/, 2020.

[55] *Torus wallet*, https://tor.us.

[56] *Zengo wallet*, https://zengo.com.

[57] Y. Desmedt, "Threshold cryptography," in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg and S. Jajodia, Eds. Boston, MA: Springer US, 2011, pp. 1288–1293, ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_330. [Online]. Available: https://doi.org/10.1007/978-1-4419-5906-5_330.

[58] *Nist- projects - multi-party threshold cryptography*, https://csrc.nist.gov/Projects/threshold-cryptography.

[59] *Kraken*, https://www.kraken.com/.

[60] *Torus*, http://Tor.us.

[61] *Coindesk airdrop archive*, https://www.coindesk.com/tag/airdrops.

[62] *Cocoricos airdrops*, https://cocoricos.io/airdrops.

[63] *Airdrop king*, https://airdropking.io/en/.

[64] *Sepior*, https://sepior.com/.

[65] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *Annual International Cryptology Conference*, Springer, 1995, pp. 339–352.

[66] R. Ostrovsky and M. Yung, "How to withstand mobile virus attacks," in *Proceedings of the tenth annual ACM symposium on Principles of distributed computing*, 1991, pp. 51–59.

[67] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 2019, pp. 241–264.

[68] M. Naor and O. Reingold, "Synthesizers and their application to the parallel construction of pseudo-random functions," in *Proceedings of IEEE 36th Annual Foundations of Computer Science*, 1995, pp. 170–181.

[69] A. Banerjee, C. Peikert, and A. Rosen, "Pseudorandom functions and lattices," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2012, pp. 719–737.

[70] M. Naor, B. Pinkas, and O. Reingold, "Distributed pseudo-random functions and kdcs," in *International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 1999, pp. 327–346.

[71] D. Boneh, K. Lewi, H. Montgomery, and A. Raghunathan, "Key homomorphic prfs and their applications," in *Annual Cryptology Conference*, Springer, 2013, pp. 410–428.

[72] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography," *J. Cryptol.*, vol. 18, no. 3, pp. 219–246, 2005, ISSN: 0933-2790. DOI: 10.1007/s00145-005-0318-0. [Online]. Available: https://doi.org/10.1007/s00145-005-0318-0.

[73] M. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen, "Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity," in *Advances in Cryptology – ASIACRYPT 2016*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 191–219, ISBN: 978-3-662-53887-6.

[74] L. Grassi, C. Rechberger, D. Rotaru, P. Scholl, and N. P. Smart, "Mpc-friendly symmetric key primitives," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16, New York, NY, USA: Association for Computing Machinery, 2016, pp. 430–443, ISBN: 9781450341394. DOI: 10.1145/2976749.2978332. [Online]. Available: https://doi.org/10.1145/2976749.2978332.

[75] J. B. Nielsen, "A threshold pseudorandom function construction and its applications," in *Annual International Cryptology Conference*, Springer, 2002, pp. 401–416.

[76] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions," *J. ACM*, vol. 51, no. 2, pp. 231–262, 2004, ISSN: 0004-5411. DOI: 10.1145/972639.972643. [Online]. Available: https://doi.org/10.1145/972639.972643.

[77] Y. Dodis and A. Yampolskiy, "A verifiable random function with short proofs and keys," in *International Workshop on Public Key Cryptography*, Springer, 2005, pp. 416–431.

[78] Y. Dodis, "Efficient construction of (distributed) verifiable random functions," in *International Workshop on Public Key Cryptography*, Springer, 2003, pp. 1–17.

[79] *Zengo, kzen networks*, https://zengo.com/.

[80] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979, ISSN: 0001-0782.

[81] M. Ito, A. Saito, and T. Nishizeki, "Secret sharing scheme realizing general access structure," *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, vol. 72, no. 9, pp. 56–64, 1989.

[82] R. Cramer, I. Damgård, and Y. Ishai, "Share conversion, pseudorandom secret-sharing and applications to secure computation," in *Theory of Cryptography Conference*, Springer, 2005, pp. 342–362.

[83] R. Cramer and S. Fehr, "Optimal black-box secret sharing over arbitrary abelian groups," in *Advances in Cryptology — CRYPTO 2002*, M. Yung, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 272–287.

[84] *Nist roadmap toward criteria for threshold schemes for cryptographic primitives*, https://csrc.nist.gov/publications/detail/nistir/8214a/final.

[85] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology - CRYPTO 2012*, 2012, pp. 643–662.

[86] D. Lu, T. Yurek, S. Kulshreshtha, R. Govind, A. Kate, and A. Miller, "Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 887–903.

[87] A. Barak, M. Hirt, L. Koskas, and Y. Lindell, "An end-to-end system for large scale p2p mpc-as-a-service and low-bandwidth mpc for weak participants," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 695–712.

[88] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra, *A new approach to practical active-secure two-party computation*, Cryptology ePrint Archive, Report 2011/091, https://eprint.iacr.org/2011/091, 2011.

[89] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Advances in Cryptology – Crypto*, 1991, pp. 420–432.

[90] G. Blakley, "Safeguarding cryptographic keys," in *Proceedings of the 1979 AFIPS National Computer Conference*, Monval, NJ, USA: AFIPS Press, 1979, pp. 313–317.

[91] J. C. Benaloh, "Secret sharing homomorphisms: Keeping shares of a secret secret (extended abstract)," in *Advances in Cryptology — CRYPTO' 86*, A. M. Odlyzko, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 251–260, ISBN: 978-3-540-47721-1.

[92] M. Bellare and P. Rogaway, "Robust computational secret sharing and a unified account of classical secret-sharing goals," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, ser. CCS '07, New York, NY, USA: Association for Computing Machinery, 2007, pp. 172–184, ISBN: 9781595937032. DOI: 10.1145/1315245.1315268. [Online]. Available: https://doi.org/10.1145/1315245.1315268.

[93] I. Damgård and R. Thorbek, "Linear integer secret sharing and distributed exponentiation," in *Public Key Cryptography - PKC 2006*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 75–90, ISBN: 978-3-540-33852-9.

[94] J. Benaloh and J. Leichter, "Generalized secret sharing and monotone functions," in *Advances in Cryptology — CRYPTO' 88*, S. Goldwasser, Ed., New York, NY: Springer New York, 1990, pp. 27–35.

[95] T. Chou and C. Orlandi, "The simplest protocol for oblivious transfer," in *LATINCRYPT*, 2015.

[96] N. Dottling, S. Ghosh, J. B. Nielsen, T. Nilges, and R. Trifiletti, "Tinyole: Efficient actively secure two-party computation from oblivious linear function evaluation," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA: Association for Computing Machinery, 2017, ISBN: 9781450349468. DOI: 10.1145/3133956.3134024. [Online]. Available: https://doi.org/10.1145/3133956.3134024.

[97] S. Ghosh, J. B. Nielsen, and T. Nilges, "Maliciously secure oblivious linear function evaluation with constant overhead," in *Advances in Cryptology – ASIACRYPT 2017*, T. Takagi and T. Peyrin, Eds., Cham: Springer International Publishing, 2017, pp. 629–659, ISBN: 978-3-319-70694-8.

[98] M. Keller, E. Orsini, and P. Scholl, "Mascot: Faster malicious arithmetic secure computation with oblivious transfer," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16, Vienna, Austria: Association for Computing Machinery, 2016, 830â 842, ISBN: 9781450341394. DOI: 10.1145/2976749.2978357. [Online]. Available: https://doi.org/10.1145/2976749.2978357.

[99] N. Döttling, D. Kraschewski, and J. Müller-Quade, "David & goliath oblivious affine function evaluation-asymptotically optimal building blocks for universally composable two-party computation from a single untrusted stateful tamper-proof hardware token.,"

[100] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology – CRYPTO 2012*, R. Safavi-Naini and R. Canetti, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 643–662, ISBN: 978-3-642-32009-5.

[101] A. Banerjee and C. Peikert, "New and improved key-homomorphic pseudorandom functions," in *Annual Cryptology Conference*, Springer, 2014, pp. 353–370.

[102] A. Adelsbach and A.-R. Sadeghi, "Zero-knowledge watermark detection and proof of ownership," in *Information Hiding*, 2001.

[103] F. Cayre, C. Fontaine, and T. Furon, "Watermarking security: Theory and practice," *IEEE Transactions on Signal Processing*, vol. 53, no. 10, pp. 3976–3987, 2005.

[104] M. Luo and A. G. Bors, "Surface-preserving robust watermarking of 3-d shapes," *IEEE Transactions on Image Processing*, vol. 20, no. 10, pp. 2813–2826, 2011. DOI: 10.1109/TIP.2011.2142004.

[105] R. Namba and J. Sakuma, "Robust watermarking of neural network with exponential weighting," in *2019 ACM Asia CCS*, ser. Asia CCS '19, Association for Computing Machinery, 2019, pp. 228–240, ISBN: 9781450367523. [Online]. Available: https://doi.org/10.1145/3321705.3329808.

[106] M. Noorkami and R. M. Mersereau, "A framework for robust watermarking of h.264-encoded video with controllable detection performance," *IEEE Transactions on Information Forensics and Security*, vol. 2, no. 1, pp. 14–23, 2007. DOI: 10.1109/TIFS.2006.890306.

[107] P. Rogaway, "Formalizing human ignorance," in *Progress in Cryptology - VIETCRYPT 2006*, P. Q. Nguyen, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 211–228, ISBN: 978-3-540-68800-6.

[108] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, 2008.

[109] *Chain link*, https://chain.link/.

[110] L. G. Valiant, "Short monotone formulae for the majority function," *Journal of Algorithms*, vol. 5, no. 3, pp. 363–366, 1984.

[111] S. Hoory, A. Magen, and T. Pitassi, "Monotone circuits for the majority function," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, Springer, 2006, pp. 410–425.

[112] R. B. Boppana, "Amplification of probabilistic boolean formulas," in *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, 1985, pp. 20–29.

[113] O. Goldreich, "Valiant?s polynomial-size monotone formula for majority," 2011. eprint: http://www.wisdom.weizmann.ac.il/~oded/PDF/mono-maj.pdf.

[114] "Ethereum Website." https://www.ethereum.org/. (n.d.).

[115] B. Chor, A. Fiat, and M. Naor, "Tracing traitors," in *CRYPTO*, 1994.

[116] D. Boneh and M. Franklin, "An efficient public key traitor tracing scheme," in *CRYPTO*, 1999.

[117] A. Kiayias and Q. Tang, "Traitor deterring schemes: Using bitcoin as collateral for digital content," in *ACM CCS*, 2015.

[118] N. Memon and P. W. Wong, "A buyer-seller watermarking protocol," *IEEE Transactions on Image Processing*, vol. 10, no. 4, pp. 643–649, 2001, ISSN: 1057-7149. DOI: 10.1109/83.913598.

[119] A. Adelsbach and A.-R. Sadeghi, "Zero-knowledge watermark detection and proof of ownership," in *Information Hiding*, I. S. Moskowitz, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 273–288, ISBN: 978-3-540-45496-0.

[120] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *IEEE Symposium on Security and Privacy*, 2016.

[121] Z. A. Genc, V. Iovino, and A. Rial, "The simplest protocol for oblivious transfer revisited," 2017. [Online]. Available: https://eprint.iacr.org/2017/370.pdf.

[122] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Proceedings on Advances in cryptology—CRYPTO '86*, 1987, pp. 186–194.

[123] J. Camenisch and M. Stadler, "Proof systems for general statements about discrete logarithms," *Technical report/Dept. of Computer Science, ETH Zürich*, vol. 260, 1997.

[124] T. Härder and A. Bühmann, "Database caching – towards a cost model for populating cache groups," in *Advances in Databases and Information Systems*, A. Benczúr, J. Demetrovics, and G. Gottlob, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 215–229.

[125] S. B. Yao, "An attribute based model for database access cost analysis," *ACM Trans. Database Syst.*, vol. 2, no. 1, pp. 45–67, Mar. 1977, ISSN: 0362-5915. DOI: 10.1145/320521.320535. [Online]. Available: http://doi.acm.org/10.1145/320521.320535.

[126] P. Meerwald, *Watermarking source code*, Online, 2005. [Online]. Available: http://www.cosy.sbg.ac.at/~pmeerw/Watermarking.

[127] I. J. Cox, J. Kilian, F. T. Leighton, and T. Shamoon, "Secure spread spectrum watermarking for multimedia," *IEEE TIP*, vol. 6, no. 12, pp. 1673–1687, 1997.

[128] I. Amer, T. Sheha, W. Badawy, and G. Jullien, "A tool for robustness evaluation of image watermarking algorithms," in *Advanced Techniques in Computing Sciences and Software Engineering*, K. Elleithy, Ed., Dordrecht: Springer Netherlands, 2010, pp. 59–63, ISBN: 978-90-481-3660-5.

[129] B. Y. Lei, I. Y. Soon, and Z. Li, "Blind and robust audio watermarking scheme based on svd–dct," *Signal Processing*, vol. 91, no. 8, pp. 1973–1984, 2011, ISSN: 0165-1684. DOI: https://doi.org/10.1016/j.sigpro.2011.03.001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0165168411000764.

[130] W.-N. Lie and L.-C. Chang, "Robust and high-quality time-domain audio watermarking based on low-frequency amplitude modification," *IEEE Transactions on Multimedia*, vol. 8, no. 1, pp. 46–59, 2006, ISSN: 1520-9210. DOI: 10.1109/TMM.2005.861292.

[131] Y. Erfani and S. Siahpoush, "Robust audio watermarking using improved ts echo hiding," *Digital Signal Processing*, vol. 19, no. 5, pp. 809–814, 2009, ISSN: 1051-2004. DOI: https://doi.org/10.1016/j.dsp.2009.04.003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1051200409000426.

[132] R. Lancini, F. Mapelli, and S. Tubaro, "A robust video watermarking technique in the spatial domain," in *International Symposium on VIPromCom Video/Image Processing and Multimedia Communications*, 2002, pp. 251–256. DOI: 10.1109/VIPROM.2002.1026664.

[133] J. Zhang, A. T. S. Ho, G. Qiu, and P. Marziliano, "Robust video watermarking of h.264/avc," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 2, pp. 205–209, Feb. 2007, ISSN: 1549-7747. DOI: 10.1109/TCSII.2006.886247.

[134] R. Venkatesan, V. Vazirani, and S. Sinha, "A graph theoretic approach to software watermarking," in *Information Hiding*, I. S. Moskowitz, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 157–168, ISBN: 978-3-540-45496-0.

[135] S. Kim and D. J. Wu, "Watermarking cryptographic functionalities from standard lattice assumptions," in *Advances in Cryptology – CRYPTO 2017*, J. Katz and H. Shacham, Eds., Cham: Springer International Publishing, 2017, pp. 503–536.

[136] "Relic: Efficient library for cryptography." (n.d.).

[137] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Adaptive security for threshold cryptosystems," in *Annual International Cryptology Conference*, Springer, 1999, pp. 98–116.

[138] S. Jarecki and A. Lysyanskaya, "Adaptively secure threshold cryptography: Introducing concurrency, removing erasures," in *International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2000, pp. 221–242.

[139] K. Chalkias and G. Stephanides, "Timed release cryptography from bilinear pairings using hash chains," English, in *Communications and Multimedia Security*, ser. Lecture Notes in Computer Science, H. Leitold and E. Markatos, Eds., vol. 4237, Springer Berlin Heidelberg, 2006, pp. 130–140, ISBN: 978-3-540-47820-1.

[140] H.-C. Chen and R. Deviani, "A secure e-voting system based on rsa time-lock puzzle mechanism.," in *BWCCA*, IEEE, 2012, pp. 596–601, ISBN: 978-1-4673-2972-9. [Online]. Available: http://dblp.uni-trier.de/db/conf/bwcca/bwcca2012.html#ChenD12.

[141] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," English, in *Advances in Cryptology â CRYPTOâ 92*, ser. Lecture Notes in Computer Science, E. Brickell, Ed., vol. 740, Springer Berlin Heidelberg, 1993, pp. 139–147, ISBN: 978-3-540-57340-1.

[142] M. Jakobsson and A. Juels, "Proofs of work and bread pudding protocols(extended abstract)," English, in *Secure Information Networks*, ser. IFIP â The International Federation for Information Processing, B. Preneel, Ed., vol. 23, Springer US, 1999, pp. 258–272, ISBN: 978-1-4757-6487-1.

[143] D. Boneh and M. Naor, "Timed commitments," English, in *Advances in Cryptology â CRYPTO 2000*, ser. Lecture Notes in Computer Science, M. Bellare, Ed., vol. 1880, Springer Berlin Heidelberg, 2000, pp. 236–254, ISBN: 978-3-540-67907-3.

[144] J. A. Garay and M. Jakobsson, "Timed release of standard digital signatures," in *FC'02: Proceedings of the 6th international conference on Financial cryptography*, Southampton, Bermuda: Springer-Verlag, 2003, pp. 168–182. [Online]. Available: http://www.markus-jakobsson.com/wp-content/uploads/timed-release.pdf.

[145] J. A. Garay and C. Pomerance, "Timed fair exchange of standard signatures," English, in *Financial Cryptography*, ser. Lecture Notes in Computer Science, R. Wright, Ed., vol. 2742, Springer Berlin Heidelberg, 2003, pp. 190–207, ISBN: 978-3-540-40663-1.

[146] M. Bellare and S. Goldwasser, "Encapsulated key escrow," University of California, San Diego, Tech. Rep., 1996.

[147] M. Bellare and S. Goldwasser, "Verifiable partial key escrow," in *Proceedings of the 4th ACM Conference on Computer and Communications Security*, ser. CCS '97, Zurich, Switzerland: ACM, 1997, pp. 78–91, ISBN: 0-89791-912-2. DOI: 10.1145/266420.266439. [Online]. Available: http://doi.acm.org/10.1145/266420.266439.

[148] B. Applebaum, "Randomized encoding of functions," English, in *Cryptography in Constant Parallel Time*, ser. Information Security and Cryptography, Springer Berlin Heidelberg, 2014, pp. 19–31, ISBN: 978-3-642-17366-0.

[149] N. Bitansky, S. Goldwasser, A. Jain, O. Paneth, V. Vaikuntanathan, and B. Waters, *Time-lock puzzles from randomized encodings*, Cryptology ePrint Archive, Report 2015/514, 2015.

[150] Y. Watanabe and J. Shikata, "Timed-release secret sharing scheme with information theoretic security," *CoRR*, vol. abs/1401.5895, 2014. [Online]. Available: http://arxiv.org/abs/1401.5895.

[151] D. Boneh and M. K. Franklin, "Identity-based encryption from the weil pairing," in *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '01, London, UK, UK: Springer-Verlag, 2001, pp. 213–229, ISBN: 3-540-42456-3. [Online]. Available: http://dl.acm.org/citation.cfm?id=646766.704155.

[152] M. C. Mont, K. Harrison, and M. Sadler, *The hp time vault service: Innovating the way confidential information is disclosed, at the right time*, 2002.

[153] J. Y. Halpern and V. Teague, "Rational secret sharing and multiparty computation: Extended abstract," *CoRR*, vol. abs/cs/0609035, 2006. [Online]. Available: http://arxiv.org/abs/cs/0609035.

[154] S. D. Gordon and J. Katz, *Rational secret sharing, revisited*, Cryptology ePrint Archive, Report 2006/142, 2006.

[155] M. Lepinski, S. Micali, C. Peikert, and A. Shelat, "Completely fair sfe and coalition-safe cheap talk," in *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '04, St. John's, Newfoundland, Canada: Association for Computing Machinery, 2004, p. 110, ISBN: 1581138024. DOI: 10.1145/1011767.1011769. [Online]. Available: https://doi.org/10.1145/1011767.1011769.

[156] J. Alwen, J. Katz, Y. Lindell, G. Persiano, a. shelat abhi, and I. Visconti, "Collusion-free multiparty computation in the mediated model," in *Advances in Cryptology - CRYPTO 2009*, S. Halevi, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 524–540, ISBN: 978-3-642-03356-8.

[157] J. Alwen, A. Shelat, and I. Visconti, "Collusion-free protocols in the mediated model," in *Advances in Cryptology – CRYPTO 2008*, D. Wagner, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 497–514, ISBN: 978-3-540-85174-5.

[158] Y. L. Michele Ciampi and V. Zikas, *Collusion-preserving computation without a mediator*, Cryptology ePrint Archive, Report 2020/497, 2020.

[159] R. B. Myerson, *Game theory*. Harvard university press, 2013.

[160] S. Kuhn, "Prisonerâ s dilemma," 1997.

[161] J. C. Harsanyi, "Games with incomplete information played by bayesian players, i–iii part i. the basic model," *Management science*, vol. 14, no. 3, pp. 159–182, 1967.

[162] A. Kawachi, Y. Okamoto, K. Tanaka, and K. Yasunaga, *General constructions of rational secret sharing with expected constant-round reconstruction*, Cryptology ePrint Archive, Report 2013/874, 2013.

[163] T. Toft *et al.*, "Primitives and applications for multi-party computation," *Unpublished doctoral dissertation, University of Aarhus, Denmark*, 2007.

[164] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft, "Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation," in *Theory of Cryptography*, S. Halevi and T. Rabin, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 285–304, ISBN: 978-3-540-32732-5.

[165] J. A. Akinyele, C. Garman, I. Miers, *et al.*, "Charm: A framework for rapidly prototyping cryptosystems," *Journal of Cryptographic Engineering*, pp. 111–128, 2013. [Online]. Available: http://dx.doi.org/10.1007/s13389-013-0057-3.

[166] O. Catrina and S. De Hoogh, "Improved primitives for secure multiparty integer computation," in *International Conference on Security and Cryptography for Networks*, Springer, 2010, pp. 182–199.

[167] K. Deb, M. S. Al-Seraj, M. M. Hoque, and M. I. H. Sarkar, "Combined dwt-dct based digital image watermarking technique for copyright protection," in *2012 7th International Conference on Electrical and Computer Engineering*, 2012, pp. 458–461.

[168] S. Das, *DWT-DCT-Digital-Image-Watermarking*, https://github.com/diptamath/DWT-DCT-Digital-Image-Watermarking.

[169] M. E. Hellman, "An overview of public key cryptography," *IEEE Communications Magazine*, vol. 40, no. 5, pp. 42–49, 2002.

[170] N. Atzei, M. Bartoletti, S. Lande, and R. Zunino, "A formal model of bitcoin transactions," in *Financial Cryptography and Data Security*, S. Meiklejohn and K. Sako, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2018, pp. 541–560, ISBN: 978-3-662-58387-6.

[171] P. Das, S. Faust, and J. Loss, "A formal treatment of deterministic wallets," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19, London, United Kingdom: Association for Computing Machinery, 2019, pp. 651–668, ISBN: 9781450367479. DOI: 10.1145/3319535.3354236. [Online]. Available: https://doi.org/10.1145/3319535.3354236.

[172] *Custodial vs. non-custodial wallets*, https://www.gemini.com/cryptopedia/crypto-wallets-custodial-vs-noncustodial, 2021.

[173] *Sok: A taxonomy of cryptocurrency wallets*, https://eprint.iacr.org/2020/868.pdf, 2020.

[174] *Fortune - nearly 4 million bitcoins lost forever*, https://fortune.com/2017/11/25/lost-bitcoins/.

[175] *Poloniex loses 12.3pc of its bitcoins in latest bitcoin exchange hack*, https://www.coindesk.com/markets/2014/03/05/poloniex-loses-123-of-its-bitcoins-in-latest-bitcoin-exchange-hack/, 2014.

[176] *Details of $5 million bitstamp hack revealed*, https://www.coindesk.com/markets/2015/07/01/details-of-5-million-bitstamp-hack-revealed/, 2015.

[177] *Chinese bitcoin exchange okex hacked for $3 mln, police not interested*, https://cointelegraph.com/news/chinese-bitcoin-exchange-okex-hacked-for-3-mln-police-not-interested, 2017.

[178] S. Kim, A. Sarin, and D. Virdi, "Crypto-assets unencrypted," *Journal of Investment Management, Forthcoming*, 2018.

[179] V. Shoup, "Practical threshold signatures," in *International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2000, pp. 207–220.

[180] S. Barber, X. Boyen, E. Shi, and E. Uzun, "Bitter to better — how to make bitcoin a better currency," in *Financial Cryptography and Data Security*, A. D. Keromytis, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 399–414, ISBN: 978-3-642-32946-3.

[181] A. Marcedone, R. Pass, and A. Shelat, "Minimizing trust in hardware wallets with two factor signatures," in *Financial Cryptography and Data Security*, I. Goldberg and T. Moore, Eds., Cham: Springer International Publishing, 2019, pp. 407–425, ISBN: 978-3-030-32101-7.

[182] X. He, J. Lin, K. Li, and X. Chen, "A novel cryptocurrency wallet management scheme based on decentralized multi-constrained derangement," *IEEE Access*, vol. 7, pp. 185 250–185 263, 2019. DOI: 10.1109/ACCESS.2019.2961183.

[183] M. Drijvers, K. Edalatnejad, B. Ford, *et al.*, "On the security of two-round multi-signatures," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1084–1101. DOI: 10.1109/SP.2019.00050.

[184] *Multisig wallet security*, https://medium.com/the-capital/multisig-wallet-security-e2a1dee95cc0, 2021.

[185] *Multisig wallets explained*, https://medium.com/block-journal/multi-sig-wallets-explained-5544c122a1de, 2019.

[186] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, no. 1, pp. 36–63, 2001.

[187] M. Bellare and G. Neven, "Identity-based multi-signatures from rsa," in *Cryptographers' Track at the RSA Conference*, Springer, 2007, pp. 145–162.

[188] G. Bleumer, "Threshold signature," in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg, Ed. Boston, MA: Springer US, 2005, pp. 611–614, ISBN: 978-0-387-23483-0. DOI: 10.1007/0-387-23483-7_429. [Online]. Available: https://doi.org/10.1007/0-387-23483-7_429.

[189] *Refresh when you wake up: Proactive threshold wallets with offline devices*, https://arpa.medium.com/threshold-signature-explained-brining-exciting-apps-with-tss-8a75b43e19bf.

[190] *Attacking threshold wallet*, https://eprint.iacr.org/2020/1052.pdf, 2020.

[191] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979, ISSN: 0001-0782. DOI: 10.1145/359168.359176. [Online]. Available: https://doi.org/10.1145/359168.359176.

[192] A. Beimel, "Secret-sharing schemes: A survey," in *International conference on coding and cryptology*, Springer, 2011, pp. 11–46.

[193] R. Gennaro, S. Goldfeder, and A. Narayanan, "Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security," in *International Conference on Applied Cryptography and Network Security*, Springer, 2016, pp. 156–174.

[194] *Why threshold signature wallets are better than multisig: Top 5 reasons*, https://sepior.com/blog/top-5-reasons-threshold-signature-wallets-are-better-than-multisig.

[195] P. L. Seijas, S. J. Thompson, and D. McAdams, "Scripting smart contracts for distributed ledger technology.," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 1156, 2016.

[196] A. Voskobojnikov, B. Obada-Obieh, Y. Huang, and K. Beznosov, "Surviving the cryptojungle: Perception and management of risk among north american cryptocurrency (non)users," in *Financial Cryptography*, 2020.

[197] *Holistic privacy and usability of a cryptocurrency wallet*, https://arxiv.org/pdf/2105.02793.pdf/.

[198] C. Sas and I. E. Khairuddin, "Design for trust: An exploration of the challenges and opportunities of bitcoin users," *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2017.

[199] A. Mai, K. Pfeffer, M. Gusenbauer, E. Weippl, and K. Krombholz, "User mental models of cryptocurrency systems - a grounded theory approach," in *SOUPS @ USENIX Security Symposium*, 2020.

[200] S. Abramova, A. Voskobojnikov, K. Beznosov, and R. Böhme, "Bits under the mattress: Understanding different risk perceptions and security behaviors of crypto-asset users," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–19.

[201] M. Fröhlich, P. Hulm, and F. Alt, "Under pressure. a user-centered threat model for cryptocurrency owners," 2021.

[202] S. Ghesmati, W. Fdhila, and E. Weippl, "User-perceived privacy in blockchain," *Cryptology ePrint Archive*, 2022.

[203] M. G, "Coinjoin: Bitcoin privacy for the real world," 2013. [Online]. Available: https://bitcointalk.org/index.php(2013).

[204] T. Ruffing, P. Moreno-Sanchez, A. Kate, and J. Vaidya, "Coinshuffle: Practical decentralized coin mixing for bitcoin," in *Computer Security - ESORICS 2014*, 2014.

[205] T. Ruffing and P. Moreno-Sanchez, "Valueshuffle: Mixing confidential transactions for comprehensive transaction privacy inâ bitcoin," in *Financial Cryptography and Data Security*, 2017.

[206] M. G, "Coinswap: Transaction graph disjoint trustless trading," 2013. [Online]. Available: https://bitcointalk.org/index.php%20(2013).

[207] C. Herley and P. Van Oorschot, "A research agenda acknowledging the persistence of passwords," *IEEE Security & privacy*, vol. 10, no. 1, pp. 28–36, 2011.

[208] *Metamask wallet*, https://metamask.io/.

[209] *Robinhood crypto*, https://robinhood.com/us/en/about/crypto/.

[210] *Bitgo*, https://www.bitgo.com/.

[211] S. L. Garfinkel and R. C. Miller, "Johnny 2: A user test of key continuity management with s/mime and outlook express," in *Proceedings of the 2005 Symposium on Usable Privacy and Security*, ser. SOUPS '05, Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2005, pp. 13–24, ISBN: 1595931783. DOI: 10.1145/1073001.1073003. [Online]. Available: https://doi.org/10.1145/1073001.1073003.

[212] S. Gaw, E. W. Felten, and P. Fernandez-Kelly, "Secrecy, flagging, and paranoia: Adoption criteria in encrypted email," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '06, Montréal, Québec, Canada: Association for Computing Machinery, 2006, pp. 591–600, ISBN: 1595933727. DOI: 10.1145/1124772.1124862. [Online]. Available: https://doi.org/10.1145/1124772.1124862.

[213] S. Sheng, L. Broderick, C. A. Koranda, and J. J. Hyland, "Why johnny still can't encrypt: Evaluating the usability of email encryption software," in *Symposium On Usable Privacy and Security*, ACM, 2006, pp. 3–4.

[214] T. Bui, S. P. Rao, M. Antikainen, and T. Aura, "Pitfalls of open architecture: How friends can exploit your cryptocurrency wallet," in *Proceedings of the 12th European Workshop on Systems Security*, 2019, pp. 1–6.

[215] R. Gennaro and S. Goldfeder, "Fast multiparty threshold ecdsa with fast trustless setup," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1179–1194.

[216] Y. Lindell and A. Nof, "Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18, Toronto, Canada: Association for Computing Machinery, 2018, pp. 1837–1854, ISBN: 9781450356930. DOI: 10.1145/3243734.3243788. [Online]. Available: https://doi.org/10.1145/3243734.3243788.

[217] W. Dai, J. Deng, Q. Wang, C. Cui, D. Zou, and H. Jin, "Sblwt: A secure blockchain lightweight wallet based on trustzone," *IEEE Access*, vol. 6, pp. 40 638–40 648, 2018. DOI: 10.1109/ACCESS.2018.2856864.

[218] H. Rezaeighaleh and C. C. Zou, "Deterministic sub-wallet for cryptocurrencies," in *2019 IEEE International Conference on Blockchain (Blockchain)*, 2019, pp. 419–424. DOI: 10.1109/Blockchain.2019.00064.

[219] K. Owens, O. Anise, A. Krauss, and B. Ur, "User perceptions of the usability and security of smartphones as {fido2} roaming authenticators," in *Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021)*, 2021, pp. 57–76.

[220] *Prolific participants*, https://www.prolific.co/#check-sample.

[221] S. Albakry, K. Vaniea, and M. K. Wolters, "What is this url's destination? empirical evaluation of users' url reading," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, ser. CHI '20, Honolulu, HI, USA: Association for Computing Machinery, 2020, pp. 1–12, ISBN: 9781450367080. DOI: 10.1145/3313831.3376168. [Online]. Available: https://doi.org/10.1145/3313831.3376168.

[222] S. Ghorbani Lyastani, M. Schilling, M. Neumayr, M. Backes, and S. Bugiel, "Is fido2 the kingslayer of user authentication? a comparative usability study of fido2 passwordless authentication," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 268–285. DOI: 10.1109/SP40000.2020.00047.

[223] *Educational attainment in the united states: 2020*, https://www.census.gov/data/tables/2020/demo/educational-attainment/cps-detailed-tables.html.

[224] P. Hitlin, "Turkers in this canvassing: Young, well-educated and frequent users," in *Research in the Crowdsourcing Age, a Case Study*, 2016.

[225] M. L. McHugh, "Interrater reliability: The kappa statistic," *Biochemia medica*, vol. 22, no. 3, pp. 276–282, 2012. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/23092060.

[226] R. L. Plackett, "Karl pearson and the chi-squared test," *International Statistical Review / Revue Internationale de Statistique*, vol. 51, no. 1, pp. 59–72, 1983, ISSN: 03067734, 17515823. [Online]. Available: http://www.jstor.org/stable/1402731.

[227] "Smooth tests of goodness of fit: An overview," *International Statistical Review / Revue Internationale de Statistique*, vol. 58, no. 1, pp. 9–17, 1990, ISSN: 03067734, 17515823. [Online]. Available: http://www.jstor.org/stable/1403470.

[228] T. W. MacFarland and J. M. Yates, "Mann–whitney u test," in *Introduction to Nonparametric Statistics for the Biological Sciences Using R*. Cham: Springer International Publishing, 2016, pp. 103–132, ISBN: 978-3-319-30634-6. DOI: 10.1007/978-3-319-30634-6_4. [Online]. Available: https://doi.org/10.1007/978-3-319-30634-6_4.

[229] G. Rupert Jr *et al.*, *Simultaneous statistical inference*. Springer Science & Business Media, 2012.

[230] D. di Prisco and D. Strangio, "Technology and financial inclusion: A case study to evaluate potential and limitations of blockchain in emerging countries," *Technology Analysis & Strategic Management*, vol. 0, no. 0, pp. 1–14, 2021.

[231] D. Shin and W. T. Bianco, "In blockchain we trust: Does blockchain itself generate trust?" *Social Science Quarterly*, vol. 101, no. 7, pp. 2522–2538, 2020.

[232] S. Bellman, E. J. Johnson, and G. L. Lohse, "On site: To opt-in or opt-out? it depends on the question," *Communications of the ACM*, vol. 44, no. 2, pp. 25–27, 2001.

[233] J. P. Kesan and R. C. Shah, "Setting software defaults: Perspectives from law, computer science and behavioral economics," *Notre Dame L. Rev.*, vol. 82, p. 583, 2006.

[234] K. M. Ramokapane, A. C. Mazeli, and A. Rashid, "Skip, skip, skip, accept!!!: A study on the usability of smartphone manufacturer provided default features and user privacy," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 2, pp. 209–227, 2019. DOI: doi:10.2478/popets-2019-0027. [Online]. Available: https://doi.org/10.2478/popets-2019-0027.

[235] D. of the Prime Minister and A. G. Cabinet, "Harnessing the power of defaults," [Online]. Available: https://behaviouraleconomics.pmc.gov.au/sites/default/files/resources/harnessing-power-defaults.pdf.

[236] K. Eldefrawy, S. Hwang, R. Ostrovsky, and M. Yung, "Communication-efficient (proactive) secure computation for dynamic general adversary structures and dynamic groups," in *Security and Cryptography for Networks(SCN)*, C. Galdi and V. Kolesnikov, Eds., ser. Lecture Notes in Computer Science, vol. 12238, Springer, 2020, pp. 108–129.

[237] M. Hirt and D. Tschudi, "Efficient general-adversary multi-party computation," in *Advances in Cryptology - ASIACRYPT*, K. Sako and P. Sarkar, Eds., ser. Lecture Notes in Computer Science, vol. 8270, Springer, 2013, pp. 181–200.

[238] M. Fitzi, M. Hirt, and U. M. Maurer, "General adversaries in unconditional multi-party computation," in *Advances in Cryptology - ASIACRYPT*, K. Lam, E. Okamoto, and C. Xing, Eds., ser. Lecture Notes in Computer Science, vol. 1716, Springer, 1999, pp. 232–246.

[239] J. Lampkins and R. Ostrovsky, "Communication-efficient MPC for general adversary structures," in *Security and Cryptography for Networks (SCN)*, M. Abdalla and R. D. Prisco, Eds., ser. Lecture Notes in Computer Science, vol. 8642, Springer, 2014, pp. 155–174.

[240] M. Al-Rubaie and J. M. Chang, "Privacy-preserving machine learning: Threats and solutions," *IEEE Security & Privacy*, vol. 17, no. 2, pp. 49–58, 2019.

[241] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 19–38. DOI: 10.1109/SP.2017.12.

[242] J. Doerner, Y. Kondi, E. Lee, and A. Shelat, "Threshold ecdsa from ecdsa assumptions: The multiparty case," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1051–1066. DOI: 10.1109/SP.2019.00024.

[243] *The transport layer security (tls) protocol*, https://tools.ietf.org/html/rfc8446.

[244] A. J. Menezes, J. Katz, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1996.

[245] B. Libert, D. Stehl, and R. Titiu, "Adaptively secure distributed prfs from lwe," in *Theory of Cryptography Conference*, Springer, 2018, pp. 391–421.

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

[247] T. Rabin and M. Ben-Or, "Verifiable secret sharing and multiparty protocols with honest majority," in *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, 1989, pp. 73–85.

[248] M. Stadler, "Publicly verifiable secret sharing," in *Advances in Cryptology — EUROCRYPT '96*, U. Maurer, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 190–199, ISBN: 978-3-540-68339-1.

[249] R. Thorbek, "Proactive linear integer secret sharing," 2009. arXiv: https://eprint.iacr.org/2009/183.pdf.

[250] I. G. Aniket Kate Yizhou Huang, "Distributed key generation in the wild." [Online]. Available: https://eprint.iacr.org/2012/377.pdf.

[251] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual international cryptology conference*, Springer, 1991, pp. 129–140.

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

[253] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl, "Asynchronous verifiable secret sharing and proactive cryptosystems," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ser. CCS '02, Washington, DC, USA: Association for Computing Machinery, 2002, pp. 88–97, ISBN: 1581136129. DOI: 10.1145/586110.586124.

[254] *Tendermint*, http://Tendermint.com.

[255] M. R. Albrecht, R. Player, and S. Scott, *On the concrete hardness of learning with errors*, Cryptology ePrint Archive, Report 2015/046, https://eprint.iacr.org/2015/046, 2015.