Collusion-Deterrent Threshold Information Escrow

Easwar Vivek Mangipudi   Donghang Lu   Aniket Kate
{evmangipudi,lu562,aniket}@purdue.edu
Purdue University

ABSTRACT

Timed-release encryption (TRE) is a prominent distributed way for sending messages to the future. Beyond its applications to e-voting and auctions, TRE can be easily generalized to a threshold information escrow (TIE) service, where a user can encrypt her message to any *condition* instead of just expiration of time as in TRE. Nevertheless, TRE and by extension TIE realized using threshold escrow agents is vulnerable to premature, selective, and undetectable unlocking of messages through *collusion* among curious agents offering the service. This work presents a novel provably secure TIE scheme where any collusion attempt among the escrow agents offering the service towards premature decryption results in penalization through a loss of cryptocurrency and getting banned from the system.

The proposed collusion-deterrent escrow (CDE) scheme introduces a novel incentive-penalty mechanism using a user-induced information asymmetry among the agents such that they stay honest until the user-specified condition for decryption is met. In particular, each agent makes an escrow deposit before the start of the protocol such that the cryptocurrency deposit amount is transferred back to the agent when the condition specified by the user is met or can be transferred by anyone who holds the secret key corresponding to the public key of the protocol instance. CDE offers information escrow as a service and ensures that whenever the agents collude to decrypt the user data before the condition is met, there would be at least one whistle-blower agent who can withdraw/transfer the deposits of all other agents thereby penalizing them. We analyse the CDE protocol and model collusion as a game induced among rational agents offering the service and show in game-theoretic terms that the agents do not collude at equilibrium. We also present a prototype implementation of the CDE protocol and demonstrate its efficiency towards use in practice. We find this work to be an important step towards weakening the strong non-collusion assumptions across multi-party computation applications.

1 INTRODUCTION

Timed-release encryption (TRE) [25, 30, 55, 63] involves encrypting a message for a specific time period such that the message cannot be retrieved before the period ends. TRE allows sending messages "to the future", and has been found to be useful in different scenarios including e-voting [23], sealed-bid auctions [22] and client puzzles in the challenge-response systems [36]. In the cryptographic literature, the two prominent ways of implementing a TRE scheme are time-lock puzzles and distributed/threshold escrows.

In (computation-based) time-lock puzzles [14, 54, 63], a predefined number of computation steps need to be performed to solve the puzzle and obtain a secret. Here, the puzzle creator is expected to estimate the required time-release period in the form of the number of computation steps. However, time-lock puzzles suffer from the fact that the expected time of computation need not remain the same into the future owing to various factors like computational and technological advancements and evolving adversary. Indeed, Rivest's time-lock puzzle LCS35 [64] has been an overestimation [40] and got solved 15 years before the expected time recently by an architect called Frank Gehry [4]. In TRE based on distributed/threshold escrow [15, 24, 25, 30, 60, 63, 71], the input message is shared among a group of agents through a suitable distributed cryptographic protocol such that a threshold number of agents are expected to combine their shares to open the message when the timed-release period expires. Here, the secrecy of the message is maintained before the timed-release condition is met as long as a lower than the threshold number of agents get compromised. While this threshold-bounded adversary assumption looks reasonable for many applications of distributed cryptography (e.g., Random Beacon protocols [1], threshold signatures wallets), it is indeed a strong assumption for TRE given the longevity of its usages, possibly spanning years.

TRE can be generalized to even wider applications: Instead of encrypting the data until a time period expires, the user can encrypt data until a certain 'condition' is met. This condition can be anything of user's choice, for example, the value of shares of a certain company hitting a certain value, temperatures rising to a certain level to release funds for environmental programs, allegation escrows [8, 47, 61] etc. When the encryption is to a certain general condition, we call it threshold information escrow (TIE) [9], making TRE a special case of TIE. As we see later (expanded in Section 3.3), this condition can be anything that can be checked by a program (or smart contract). Offering information escrows, especially software escrows is a prevalent practice in the industry with companies like Escrowtech [2], Iron Mountain [3] providing software escrow services to technological companies which routinely appear in Fortune 500 list. However, firms offering the service act as trusted third parties for the service.

Unlike most other applications of threshold cryptography [16, 29, 31], for TIE, the trust of the distributed (escrow) agents may have to last for several years. If the escrow agents running a TIE service decide to collude and selectively open a message from a particular user, they can do it passively in an undetectable manner and without affecting the secrecy of others' messages. Maintaining the non-collusion assumption among escrow agents *continuously* over a long duration of time is indeed a challenge. As the agents may collude among themselves in an undetectable manner, it is difficult if not impossible to prevent such a collusion in longevity; unless the protocol design itself makes collusion non-profitable. In this work, we aim to design such a protocol where the best strategy for any rational agent is to not collude.

Contributions. In the form of *collusion deterrent escrow* (CDE), this work offers a novel 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 dis-incentivizes 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 [15, 24, 37, 38, 59, 71] and overcoming it has been a significant barrier for the community for a long time. To the best of our knowledge, this is the first work where collusion among the multi-party computation agents is being addressed though dis-incentivizing the agents instead of just binding them with the *non-collusion assumption*.

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 collusions. 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 of the agents.

We define our collusion deterrent escrow concept as an ideal functionality $\mathcal{F}_{CDE}$; towards realizing it, we propose a cryptographic primitive called distributed receiver oblivious transfer (DROT). DROT is a distributed version of the two party oblivious transfer [31] protocol, where multiple receivers share the choice bit in a threshold manner. Our CDE protocol employs a novel combination of DROT, robust bit watermarking, distributed key generation, and secure bit decomposition to securely realize $\mathcal{F}_{CDE}$. The protocol supports any condition that can be checked through a cryptocurrency smart contract (even through interaction with real world) as a condition of data release in the protocol.

We provide an implementation of the CDE protocol using SCALE-MAMBA [26] and HoneybadgerMPC [52] frameworks. Our prototype implementation shows that the system realizing the CDE takes less than a minute to setup and $\sim$ 120 milliseconds of interaction to transfer key shares to the agents offering the service.

1.1 Organization of the paper

Section 2 describes the system setup, problem definition and gives an overview of the solution. Section 3 introduces the different multi-party computation modules,robust watermarking, and a claim-or-refund smart contract required to realize the collusion deterrent escrow (CDE) protocol. Section 4 describes the different steps and algorithms of the proposed CDE protocol, and Section 5 models the CDE protocol as a mechanism inducing a game among the agents to show that the agents do not collude while playing their best response strategies. Section 6 offers the security definition, Section 7 provides the implementation details and the various times taken for different steps of the protocol. The related work and conclusion are discussed in sections 8, 9.

2 SYSTEM SETUP AND SOLUTION OVERVIEW

2.1 System Setup

The system consists of n agents who offer the information escrow service and a user engaging the service, in a multi-party computation (MPC) setting. All the communication is over secure and authenticated channels. We consider a *mixed-behaviour* model [53] where the agents are either *rational or malicious*, the rational agents aim to maximize their utility at any given point of time, the malicious ones can deviate from the protocol arbitrarily. However, no more than t agents can be malicious at any time. 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.

Adversary Model. We consider a $t-$bounded adversary under a *static corruption* setting. Up-to t of n parties can be corrupted before the start of the protocol and the corrupted parties remain corrupted throughout the protocol. We consider a *malicious* adversary who can make the corrupted parties deviate arbitrarily from the protocol. However, any number of parties among the n parties can *collude*.[1]

Problem Definition. The user has a secret message that she wishes to encrypt to a certain condition and n 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. Within this setting, we wish to achieve the following security and privacy goals:

- **Privacy** : No secret information can be retrieved from the system unless more than a threshold number of agents collaborate.
- **Correctness** : Any secret message can be retrieved if more than threshold number of agents collaborate (even before the user specified condition is met).
- **Revealing** : If the data of a user is decrypted, all other secret information of the protocol will be available to a certain subset of agents chosen by the user.

Notation: Secret key sk corresponds to public key pk, we denote the j^{th} bit of key sk as sk_j, a share of a secret key sk as $[sk]$ and the vector of all the such shares as $\langle [sk] \rangle$. $[sk]_i$ indicates the i^{th} share of sk. An $(n, t+1)$ threshold secret sharing requires at-least $t+2$ agents to reconstruct the secret.

2.2 Key Idea

The solution is based on a multi-agent model, where the agents offer the information escrow service for a condition (eg: time period) τ. A public key - secret key pair $(pk, sk)_\tau$ is associated with the

[1]Protocols secure against static adversary can be made secure against adaptive adversary using standard techniques [19, 43].

condition, corresponding to which the documents of the user are encrypted. Each secret key sk[2] is $(n, t+1)$ secret-shared among the agents, where at-least $t+2$ agents are needed to reconstruct the secret. The solution involves a multi-party computation among the agents and the user such that the user transfers a copy of the document to the agents, *watermarked with the secret key sk*, corresponding to pk used for encryption. None of the agents or the user knows the watermark of the transferred document and hence have no information about the secret key sk, unless at-least $t+2$ agents collaborate. If they indeed collaborate and decrypt the document, the watermark of the document would reveal the secret key. The secret key is revealed to a subset of agents who can detect the watermark, the user chooses this subset and provides them with the detection key.

Before interacting with the user, all the agents make a claim-or-refund crypto-currency deposit to an address associated with pk with a user-defined condition such the funds can be transferred before the condition is met using sk or the funds return to the respective agents after the expiry of the condition. If the agents collude to reveal the user document, any agent with access to the detection key forwarded by the user can read the watermark sk and transfer all the deposits of other agents including his own. In case the deposits are transferred (visible publicly on the blockchain) before the condition is met, *all the agents except the transferring agent are banned* from offering any future service. Since the agents do not have information on which agent has access to the correct detection key, no agent attempts to collude for the fear of losing the deposit and getting banned from the system; the best response-strategy of the agents is to not-collude (refer Section 5 for further details). Figure 1 depicts the three steps of making conditional deposit, document transfer from the user, and agents attempting collusion attack with one of the agents with watermarking key transferring the deposits of all agents to himself.

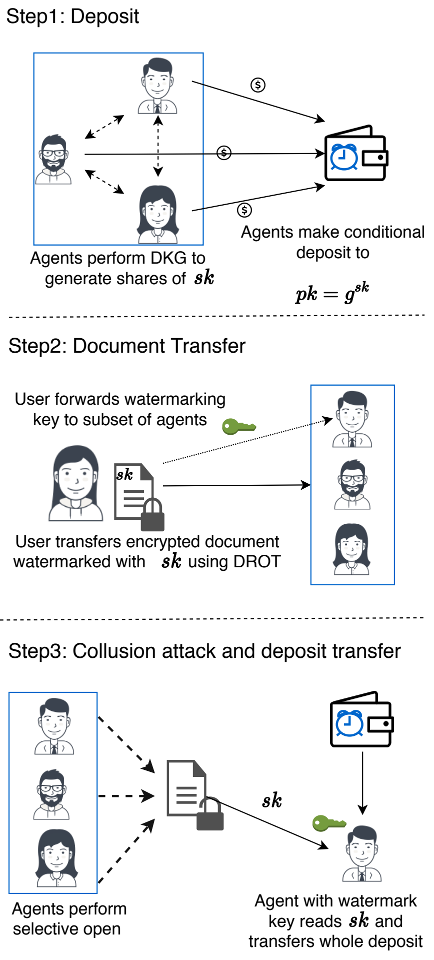
Step1: Deposit
Agents perform DKG to generate shares of sk
Agents make conditional deposit to
$pk = g^{sk}$
Step2: Document Transfer
User forwards watermarking key to subset of agents
sk
User transfers encrypted document watermarked with sk using DROT
Step3: Collusion attack and deposit transfer
sk
Agents perform selective open
Agent with watermark key reads sk and transfers whole deposit

Figure 1: Steps involved in the collusion deterrent escrow mechanism

2.3 Protocol Overview

At the beginning of the protocol, the agents run a distributed key generation (DKG) scheme [37] to generate a public key pk and the $(n, t+1)$ threshold secret shares $[sk]$ of the corresponding secret key sk, with each agent $\mathsf{A}_i, 1 \le i \le n$ receiving the share $[sk]_i$. The agents run a secure bit decomposition (MPC) protocol on the secret shares to obtain the shares of the *bits* of the secret key sk i.e., each agent A_i will have a secret share of the bit $sk_j, 0 \le j \le \kappa - 1$ (sk_j is the j^{th} bit of the κ-bit secret key sk), denoted by $[sk_j]_i$. A public server or public bulletin board is available to all the users and agents where they store non-secret information and different encryptions whenever needed. The public key pk is stored on the server, each agent $\mathsf{A}_i, 1 \le i \le n$ makes a crypto-currency claim-or-refund deposit using a smart contract to the address associated with pk with the condition τ 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 IE service for the message m, she splits the message/document into κ parts (κ is the bit length of secret key sk) and watermarks each part with robust bit watermarking to generate two versions of each part. For each part $m_k, 0 \le k \le \kappa - 1$, she obtains two watermarked parts $(m_{k,0}, m_{k,1})$ with watermarks corresponding to bit 0 and 1. She then symmetric-key encrypts all the parts $m_{k,\ell}, 0 \le k \le \kappa - 1, \ell \in \{0, 1\}$ using randomly sampled keys $\mathsf{k}_{k,\ell}$ to obtain the ciphertexts $c_{k,\ell}$. She performs 2−party computation with each of the servers such that each agent obtain shares of key $\mathsf{k}_{k,sk_k} = \mathsf{k}_0 + (\mathsf{k}_1 - \mathsf{k}_0)[sk_k]; k \in \{0, 1, \cdots \kappa - 1\}$ (Refer Section 4.2 for details). Note that the equation is a representation of oblivious transfer functionality but with a share value instead of choice bit. This functionality is realized by running a version of oblivious transfer protocol where the input of the agent is share of the secret key bit $[sk_k]$ and the input of the user is the key pair $(\mathsf{k}_0, \mathsf{k}_1)$. The user runs κ such computations with each server such that the servers obtain key shares for κ document parts. The ciphertexts $c_{k,\ell}$ are published and so are available to all the agents. When the agents collaborate, they can reconstruct the keys k_{k,sk_k} and decrypt the corresponding ciphertexts c_{k,sk_k}. However, even through collaboration they will not be able to decrypt the ciphertexts $c_{k,1-sk_k}$. The transfer of the document from the user to the agents is depicted in Figure 2.

[2] for ease of exposition, we drop the index τ further.

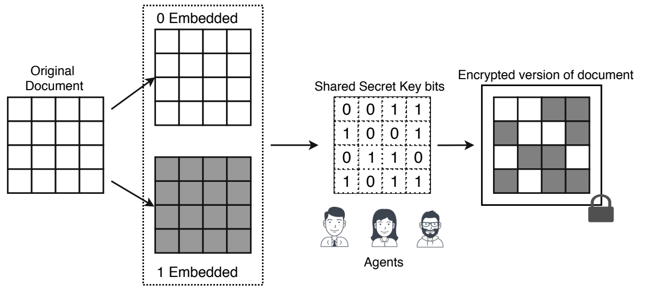

Figure 2: Watermarking the document blocks and transferring to the agents

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 bit to whoever can read the watermark. At a later point of time, when the agents decrypt the encryption c_{k,sk_k} of the watermarked message m_{k,sk_k}, 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 $\mathsf{k}_{k,sk_k}, 0 \leq k \leq \kappa - 1$ by combining the shares $[\mathsf{k}_{k,sk_k}]$. The cipher texts $c_{k,sk_k}, 0 \leq k \leq \kappa - 1$ are decrypted using the reconstructed keys to reveal the message parts m_{k,sk_k}. 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 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_k as 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.

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 which 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 will be shown in Section 5, 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 5. 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-blower) after collusion. This is further expanded in Section 5.2.

3 FUNCTIONAL BLOCKS

The basic building blocks that the proposed Collusion Deterrent Escrow protocol employs are described below.

3.1 Multi-party computation

Secure multi-party computation (MPC) [10, 26, 52, 58] 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 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 [11] are used to multiply two secret shares.

Distributed key generation (DKG). A $(n, t + 1)$ DKG [37, 59] 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 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_i of the key sk, and the reconstruction phase involving every node broadcasting their share and running the reconstruction algorithm on the collected shares. The two algorithms for share generation and reconstruction are:

- $\mathsf{dkg.share}(n, t, \kappa)$ takes in the total number of parties n, the threshold $t+1$, the security parameter κ and returns to each party i, a share of secret key $[sk]_i$ and the public key pk corresponding to sk.
- $\mathsf{dkg.recon}\langle [sk]_i \rangle$ takes in the vector of shares with at least $t + 2$ valid shares and returns the reconstructed value sk.

Bit Decomposition. A bit decomposition protocol [21, 66, 69] 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 $\langle [sk] \rangle$, the protocol outputs the shares $\langle [sk_i] \rangle$, where $sk_i, 0 \leq i \leq \kappa - 1$ are the bits of the value sk. It is defined by two algorithms:

- bit.decomp($n, \langle[sk]\rangle$) takes in the total number of users and the shares of the secret key sk and returns the vectors $\langle[sk_i]\rangle$ of shares of the bits of the secret key.
- bit.recon$\langle[sk_i]\rangle$ takes in the vector of shares of a particular bit and returns the reconstructed bit sk_i.

3.2 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, one each for key generation, embedding the watermark and detection of the watermark. $\mathcal{M}$ is the set of all possible documents, $\mathcal{WM} \in \{0,1\}$ the set of all possible watermarks, $\mathcal{K}$ is the set of all keys and κ is the security parameter. The three algorithms define the scheme:

- wm.gen (κ): Given κ, outputs keys $k_{emd}, k_{det} \in \mathcal{K}$ probabilistically.
- wm.embed (M, w, k_{emd}): Takes the document M, watermark $w \in \mathcal{WM}$ 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.* [5]. 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 $\forall w \in \mathcal{WM}$, if wm.embed$(M, w, k_{emd}) \rightarrow M'$, then $sim(M, M') = \top$.
- *Robustness:* No known algorithm [65] 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.

The CDE protocol uses a robust watermarking scheme to watermark either the bit 0 or 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 from the data, we just require that such an algorithm should not be available or known to humans; this approach was formalized as one having explicit-reduction in the work by Rogaway[65].

3.3 Claim-or-refund deposit

A claim-or-refund escrow deposit involves a deposit which 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 CDE protocol.

Before the start of the CDE 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. Cryptocurrencies like Bitcoin offer operators like OP_CHECKLOCKTIMEVERIFY and OP_CHECKMULTISIGVERIFY for checking the time lock and verify multiple-signatures. However, the operator set of Bitcoin is limited intentionally, Turing-complete languages supported by systems like Ethereum can be used when the user-specified condition is complex. The cryptocurrency script or smart contract implements the required claim-or-refund functionality with the embedded condition.

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
```

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* which interact with 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.

4 CRYPTOGRAPHIC CONSTRUCTION

The Collusion Deterrent Escrow (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 (*DROT*) to transfer the message blocks to the agents and to open the message by the agents. The CDE protocol with a user/sender U and n agents ($\mathsf{A}_1, \cdots \mathsf{A}_n$) constitutes the following algorithms:

- KeySetup(n, t, κ) generates a bit-wise shared secret key sk for the n participating agents with threshold $t+1$. It also generates the corresponding public key pk and the other public components.
- MessageSetup(m, κ, pk) outputs κ pairs of encryptions of binary-watermarked parts of the message m encrypted to the public key pk.
- *DROT* $\left((\mathsf{k}_{k,0}, \mathsf{k}_{k,1}), \langle[sk_k]\rangle\right)$ The *DROT* protocol takes the keys $\mathsf{k}_{k,0}, \mathsf{k}_{k,1}$ from the sender and the shares of the bit sk_k from the receivers and transfers k_{k,sk_k} corresponding to the bit sk_k to the receivers for $0 \le k \le \kappa - 1$.
- Open ($(\langle[s_0]\rangle, \cdots, \langle[s_{\kappa-1}]\rangle), (c_{0,0}, c_{0,1}), \cdots, (c_{\kappa-1,0}, c_{\kappa-1,1})$) 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.

4.1 Cryptographic Setup

KeySetup(n, t, κ)**.** It provides the bit-wise shared secret key sk using $(n, t+1)$-secret sharing and the corresponding public key to the n agents. The algorithm first generates the public parameters using grp.gen$(\cdot)$ which takes the security parameter κ as input. It generates the cyclic group $\mathbb{G}$ of prime order p and two generators g, h. The agents run the distributed key generation (DKG) algorithm dkg.gen() to generate the public key pk and the vector of secret key shares $\langle[sk]\rangle$, with each agent A_i, $1 \leq i \leq n$ obtaining the share $[sk]_i$. The agents run a bit decomposition algorithm bit.decomp$(\cdot)$ with the shares to obtain the bit-wise threshold-shares $[sk_j]_i$ of the bits $sk_j, 0 \leq j \leq \kappa - 1$ for each agent i. KeySetup is depicted in Algorithm 2.

Algorithm 2 KeySetup (n, t, κ)

```
1: $\mathbb{G}, g, h, p \leftarrow$ grp.gen$(\kappa)$
2: $\big(\langle[sk]\rangle, pk\big) \leftarrow$ dkg.gen$(n, t, \kappa)$
3: $\big(\langle[s_0]\rangle, \cdots, \langle[s_{\kappa-1}]\rangle\big) \leftarrow$ bit.decomp$\big(\langle[sk]\rangle\big)$
```

MessageSetup(m, κ, k_w)**.** It is run by the sender who takes the message m and use an algorithm split(m, κ) to divide it into κ parts $(m_0, \ldots, m_{\kappa-1})$ where κ is the bit-length of the secret key sk. The sender forms the robust binary watermarked versions of the message parts (using wm.embed$(\cdot)$) with the watermark embedding key k_w, watermarked with $\{0, 1\}$ to obtain $\{(m_{0,0}, m_{0,1}), \cdots, (m_{\kappa-1,0}, m_{\kappa-1,1})\}$ and encrypts them using the randomly chosen symmetric keys $\mathsf{k}_{k,\ell}$ to produce $c_{k,\ell}, 0 \leq k \leq \kappa - 1, \ell \in \{0, 1\}$. MessageSetup is described in Algorithm 3.

Algorithm 3 MessageSetup(m, κ, k_w)

```
1: $(m_0, \ldots, m_{\kappa-1}) \leftarrow$ split$(m, \kappa)$
2: for $k = 0 \ldots \kappa - 1$ do
3:     $m_{k,0} \leftarrow$ wm.embed$(m_k, 0, k_w)$
4:     $m_{k,1} \leftarrow$ wm.embed$(m_k, 1, k_w)$
5:     $\mathsf{k}_{k,0}, \mathsf{k}_{k,1} \xleftarrow{\$} \mathcal{K}$; $\mathcal{K}$ – key space
6:     $c_{k,0} \leftarrow$ enc$(\mathsf{k}_{k,0}, m_{k,0})$
7:     $c_{k,1} \leftarrow$ enc$(\mathsf{k}_{k,1}, m_{k,1})$
```

4.2 Distributed Receiver Oblivious Transfer—DROT

Oblivious Transfer is a two 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 Collusion Deterrent Escrow protocol 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 receiver receives the shares $[\mathsf{k}_s]$. The receivers can reconstruct k_s by collaboration, however, they cannot reconstruct 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.

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

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

Init Session and Sender Input:

- Upon receiving the message (INIT, $sid, \mathsf{k}_0, \mathsf{k}_1$) from sender S, record $\langle \mathsf{S}, \mathsf{k}_0, \mathsf{k}_1, sid \rangle$, forward the message (INITD, S, sid) to each receiver R_j

Receiver input:

- Receive the message (INPUT, $sid, [s]_j, j$) from the agent A_j and add $\langle j, [s]_j \rangle$ to the set $\mathcal{I}_{sid}$.
- When $|\mathcal{I}_{sid}| = n$, compute the secret bit s from the shares $[s]_j$ and send a $\top$ to S and each R_j for $1 \leq j \leq n$

Release:

- Receive the message (REL, sid, S) from the receiver R_j and store j in the set $\mathcal{R}_{sid}$ if s is computed.
- When $|\mathcal{R}_{sid}| \geq t + 2$ send k_s to the receivers R_j for $1 \leq j \leq n$.

Figure 3: Ideal Functionality Of DROT

Ideal functionality of DROT: The functionality (refer Figure 3) interacts with the sender S and the n receivers $\mathsf{R}_j, 1 \leq j \leq n$. S has the messages $\mathsf{k}_0, \mathsf{k}_1$ and each receiver R_j has the share $[s]_j$ of the bit s. The adversary $\mathcal{A}$ can corrupt the sender and upto t receivers. The sender initiates the transfer by sending the INIT message with a session id sid and forwarding the two messages $\mathsf{k}_0, \mathsf{k}_1$ to $\mathcal{F}_{DROT}$. The functionality stores them and forwards a INITD message to all the receivers informing that the sender initiated a session. The receivers forward their shares as INPUT to the functionality $\mathcal{F}_{DROT}$ which computes s by combining the shares $[s]$ after which it sends a message $\top$ to all the parties acknowledging that the shares have been received. The receivers can then send the REL message to the functionality. When it receives the REL message from at least $t+2$ receivers, it forwards (releases) the message k_s to every receiver.

Protocol. We realize *DROT* using multiple instances of two-party computation, realizing an oblivious linear function evaluation (OLE) [32] between the sender and the receivers. Before explaining the *DROT* protocol, we briefly introduce the realization of OLE in this work using two-party computation as depicted in Figure 5. The sender has two messages (a, b) and the receiver has a value x. These three values (x, a, b) are secret shared between the two parties by sending a share of each value to the other party correspondingly. We depict the sender side shares using as $([x], [a], [b])_S$ and receiver side shares as $([x], [a], [b])_R$. Using standard two-party protocols to generate shares of addition and multiplication, both the parties generate shares of the equation $y = a + (a - b) \cdot x$. The sender forwards his share y_S to the receiver who computes the value y. To transfer the keys using *DROT*, the above described two-party

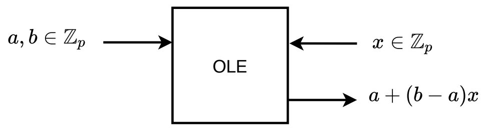

Figure 4: Oblivious linear function evaluation (OLE)

$$\begin{array}{lcr}
\textbf{Sender} & & \textbf{Receiver} \\
a, b \in \mathbb{Z}_p & & x \in \mathbb{Z}_p \\
a = a_S + a_R & & \\
b = b_S + b_R & & x = x_S + x_R \\
& \xleftarrow{[x]=x_S} & \\
& \xrightarrow{([a],[b])_R=(a_R,b_R)} & \\
([x],[a],[b])_S & & ([x],[a],[b])_R \\
= (x_S, a_S, b_S) & & = (x_R, a_R, b_R) \\
& \text{Using 2PC} & \\
y_S & \longleftrightarrow & y_R \\
= [a] + [(b-a)\cdot x] & & = [a] + [(b-a)\cdot x] \\
& \xrightarrow{y_S} & \\
& & y = a + (a-b)\cdot x \\
& & y = \mathsf{recon}(y_S, y_R)
\end{array}$$

Figure 5: OLE realization using two-party computation

computation is run by the sender with all the agents where the input of the sender is messages $(\mathsf{k}_0, \mathsf{k}_1)$ and the input of each receiver is share $[s]$ of the bit s. The receivers prove in zero-knowledge that the input shares indeed correspond to a bit s.

The Collusion Deterrent Escrow uses the *DROT* protocol to transfer the encrypted message blocks and the corresponding keys' shares to the agents. The user encrypts the message blocks $m_{k,\ell}$, using keys $\mathsf{k}_{k,\ell}$ to obtain $c_{k,\ell}$ for $k \in \{0, 1 \cdots, \kappa - 1\}$ and $\ell \in \{0, 1\}$ and broadcasts the ciphertexts to all the agents. User transfers shares of keys k_{k,sk_k} to the agents using *DROT*. As a part of *DROT* the user runs κ instances of OLE, for each instance k, the input of the user is $(\mathsf{k}_{k,0}, \mathsf{k}_{k,1})$ where as the inputs of the agents would be shares $[sk_k]$ of the bit sk_k of the signing key sk. At the end of the runs, each agent R_j has the values $[\mathsf{k}_{k,sk_k}]_j$ which are share of the values $\mathsf{k}_{k,sk_k} = \mathsf{k}_0 - (\mathsf{k}_1 - \mathsf{k}_0) \cdot sk_k$. While running *DROT* for Collusion Deterrent Escrow, the agents need to prove to the user that the input bit shares indeed correspond to bits of the secret key sk. This way of directly computing the shares of the keys k_{k,sk_k} prevents the agents from learning the other key corresponding to $\mathsf{k}_{k,1-sk_k}$. 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.

Let $[sk_k]_j$ be the share of the bit sk_k of agent R_j. As a part of *DROT*, while realizing the OLE the agent shares this value with the user by generating two shares $[sk_k]_{j,S}, [sk_k]_{j,R}$ and forwarding $[sk_k]_{j,S}$ to the user. The user shares values $\mathsf{k}_{k,0}, \mathsf{k}_{k,1}$ with agent R_j. Both participate in a two-party computation to generate shares of the value k_{k,sk_k} as discussed in OLE equation evaluation. To provide verifiability of the validity of the input shares, the agent commits to $[sk_k]_{j,R}$ by publishing $g^{[sk_k]_{j,R}} h^{v_{k,j}}; v_{k,j} \in \mathbb{Z}_p$. After interacting with the agents, the client verifies if the shares obtained indeed correspond to shares of valid bits of secret key sk by checking:

$$\prod_{k=0}^{\kappa-1} \prod_{j=0}^{\tau} \left(g^{[sk_k]_{j,S}} g^{[sk_k]_{j,R}} h^{v_{k,j}} \right)^{\lambda_j \cdot 2^k} = g^{sk} h^{v} \qquad (1)$$

where $v = \sum_{k=0}^{\kappa-1} \sum_{j=0}^{\tau} (\lambda_j \cdot v_{k,j}) 2^k$ is computed by all agents and published. λ_j are the reconstruction coefficients and $\tau > t + 2$ is the number of shares employed for reconstruction.

We postpone the security analysis for *DROT* to the Appendix ??.

Algorithm 4
Open $(\langle [s_0] \rangle, \ldots, \langle [s_{\kappa-1}] \rangle, (c_{0,0}, c_{0,1}) \cdots (c_{\kappa-1,0}, c_{\kappa-1,1}))$

```
1: $sk = (sk_0, \ldots, sk_{\kappa-1})$
2: **for** $k = 0 \ldots \kappa - 1$ **do**
3:     $\mathsf{k}_{k,sk_k} \leftarrow \mathsf{recon}([\mathsf{k}_{k,sk_k}]_1, \ldots, [\mathsf{k}_{k,sk_k}]_{t+1})$
4:     $c_k \leftarrow \mathsf{dec}(c_{k,sk_k}, \mathsf{k}_{k,sk_k})$
5:     $m_k \leftarrow \mathsf{dec}(c_k, sk)$
6: $m' \leftarrow \mathsf{combine}(m_0, \ldots, m_{\kappa-1})$
7: **return** $m'$
```

4.3 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 $[\mathsf{k}_{k,sk_k}], 0 \leq k \leq \kappa - 1$ for each k and the corresponding ciphertext c_{k,sk_k} is decrypted. This ciphertext decrypts to the watermarked message part m_k. After all parts of the messages $m_k, k \in \{0, \ldots, \kappa - 1\}$ are recovered, they are combined to form the final message m' which is a watermarked version of m. m' contains the whole of the secret key sk as watermark embedded in it. The Open algorithm is presented in Algorithm 4.

Detection key distribution. In the CDE 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.

4.4 Escrow deposits

Before the start of the protocol, the user and the agents agree on the deposit value D. The agents proceed to deploy the scripts/smart-contracts embedding the condition τ specified by the user to the address corresponding to the public key pk. Each agent i creates a deposit transaction TX_i for the value D and the user verifies that the agreed on value of funds have been held in the deposit. When the condition τ is met, the deposits are automatically transferred back to the agents. Any agent with sk can transfer all the deposits to the address of his choice at any point in time. In the non-colluding scenario, after the condition τ is met, the agents take a non-zero time to compute the secret key sk. The deposits are transferred back instantaneously to all the agents by the smart contracts before the sk is computed by the agents once τ is met. If the deposit is ever transferred before τ, all the agents except the agent that transferred the deposit is banned from offering any future service there by penalizing them. This ban can be either permanent or for a specified period of time. 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. The agents provide strong identities linked to their physical identities, hence the agents can not launch any sybil attack to target and ban other agents.

5 GAME-THEORETIC MODELLING AND ANALYSIS

We model and analyze the collusion deterrent escrow (CDE) protocol as a mechanism through which the user induces a game [57] 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 modelled as collusion strategies played by the agents. For simplicity, we initially analyze the scenario with one regulator (user) and two agents $A1, A2$.

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 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 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.

Table 1: (Trivial) Pay-off matrix for the two agent threshold escrow *without* collusion-deterrence.

A2 \ A1	*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)	$(v + \frac{d}{2}, v + \frac{d}{2})$

Here, v is the fee offered by the user and d is the value of the escrowed document to the agents.

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 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 [57] 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. Following [48], we model the collusion among agents with a side contract which simplifies all the bargaining and communication that may occur among the two agents. 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.

The grand contract induces a bayesian game among the agents which is defined by the following:

- The set of agents $N = \{1, 2\}$
- The typeset of the agents $\Theta = \{e, ie\}$
- The strategy space of each agent i is $S_i = \{$*reject*, *accept*, *wait1*, *transfer1*, *wait2*, *transfer2*$\}$
- The utilities corresponding to the strategy and the type of the agent $u_i(\theta_i, s_i, s_{-i}), \theta_i \in \Theta, s_i \in S_i$. s_{-i} indicates the strategy(ies) of player(s) other than player i.

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 $\Theta = \{e, ie\}$ 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. The utilities of the players are defined as $u_i(\theta_i, s_i, s_{-i}) = v$ if both s_i, s_{-i} are *accept*; $u_i(\theta_i, s_i, s_{-i}) = 0$ if either s_i or s_{-i} is *reject* $\forall \theta_i \in \Theta$. Utilities of other combinations of (θ_i, s_i, s_{-i}) are as defined in the Tables 2, 3. The utilities of players playing *wait2* and *transfer2* listed in Table 3 are for both the types.

5.1 Collusion

Once the regulator offers the grand contract, the agents are free to collude among themselves which is captured by the side contract. When the agents sign the side contract it entails them to choose one of two collusion scenarios, Collusion-1 or Collusion-2. In Collusion-1, the agents can only play the strategies $\{$*transfer1*, *wait1*$\}$. Similarly in Collusion-2, the agents can only play the strategies $\{$*transfer2*,*wait2*$\}$.

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

Table 2: Pay-offs of rational agents under different strategies in Collusion-1- document decryption (strictly dominant strategies indicated in grey)

A2 \ A1		Efficient(e)		Inefficient(ie)
		wait1	*transfer1*	*wait1*
Efficient (e)	*wait1*	(α, α)	$(\alpha + D, \alpha - F - D)$	(α, α)
	transfer1	$(\alpha - F - D, \alpha + D)$	$(\alpha - \frac{F}{2}, \alpha - \frac{F}{2})$	$(\alpha - F - D, \alpha + D)$
Inefficient (ie)	*wait1*	(α, α)	$(\alpha + D, \alpha - F - D)$	(α, α)

Table 3: Pay-offs of rational agents under different strategies in Collusion-2- secret key reconstruction

A2 \ A1	*wait2*	*transfer2*
wait2	(α, α)	$(\alpha + D, \alpha - F - D)$
transfer2	$(\alpha - F - D, \alpha + D)$	$(\alpha - \frac{F}{2}, \alpha - \frac{F}{2})$

be collusion scenarios using document decryption and secret key (sk) reconstruction respectively. In the CDE 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 $F \gg 0$.

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'. The agent that does not have the detection key can not transfer the deposit and hence can only wait after collusion. In the second collusion attack where the agents collude and reconstruct the secret key of the protocol (Collusion-2), each agent can choose to transfer the deposit or wait without transferring the deposit. These two scenarios are captured by the following: in Collusion-1, any efficient agent (type-e) can play the strategies {*transfer1*, *wait1*} whereas the inefficient agent (type-ie) can only play the strategy {*wait1*}. Collusion-2 is independent of the information $\mathcal{I}$ and hence agents of either type can play both the strategies {*transfer2*,*wait2*}. The agents can both either play the strategies {*transfer1*,*wait1*} or {*transfer2*, *wait2*} indicating two collusion scenarios. For the ease of exposition we depict them in separate matrices in Tables 2, 3.

Collusion game - Payoffs. The expected pay-offs of the agents in the two collusion scenarios, Collusion-1, Collusion-2 are captured by the pay-off matrices in Tables 2, 3. With document decryption - Collusion-1, when both the efficient agents decrypt the user document and not transfer the deposit (play *wait1*), they both can accrue a pay-off of $\alpha = v + \frac{d}{2}$. If one of the agents transfers (plays *transfer1*) 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, hence in expectation they accrue a pay-off $0.5(\alpha + D) + 0.5(\alpha - F - D) = \alpha - \frac{F}{2}$. $u_i(e, transfer1, s_{-i}) > u_i(e, wait1, s_{-i}) \forall s_{-i} \in \{transfer1, wait1\}$ (from Table 2), hence for an efficient agent *transfer1* is a strictly dominant strategy, he always plays *transfer1*. For the inefficient agent, *wait1* 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 Tables 2 and 3. They pay-offs of Table 3 and the pay-off submatrix of Table 2 when both the agents are efficient are similar to the pay-offs in well known prisoners' dilemma [46].

In Collusion-2 when the agents reconstruct the secret key, the obtained pay-offs are presented in Table 3. After the reconstruction of the secret key, if both the agents wait and do not attempt to transfer the deposit (play *wait2*), both the agents obtain a pay-off of α, however if one of the agents transfers the deposit (plays *transfer2*), 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 *transfer2* is a strictly dominant strategy of every agent in this collusion scenario and hence the agents always play *transfer2*.

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 *transfer2*. However, the agent will *always* play his dominant strategy of transfer to improve his pay-off even after accepting a payment from the other agent. Hence at equilibrium no transfers are made through the side contract between the agents making the side contract a null contract.

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.

Expected payoffs. Collusion-1: 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 2) is:

$$a = p^2(\alpha - \frac{F}{2}) + p(1-p)(\alpha + D) + (1-p)p(\alpha - F - D) + (1-p)^2\alpha$$

$$= \alpha(p^2 + 2p(1-p) + (1-p)^2) - \frac{p^2 F}{2} - (p - p^2)F$$

$$= \alpha - F(p - \frac{p^2}{2})$$

Collusion-2: Both the agents play *transfer2* as their dominant strategy, accruing each a pay-off of $b = \alpha - \frac{F}{2}$ as shown in Table 3. Thus, the maximum expected pay-off that can be obtained by each agent through collusion is

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

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 β is strictly less than v making collusion unviable.

5.2 Extending to the n agents scenario

As in the two agent scenario, in the n agent scenario, the regulator transfers the information $\mathcal{I}$ to each agent independently with probability p.

Collusion-1: Similar to the two agent scenario in Collusion-1 it is a dominant strategy for the efficient agent to play *transfer1*. An inefficient agent's only strategy is to play *wait1*, so he always plays *wait1*. If one agent plays *transfer1*, he obtains a pay-off of $\alpha + (n-1)D$ [3] and every other agent obtains $\alpha - (F + D)$. When more than one agent plays *transfer1*, each agent obtains $\alpha + (n-1)D$ with equal probability. With t other efficient agents in the system each playing their dominant strategy *transfer1*, each agent obtains a positive pay-off of $\alpha + (n-1)D$ with a probability of $\frac{1}{t}$ and obtains a pay-off of $\alpha - F - D$ with probability $\frac{t-1}{t}$. Hence the expected pay-off of an efficient agent playing his dominant strategy with t *other* efficient agents is:

$$\frac{1}{t+1}(\alpha + (n-1)D) + \frac{t}{t+1}(\alpha - F - D)$$

With at-least one efficient agent present in the system (playing his dominant strategy), the pay-off of an inefficient agent is $(\alpha - F - D)$. When all the agents are inefficient, the pay-off obtained by each agent in the system is α.

So the expected pay-off of agent under collusion-1 is:

$$\begin{aligned} a = p &\sum_{t=0}^{n-1} \binom{n}{t} p^t (1-p)^{n-1-t} \\ &\cdot \Big(\frac{1}{t+1}(\alpha + (n-1)D) + \frac{t}{t+1}(\alpha - F - D)\Big) \\ + (1-p) &\sum_{t=1}^{n-1} \binom{n}{t} p^t (1-p)^{n-1-t} (\alpha - F - D) + (1-p)^n \alpha \end{aligned} \quad (2)$$

Collusion-2: In this scenario which is equivalent to the agents reconstructing the secret key, the information $\mathcal{I}$ is irrelevant to the agents and all the agents play their dominant strategy *transfer2*. The expected pay-off of each agent is

$$b = \frac{1}{n}(\alpha + (n-1)D) + \frac{n-1}{n}(\alpha - F - D)$$

The total expected pay-off of each agent through the collusion game is $\beta = max(a, b)$. The regulator chooses a value p such that $\beta < v$ (where $\alpha = v + \frac{d}{2}$), for instance for $p = 1, m = \alpha - \frac{(n-1)F}{n} \ll v$ as $F \gg d, v$. Thus, the agents do not play the *accept and collude* strategy at equilibrium in the CDE protocol.

[3] Pay-off of $(n-1)D$ is equivalent to the agent transferring deposit D of all other agents to self.

Table 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.

A1 / A2	*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)	(β, β)

Theorem 5.1. *In CDE protocol, under equilibrium 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).*

Proof. For $n = 2$ when the agents collude, the expected pay-off as computed is 5.1 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 (β, β). Thus the trivial pay-off matrix of Table 1, under the CDE protocol mechanism changes to Table 4. Since $\beta < v$, it is evident from Table 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 β. 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 6. '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 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 2, 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

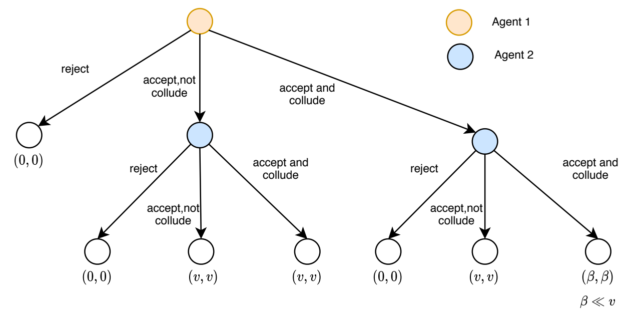

Figure 6: Extensive form game induced by the user in CDE between two agents. v is the pay-off when agents do not collude and $\beta \ll v$ is the pay-off when the agents collude

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[39, 41], 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 [39, 41, 44] 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.
Remarks:

- 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, β, 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, β 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 κ instead of just κ) 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.

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

The functionality $\mathcal{F}_{CDE}$ interacts with a user U and n agents A_j, $1 \leq j \leq n$. U has a message m_{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 (INIT, $sid, \mathcal{S}_{\mathcal{U}}, m_{\mathsf{U}}$) from user U, record and store $\langle \mathsf{U}, \mathcal{S}_{\mathcal{U}}, m_{\mathsf{U}}, sid \rangle$, forward the message (INITD, U, sid) to each agent A_j

Open message:

- Receive the message (OPEN, sid, U) from the agent A_j and store j in the set Q_{sid}.
- When $|Q_{sid}| \geq t + 2$, send m_{U} to all the agents A_k for $k \in Q_{sid}$ and forward the message "KEY" to agents A_d for $d \in Q_{sid} \cap \mathcal{U}$

Release message:

- Receive the message (REL, sid, U) from the agent A_j and store j in the set $\mathcal{R}_{sid}$.
- When $|\mathcal{R}_{sid}| \geq t + 2$, send m_{U} to all the agents A_k for $1 \leq k \leq n$.

Figure 7: Collusion Deterrent Escrow Ideal Functionality: $\mathcal{F}_{CDE}$

6 SECURITY ANALYSIS

6.1 Security Definition

The system consists of n agents $\mathsf{A}_j, 1 \leq j \leq n$ and a user U with an input m_{U}. The agents and the user are interactive Turing machines that communicate with an ideal functionality $\mathcal{F}_{CDE}$.

The adversary is a PPT machine with access to an 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 $\mathcal{F}_{CDE}$ (See Figure 7), the user chooses a set 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. The user initiates the document transfer using the INIT message and forwards the data m_{U} and the set $\mathcal{S}_{\mathcal{U}}$ to $\mathcal{F}_{CDE}$ with the session id sid. $\mathcal{F}_{CDE}$ receives and stores $\langle \mathsf{U}, \mathcal{S}_{\mathcal{U}}, m_{\mathsf{U}} \rangle$. An INITD message is sent to all the agents indicating that the session has been initiated by the user. The t-bounded adversary $\mathcal{A}$ corrupts the user, he chooses a set of agents $\mathcal{S}_{\mathcal{U}'}$ where $\mathcal{U}'$ is a set of agent indices. If $t+2$ or more number of agents decide to selectively open the user message by sending a OPEN request, the functionality stores the indices of subset of agents sending the request in the index set Q and opens the user message to the collaborating agents. The functionality forwards a message "KEY" informing all the agents from the index set $\mathcal{U} \cap Q$ - agents selected by user who are among the collaborating agents. The KEY message models the release of secret key sk to such agents in the real world protocol. If at-least $t+2$ agents decide to release the user message by forwarding REL, the functionality forwards the user message to every agent.

From the different steps of the CDE protocol described in Section 4 it can be seen that, when the agents decide to decrypt the secret message, they reconstruct the keys k_{k,sk_k} and the decrypted message consists the secret key embedded as 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 [53], 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 the *DROT* protocol from Section 4.2, 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 $\mathcal{S}_{OLE}$ for OLE functionality of Figure 8 before we proceed to provide a proof-sketch for the *DROT* protocol.

Definition 6.1. Universal Composability: Let the ensemble of the outputs of the environment $\mathcal{E}$, with attacker $\mathcal{A}$ and users running the protocol π be $\mathsf{EXE}_{\pi,\mathcal{A},\mathcal{E}}$. A Protocol π UC-realizes the ideal functionality $\mathcal{F}$, if there exists a simulator S for any PPT adversary $\mathcal{A}$ such that the ensembles $\mathsf{EXE}_{\pi,\mathcal{A},\mathcal{E}}$ and $\mathsf{EXE}_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are computationally indistinguishable for any environment $\mathcal{E}$.

Simulator $\mathcal{S}_{OLE}$: The sender and receiver running the protocol for Oblivious Linear Evaluation (OLE) presented in Figure 5 publish Pedersen commitments for each of the values in the protocol. For the corrupted sender, the simulator chooses two random values k_0', k_1' are forwards the message $(\mathsf{INIT}, sid, k_0', k_1')$ to the functionality. The receiver forwards the value x to the functionality using the message (INPUT, sid, x) and obtains the value $k_0' + (k_1' - k_0') \cdot x$. The sender and receiver do not interact beyond the evaluation and hence any output is equally likely for the receiver. The simulator can open the commitments to values of its choice at any later point of time. For the corrupted receiver case, when the receiver forwards the value x_S, the simulator forwards two random values a_1, b_1 to the receiver and forwards the message $(\mathsf{INPUT}, sid, x_S)$ to the functionality $\mathcal{F}_{OLE}$. It receives the value $y_S = a + (a-b)x_S$ from the functionality and forwards to the receiver the value y_S' such that $y_S = recon(y_S', y_R)$.

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

The functionality $\mathcal{F}_{OLE}$ interacts with the sender S and receiver R. The sender S has two messages $\mathsf{k}_0, \mathsf{k}_1 \in \mathbb{Z}_p$ and the receiver has a random value $x \in \mathbb{Z}_p$.

Init Session and Sender Input:

- Upon receiving the message $(\mathsf{INIT}, sid, \mathsf{k}_0, \mathsf{k}_1)$ from sender S, record $\langle \mathsf{S}, \mathsf{k}_0, \mathsf{k}_1, sid \rangle$, forward the message $(\mathsf{INITD}, \mathsf{S}, sid)$ to the receiver R.

Receiver input:

- Receive the message (INPUT, sid, x) from the receiver R. Compute and forward the value $k_0 + (k_1 - k_0) \cdot x \in \mathbb{Z}_p$ to R. Send $\top$ to sender S.

Figure 8: Ideal Functionality Of OLE protocol

Theorem 6.2. *Assuming a secure two-party computation of the OLE protocol, the DROT protocol (Section 4.2) securely implements the ideal functionality $\mathcal{F}_{DROT}$ (in Fig. 3) under the mixed-behaviour model [53].*

Proof sketch. The simulator $\mathcal{S}_{DROT}$ interacts with the user U and n agents $P_1, P_2, \cdots P_n$, 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 k_0, 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 $\mathcal{S}_{DROT}$ invokes $\mathcal{S}_{OLE}$ with corresponding inputs while generating the view for the adversary.

For the corrupted sender case, $\mathcal{S}_{DROT}$ simulates n agents to the sender with inputs k_0, k_1 and invokes $\mathcal{S}_{OLE}$ for each of the instance of OLE between the sender and the agents. It forwards two random input values k_0', k_1' as the message $(\mathsf{INIT}, sid, k_0', k_1')$ to the functionality $\mathcal{F}_{DROT}$. The receiver on input of value x to the functionality obtains the value $k_0' + (k_1' - k_0') \cdot x$. For the corrupted receiver case, where a maximum of t receivers are corrupted, the sender forwards $(\mathsf{INIT}, sid, k_0, k_1)$ and the honest parties P_j forward the message $(\mathsf{INPUT}, sid, [s]_j, j)$ to the functionality. The simulator interacts with the t corrupted agents and forwards initial randomized information such that the de-randomization values can be used to set any input information. It forwards random valid share inputs as the t agents to $\mathcal{F}_{DROT}$ using the message $(\mathsf{INPUT}, sid, [s]_j, j)$. When the functionality forwards the k_s value, the simulator sets the de-randomization values such that the output shares computed by the agents correspond to shares of the value k_s. For the case of corrupted sender and $t-1$ corrupted receivers, the simulator forwards the message $(\mathsf{INIT}, sid, k_0', k_1')$ to the functionality like in the corrupted sender case. Apart from that it relays, all the protocol messages as is between the sender and the $t-1$ agents. The adversary can not distinguish the views owing to the fact that the randomization information in the OLE scheme is uniform over the whole group chosen and the fact that the commitments can be opened to any value by the simulator.

From Theorem 5.1, 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 5.1 and Table 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 6.3. *Let dkg, bit-decomp be secure MPC protocols, $wm(\cdot)$ is a robust bit watermarking algorithm, and DROT is secure (as in Theorem 6.2). The CDE protocol π_{CDE} securely realizes the ideal functionality $\mathcal{F}_{CDE}$ under the mixed-behaviour model[53].*

Proof Sketch. The system consists of user U and n agents $P_1, \cdots, P_n$; 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 [37] and [62, 67] for the simulators for DKG and bit-decomposition algorithms. We refer to them as $\mathcal{S}_{DKG}$ and $\mathcal{S}_{\text{BitDec}}$.

The simulator $\mathcal{S}_{CDE}$ generates an indistinguishable view for the adversary in the real-world ideal-world paradigm. It invokes the three simulators $\mathcal{S}_{DKG}, \mathcal{S}_{\text{BitDec}}$ and $\mathcal{S}_{DROT}$ for each of the phases of the protocol run. The DKG protocol [37] 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 $\mathcal{S}_{CDE}$ invokes $\mathcal{S}_{DKG}, \mathcal{S}_{\text{BitDec}}$ during these phases of the 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 values q_i. These queries are used by the user to sample keys to encrypt the different message blocks. The simulator $\mathcal{S}_{CDE}$ invokes κ instances of $\mathcal{S}_{DROT}$ once for each of the bits of the secret key. With the only difference being, 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 instead of random values.

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 5.1. 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. □

7 IMPLEMENTATION

We implement the *Collusion Deterrent Escrow* protocol using HoneybadgerMPC [52], SCALE-MAMBA [26], and Charm cryptographic library [6]. 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. The code for running the protocols can be found at the anonymous link https://anonymous.4open.science/r/1aa37eb3-1f77-4c35-991a-0c7643faf060/.

Distributed Key Generation—DKG. The DKG protocol is realized using HoneybadgerMPC [52], a secret sharing based python MPC framework supporting malicious security. [4] We implement the DKG protocol proposed by Gennaro *et al.* [37], 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. From section 5.2, when adversary can 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.* [21]. The round complexity of protocol is $O(log(\kappa))$ where κ 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 [67]. Moreover, we use the prefix multiplication protocol introduced in [20] to replace the prefix AND sub-protocol in [67]. 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 4.2) and we implement it using SCALE-MAMBA [26] 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 [26] is leveraged in the online phase. The whole two-party

[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.

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 [28] and the implementation based on [27]. 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.

7.1 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 $n = 5, 8$. The times taken and the data transferred by each node for the DKG and the bit-decomposition (BitDec) phases are provided in Table 6. 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 time 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 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 5 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.

8 RELATED WORK

Timed-release encryption (TRE) was first introduced by May [55]. Several applications and approaches using TRE have been proposed including sealed bids [22] electronic voting systems [23], spam and denial of service preventions [33] and proof-of-work systems [42]. For time-lock puzzles, a well known puzzle was created by Rivest *et al.* in 1996 [63] 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 [18], signatures [35, 36] and key escrow [12, 13]. Besides the RSA-based construction of a time-lock puzzles recent results from Bitansky et al. show constructions based on random encodings [7, 14]. 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. [63] 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 [60] 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.* [30] 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 chose a time. The shareholders can not reconstruct the secret shared before the time specified is over [70]. Many schemes based on identity based encryption were proposed, where the trusted key distribution center is also used for ensuring the correct time [17, 24, 25, 56]. 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 [50], [51], [34] present different audio watermarking schemes while works like [49], [72] deal with robust video watermarking. For software watermarking, schemes suggested in [68], [45] can be considered. The proposed IE protocol admits any robust watermarking scheme with no known attacks [65]. Lysyanskaya *et al.*

Halpern and Teague [41] 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 perform the reconstruction. Gordon *et al.* [39] improve on the randomized protocol of [41] by overcoming the impossibility result. Lysyanskaya *et al.* [53] introduce the mixed behaviour model where the parties are either rational or adversarial, the authors provide a frame work for multi-party computation in the proposed model. We adopt the mixed-behaviour model in this work.

9 CONCLUSION AND FUTURE WORK

We propose a novel Collusion Deterrent Escrow (CDE) mechanism to realize a distributed approach for information escrows that dis-incentivizes collusion. In the CDE mechanism the escrow is offered as a service and the user availing it transfers an encrypted version of her data to be decrypted only when the user-defined condition is met. The proposed mechanism dis-incentivizes any collusion among the agents offering the service to selectively decrypt the user’s data. The agents make a conditional deposit on a cryptocurrency system before the start of the data transfer. If they collude to decrypt the data, the mechanism ensures that at-least one agent will be able to transfer all the deposits while the rest of the agents are banned from the system. This penalty mechanism eliminates the risk of selective opening encountered in such protocols proposed till now. We analyse the protocol as a game-theoretic mechanism inducing a bayesian game among the agents and show that it is the best response strategy of the agents to not collude. The cryptographic construction of CDE protocol employs robust watermarking, a

Table 5: 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

Table 6: 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	34.2sec	92MB
8	DKG	2.165sec	6.3 KB
	BitDec	34.1sec	108MB
2	DROT	124.6msec	-

claim-or-refund smart contract and a proposed multi-party extension of oblivious transfer primitive called the distributed-receiver oblivious transfer. The prototype implementation shows the ease of setup and the feasibility of the protocol.

While this work focuses on realizing secure information escrow using MPC, we find that our work can be extended to other MPC applications specially those using MPC-as-a-service for several users such as allegation escrow [8, 47], private information retrieval, and distributed setups for identity and attribute-based cryptography. In general, this work can offer a key-step towards developing a comprehensive strategy to deal with passive collusions in the MPC applications in the near future.

ACKNOWLEDGEMENTS

We thank Alexandros Psomas for helpful comments and feedback on the game theoretic analysis. We also thank Simon Heinzel for his efforts with a preliminary manuscript associated with the work. This work is partially supported by the National Science Foundation under grant CNS-1846316, the MITRE innovation program for academic cybersecurity research, and the IARPA HECTOR program.

REFERENCES

[1] drand: Distributed randomness beacon. https://drand.love/.
[2] Escrow tech. https://www.escrowtech.com/.
[3] Iron mountain. https://www.ironmountain.com/information-management/software-escrow.
[4] A programmer solved a 20 year old, forgotten crypto puzzle. https://www.wired.com/story/a-programmer-solved-a-20-year-old-forgotten-crypto-puzzle/, 2019.
[5] André Adelsbach and Ahmad-Reza Sadeghi. Zero-knowledge watermark detection and proof of ownership. In *Information Hiding*, 2001.
[6] Joseph A. Akinyele, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, and Aviel D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, pages 111–128, 2013.
[7] Benny Applebaum. Randomized encoding of functions. In *Cryptography in Constant Parallel Time*, Information Security and Cryptography, pages 19–31. Springer Berlin Heidelberg, 2014.
[8] Venkat Arun, Aniket Kate, Deepak Garg, Peter Druschel, and Bobby Bhattacharjee. Finding safety in numbers with secure allegation escrows. In *NDSS 2020*, 2020.
[9] Ian Ayres and Cait Unkovic. Information escrows. *Mich. L. Rev.*, 111:145, 2012.
[10] Assi Barak, Martin Hirt, Lior Koskas, and Yehuda 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*, pages 695–712, 2018.
[11] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology – Crypto*, pages 420–432, 1991.
[12] Mihir Bellare and Shafi Goldwasser. Encapsulated key escrow. Technical report, University of California, San Diego, 1996.
[13] Mihir Bellare and Shafi Goldwasser. Verifiable partial key escrow. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, CCS '97, pages 78–91, New York, NY, USA, 1997. ACM.
[14] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. Cryptology ePrint Archive, Report 2015/514, 2015.
[15] Ian F. Blake and Aldar C-F. Chan. Scalable, server-passive, user-anonymous timed release public key encryption from bilinear pairing. In *In Proc. IJCAI '01*, pages 504–513, 2004.
[16] Dan Boneh, Xavier Boyen, and Shai Halevi. Chosen ciphertext secure public key threshold encryption without random oracles. In David Pointcheval, editor, *Topics in Cryptology âĂŞ CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 226–243. Springer Berlin Heidelberg, 2006.
[17] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 213–229, London, UK, UK, 2001. Springer-Verlag.
[18] Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *Advances in Cryptology âĂŤ CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254. Springer Berlin Heidelberg, 2000.
[19] Ran Canetti, Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In *Annual International Cryptology Conference*, pages 98–116. Springer, 1999.
[20] Octavian Catrina and Sebastiaan De Hoogh. Improved primitives for secure multiparty integer computation. In *International Conference on Security and Cryptography for Networks*, pages 182–199. Springer, 2010.
[21] Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In Radu Sion, editor, *Financial Cryptography*, volume 6052 of *Lecture Notes in Computer Science*, pages 35–50. Springer, 2010.
[22] Konstantinos Chalkias and George Stephanides. Timed release cryptography from bilinear pairings using hash chains. In Herbert Leitold and EvangelosP. Markatos, editors, *Communications and Multimedia Security*, volume 4237 of *Lecture Notes in Computer Science*, pages 130–140. Springer Berlin Heidelberg, 2006.
[23] Hsing-Chung Chen and Rini Deviani. A secure e-voting system based on rsa time-lock puzzle mechanism. In *BWCCA*, pages 596–601. IEEE, 2012.
[24] Jung Hee Cheon, Nicholas Hopper, Yongdae Kim, and Ivan Osipkov. Timed-release and key-insulated public key encryption. *IACR Cryptology ePrint Archive*, 2004:15, 2004.
[25] Jung Hee Cheon, Nicholas Hopper, Yongdae Kim, and Ivan Osipkov. Provably secure timed-release public key encryption. *ACM Trans. Inf. Syst. Secur.*, 11(2), 2008.
[26] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - CRYPTO 2012*, pages 643–662, 2012.
[27] Sourya Das. DWT-DCT-Digital-Image-Watermarking. https://github.com/diptamath/DWT-DCT-Digital-Image-Watermarking.
[28] 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*, pages 458–461, 2012.
[29] Yvo G. Desmedt and Yair Frankel. Threshold cryptosystems. In *Proceedings on Advances in Cryptology*, CRYPTO '89, pages 307–315, New York, NY, USA, 1989. Springer-Verlag New York, Inc.
[30] Giovanni Di Crescenzo, Rafail Ostrovsky, and Sivaramakrishnan Rajagopalan. Conditional oblivious transfer and timed-release encryption. In Jacques Stern, editor, *Advances in Cryptology âĂŤ EUROCRYPT âĂŹ99*, volume 1592 of *Lecture Notes in Computer Science*, pages 74–89. Springer Berlin Heidelberg, 1999.

[31] 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)*, pages 595–612, 2018.
[32] Nico Dottling, Satrajit Ghosh, Jesper Buus Nielsen, Tobias Nilges, and Roberto 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, 2017. Association for Computing Machinery.
[33] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In ErnestF. Brickell, editor, *Advances in Cryptology âĂŤ CRYPTOâĂŹ 92*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147. Springer Berlin Heidelberg, 1993.
[34] Yousof Erfani and Shadi Siahpoush. Robust audio watermarking using improved ts echo hiding. *Digital Signal Processing*, 19(5):809 – 814, 2009.
[35] Juan A. Garay and Markus Jakobsson. Timed release of standard digital signatures. In *FC'02: Proceedings of the 6th international conference on Financial cryptography*, pages 168–182, Berlin, Heidelberg, 2003. Springer-Verlag.
[36] Juan A. Garay and Carl Pomerance. Timed fair exchange of standard signatures. In RebeccaN. Wright, editor, *Financial Cryptography*, volume 2742 of *Lecture Notes in Computer Science*, pages 190–207. Springer Berlin Heidelberg, 2003.
[37] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptol.*, 20(1):51–83, January 2007.
[38] Rosario Gennaro, Michael O. Rabin, and Tal 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*, PODC '98, pages 101–111, New York, NY, USA, 1998. ACM.
[39] S. Dov Gordon and Jonathan Katz. Rational secret sharing, revisited. Cryptology ePrint Archive, Report 2006/142, 2006.
[40] gwern.net. Time-lock encryption. https://www.gwern.net/Self-decrypting-files, 2018.
[41] Joseph Y. Halpern and Vanessa Teague. Rational secret sharing and multiparty computation: Extended abstract. *CoRR*, abs/cs/0609035, 2006.
[42] Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols(extended abstract). In Bart Preneel, editor, *Secure Information Networks*, volume 23 of *IFIP âĂŤ The International Federation for Information Processing*, pages 258–272. Springer US, 1999.
[43] Stanisław Jarecki and Anna Lysyanskaya. Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 221–242. Springer, 2000.
[44] Akinori Kawachi, Yoshio Okamoto, Keisuke Tanaka, and Kenji Yasunaga. General constructions of rational secret sharing with expected constant-round reconstruction. Cryptology ePrint Archive, Report 2013/874, 2013.
[45] Sam Kim and David J. Wu. Watermarking cryptographic functionalities from standard lattice assumptions. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 503–536, Cham, 2017. Springer International Publishing.
[46] Steven Kuhn. PrisonerâĂŹs dilemma. 1997.
[47] Benjamin Kuykendall, Hugo Krawczyk, and Tal Rabin. Cryptography for #metoo. *Proc. Priv. Enhancing Technol.*, 2019(3):409–429, 2019.
[48] Jean-Jacques Laffont and David Martimort. Collusion under asymmetric information. *Econometrica: Journal of the Econometric Society*, pages 875–911, 1997.
[49] 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*, pages 251–256, 2002.
[50] Bai Ying Lei, Ing Yann Soon, and Zhen Li. Blind and robust audio watermarking scheme based on svd?dct. *Signal Processing*, 91(8):1973 – 1984, 2011.
[51] Wen-Nung Lie and Li-Chun Chang. Robust and high-quality time-domain audio watermarking based on low-frequency amplitude modification. *IEEE Transactions on Multimedia*, 8(1):46–59, Feb 2006.
[52] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew 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*, pages 887–903, 2019.
[53] Anna Lysyanskaya and Nikos Triandopoulos. Rationality and adversarial behavior in multi-party computation. In *Annual International Cryptology Conference*, pages 180–197. Springer, 2006.
[54] Mohammad Mahmoody, Tal Moran, and Salil Vadhan. Time-lock puzzles in the random oracle model. In Phillip Rogaway, editor, *Advances in Cryptology âĂŞ CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 39–50. Springer Berlin Heidelberg, 2011.
[55] T C May. Timed-release crypto. http://cypherpunks.venona.com/archive/1993/02/msg00129.html, 1993.
[56] Marco Casassa Mont, Keith Harrison, and Martin Sadler. The hp time vault service: Innovating the way confidential information is disclosed, at the right time, 2002.
[57] Roger B Myerson. *Game theory*. Harvard university press, 2013.
[58] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. Cryptology ePrint Archive, Report 2011/091, 2011. https://eprint.iacr.org/2011/091.
[59] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, pages 129–140, London, UK, UK, 1992. Springer-Verlag.
[60] Michael O. Rabin and Christopher Thorpe. Time-lapse cryptography. Technical report, Harvard University School of Engineering and Applied Sciences, 2006.
[61] Anjana Rajan, Lucy Qin, David W. Archer, Dan Boneh, Tancrède Lepoint, and Mayank Varia. Callisto: A cryptographic approach to detecting serial perpetrators of sexual misconduct. In Ellen W. Zegura, editor, *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies, COMPASS 2018*, pages 49:1–49:4, 2018.
[62] Tord Reistad and Tomas Toft. Linear, constant-rounds bit-decomposition. In Donghoon Lee and Seokhie Hong, editors, *Information, Security and Cryptology – ICISC 2009*, pages 245–257, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
[63] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1996.
[64] Ronald L. Rivest. Description of the LCS35 Time Capsule Crypto-Puzzle. https://people.csail.mit.edu/rivest/lcs35-puzzle-description.txt, 1999.
[65] Phillip Rogaway. Formalizing human ignorance. In Phong Q. Nguyen, editor, *Progress in Cryptology - VIETCRYPT 2006*, pages 211–228, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
[66] Tomas Toft. *Topics in Cryptology – CT-RSA 2009: The Cryptographers' Track at the RSA Conference 2009, San Francisco, CA, USA, April 20-24, 2009. Proceedings*, chapter Constant-Rounds, Almost-Linear Bit-Decomposition of Secret Shared Values, pages 357–371. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.
[67] Tomas Toft et al. Primitives and applications for multi-party computation. *Unpublished doctoral dissertation, University of Aarhus, Denmark*, 2007.
[68] Ramarathnam Venkatesan, Vijay Vazirani, and Saurabh Sinha. A graph theoretic approach to software watermarking. In Ira S. Moskowitz, editor, *Information Hiding*, pages 157–168, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
[69] T. Veugen. Linear round bit-decomposition of secret-shared values. *Information Forensics and Security, IEEE Transactions on*, 10(3):498–506, March 2015.
[70] Yohei Watanabe and Junji Shikata. Timed-release secret sharing scheme with information theoretic security. *CoRR*, abs/1401.5895, 2014.
[71] Yohei Watanabe and Junji Shikata. Timed-release computational secret sharing and threshold encryption. *Designs, Codes and Cryptography*, 86(1):17–54, 2018.
[72] 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*, 54(2):205–209, Feb 2007.