Collusion-Deterrent Threshold Information Escrow

Easwar Vivek Mangipudi
Purdue University
emangipu@purdue.edu

Donghang Lu
Purdue University
lu562@purdue.edu

Alexandros Psomas
Purdue University
apsomas@purdue.edu

Aniket Kate
Purdue University
aniket@purdue.edu

***Abstract*—An information escrow (IE) service allows its users to encrypt a message such that the message is unlocked only when a user-specified condition is satisfied. Its instantiations include timed-release encryption and allegation escrows with applications ranging from e-auctions to the #metoo movement. The proposed IE systems typically employ threshold cryptography towards mitigating the single-point-of-failure problem. Here, a set of escrow agents securely realize the IE functionality as long as a threshold or more agents behave honestly. Nevertheless, these threshold information escrow (TIE) protocols are vulnerable to premature and undetectable unlocking of messages through *collusion* among rational agents offering the IE service.**

This work presents a provably secure TIE scheme in the mixed-behavior model consisting of rational and malicious escrow agents.; any collusion attempt among the agents towards premature decryption results in penalization through a loss of (crypto-)currency and getting banned from the system. The proposed collusion-deterrent escrow (CDE) scheme introduces a novel incentive-penalty mechanism among the agents to stay honest until the user-specified decryption condition is met. In particular, each agent makes a cryptocurrency deposit before the start of the protocol instance such that the deposit amount is returned to the agent when the user-specified condition is met or can be transferred by anyone who holds a secret key corresponding to a public key associated with the instance. Using a novel combination of oblivious transfer, robust bit watermarking, and secure multi-party computation, CDE ensures that whenever the agents collude to decrypt the user data prematurely, one or more whistle-blower agents can withdraw/transfer the deposits of all other agents, thereby penalizing them. We model collusion as a game induced among rational agents offering the CDE service and show that the agents do not collude at equilibrium in game-theoretic terms. We also present a prototype implementation of the CDE protocol and demonstrate its efficiency towards use in practice.

While this work does not aim to solve the collusion problem fully, it significantly raises the bar for collusion. It offers an important step towards weakening the strong non-collusion assumption pervasive across multi-party computation applications.

I. Introduction

An information escrow (IE) service [10] allows a user to encrypt her sensitive message to some *condition*, such that the message can only be revealed after the condition is met. This condition can be anything the user chooses and can be checked by a program (or smart contract). For example, these escrows can include checking for share value of some company hitting a particular value, temperatures rising to a certain level to release funds for environmental programs, allegation escrows [9], [63], [81] or timed-release encryption [29], [37], [73], [82] where users send messages to the future. The most popular among them is timed-release encryption (TRE) [29], [37], [73], [82], which involves encrypting a message for a specific time period such that it cannot be decrypted before the period ends. Applications for TRE include e-voting [27], sealed-bid auctions [26] and client puzzles in the challenge-response systems [44].

Offering information escrows, especially for software, is also prevalent in the industry. Companies like Escrowtech [2] and Iron Mountain [3] provide software escrow services to technology companies that routinely appear in the Fortune 500 list. However, firms offering the service act as trusted third parties. Towards avoiding this single-point-of-failure, many systems [9], [16], [28], [29], [37], [63], [80]–[82], [91] realize IE using threshold cryptography [17], [36], [38] and making a non-collusion assumption among a group of agents.

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

If the escrow agents running a TIE service decide to collude and open a message before the attached condition is satisfied, they can do it passively in an undetectable manner. As the agents may collude among themselves in an undetectable manner, it is difficult to prevent such collusion over a long time; unless the protocol design itself makes collusion non-profitable. This work aims to design such a TIE protocol where the best strategy for any rational agent will be not to 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 disincentivizes collusion; the agents will not attempt to collude to open any user message for fear of losing their deposits and getting banned from offering any future services. Non-collusion assumption is highly prevalent in the distributed cryptographic literature, [16], [28], [45], [46], [79], [91] and overcoming it has been a significant barrier for the community for a long time. In this work, we address collusion among the multi-party (MPC) computation agents through 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, we let all the agents be rational rather than honest in this work, 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 formally define and use a cryptographic primitive called distributed receiver oblivious transfer (DROT). DROT is a natural distributed version of the oblivious transfer [38] 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 [45], [79], and secure bit decomposition [24], [86], [89] to securely realize $\mathcal{F}_{CDE}$ in the mixed-behavior model [71] where the agents are either rational or malicious. The protocol supports any condition that can be checked through a blockchain smart contract (even through interaction with the real world) as a condition of data release.

We implement the CDE protocol using SCALE-MAMBA [33] and HoneybadgerMPC [69] libraries. Our prototype implementation (see Appendix B) shows that the system realizing the CDE takes less than a minute to set up and ~ 120 milliseconds of interaction to transfer key shares to the service agents.

Organization of the paper. Section II describes the system setup, problem definition and gives an overview of the solution. Section III introduces the required multi-party computation modules, robust watermarking, and a claim-or-refund smart contract required to realize the collusion deterrent escrow (CDE) protocol. Section IV describes the different steps and algorithms of the proposed CDE protocol, and Section V 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 VI offers the security definition, Sections VII, discuss the related work. Appendix B provides the implementation details and the various times taken for different steps of the protocol.

II. Technical Overview

A. System Model

The system consists of n agents who offer a threshold information escrow (TIE) service and a user engaging with the service in a multi-party computation (MPC) setting. The agents are associated with fixed identities (typically connected with real-world identities); the user verifies the identities of the agents before engaging the service.

We consider the *mixed-behavior* model [71] where the agents are either *rational or malicious*. Rational agents aim to maximize their utility at any given point in time, and the malicious ones can deviate from the protocol arbitrarily. Any number of agents among the n agents can collude to increase their utility. A malicious adversary can make the corrupted parties deviate arbitrarily from the protocol. We consider a $t-$bounded adversary under a static corruption setting:[1] Up to t of n parties can be corrupted before the start of the protocol execution and those parties remain corrupted throughout the execution. The adversary can corrupt a maximum of t parties such that $n \geq 2t+1$.

Communication Model: All the parties are connected through secure and authenticated point-to-point links and have access to reliable broadcast channel [45], [60]. We assume the network to be bounded synchronous [9], [47] and the execution occurs in discrete rounds. Any message generated in round i is delivered at the beginning of round $i+1$. In practice this is achieved by setting a maximum publicly known time bound on message transmission. The message is set to $\perp$ if no message is delivered by beginning of the next round Refer [21], [56] for the corresponding ideal functionality. The adversary is informed whenever communication takes place between any two parties and the attacker can arbitrarily delay the delivery of the messages between honest parties within the round boundaries.

The list of all banned agents in the system is available to all the parties (for example, through a blockchain). The users verify that the agents offering the service are not banned before the start of the protocol.

Problem Definition. The user has a private message that she wishes to encrypt to a certain condition and n agents offer the information escrow service. Each agent 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. The user chooses this subset at the time of using the service. In this setting, we wish to achieve the following security goals:

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

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

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

B. Key Idea

The escrow agents offer the information escrow service. A public-secret key pair (pk_τ, sk_τ) is associated with a condition τ, corresponding to which the user's document is encrypted.

[1]The employed protocols secure against the static adversary can be made secure against an adaptive adversary using standard techniques [22], [54].

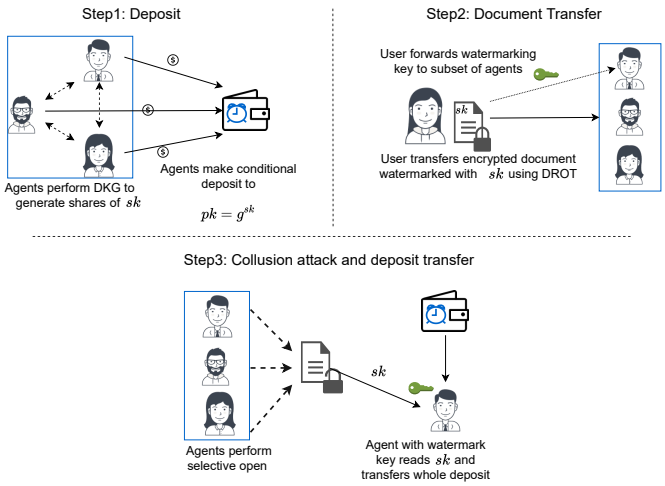

Fig. 1: Steps involved in the collusion deterrent escrow mechanism. Agents use a Distributed key generation (DKG) protocol to generate shares of secret key sk corresponding to public key pk. The user employs a Distributed receiver oblivious transfer (DROT) protocol to transfer a copy of the document to the agents such that it contains the secret key sk as the watermark.

Each secret key sk[2] is $(n, t+1)$ secret-shared among the agents using Shamir secret sharing (SSS) [84], where at-least $t+2$ agents are needed to reconstruct the secret. The key idea involves MPC among the agents and the user such that the user transfers a copy of the document *watermarked with the secret key* sk to the agents. Unless at least $t+2$ agents collaborate, none of the agents or the user can determine the transferred, watermarked document. When they collaborate to decrypt the watermarked document, it can reveal the watermark (i.e., the secret key sk). The watermark detection key is provided to all the agents.

Before interacting with the user, all the agents make a claim-or-refund cryptocurrency deposit to an address associated with pk with the user-defined condition τ; here, the funds can be claimed before the condition is met using sk or the respective agents get the refund after the condition is met. If the agents collude to reveal the watermarked document, any agent with access to the watermarking detection key can compute the watermark sk and claim all the deposits of all agents, including his own. If the deposits are claimed (visible publicly on the blockchain) before the condition is met, *all the agents except the transferring agent are banned* from offering the future IE service. If the deposit is not transferred to a known public address of an agent *i.e.,* if the identity of the transferring agent is not known, all the agents are banned. No agent attempts to collude for fear of losing the deposit and getting banned from the system. We prove that the best response strategy of the agents is to *not* collude (refer Section V for a detailed analysis). Fig. 1 depicts the three steps of making a conditional deposit, document transfer from the user, and agents attempting to collude with one of the agents with watermarking key transferring the deposits of all agents to himself.

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

The agents are typically firms (like Escrowtech [2]) offering their servers and participating in the protocol. Before the start of the protocol, the agents provide their physical identities which can be verified by the users. The users should be able to verify the identity of the agent before engaging in the protocol. In case of collusion, these identities will be banned from offering the services further. Here, note that the banning is not just for the current instance of the protocol but *any* future service. Since the identities are public and known, the agent can not offer any other service late – the agent is banned for life.

Trivially, at any point of time before the expiry of the escrow condition, the agents can collude and construct the secret key sk and transfer and distribute the deposits among themselves. However the banning policy ensures that the cost incurred by getting banned (and losing all the payments from any future service offerings), is higher than any payoff obtained from collusion and decryption of any single document. For simplicity, in the analysis of the protocol, we assume that all the agents are similar *i.e.,* how they value the document, the probability with which they succeed when transferring the deposit, the costs they incur are all similar. Any differences in these parameters will only deter the agents further from collusion. We discuss these assumptions further in V. With a (encrypted) document of value d, pay-off v for offering the escrow service for the document and assuming a future pay-off of all the services F for each agent, the expected pay-off obtained by each agent upon collusion in a two agent scenario is $v + \frac{d}{2} - \frac{F}{2}$. Since $F \gg d, v$, the pay-off is strictly less than v, the pay-off obtained without any collusion deterring collusion.

In summary: with public rational agents offering the escrow service, when the user employs robust bit watermarking to watermark the data and assuming the agents incur net pay-off loss from banning, our CDE protocol ensures that no rational agent attempts to collude. Non-collusion is the best response strategy of the rational agents in the CDE protocol.

C. Protocol Steps

At the beginning of the protocol, the agents run a distributed key generation (DKG) scheme [45] to generate a public key pk and a $(n, t+1)$ SSS of the corresponding secret key sk, with each agent A_j for $j \in [1, n]$ receiving the j^{th} share $[\![sk]\!]_j$. Here, an $(n, t+1)$ threshold secret sharing requires at least $t+2$ agents to reconstruct the secret.

The agents run a secure bit decomposition protocol [24], [86], [89] on the shared key sk to obtain the SSS shares of the secret key sk bits; i.e., each agent A_j now have a SSS share of the bit $sk_i, i \in [0, \lambda - 1]$ (sk_i is the i^{th} bit of the λ-bit secret key sk), denoted by $[\![sk_i]\!]_j$. A public bulletin board server is available to all the users and agents where they publish non-secret information and encrypted data whenever needed. The public key pk is stored on the server, each agent A_j 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 TIE service for the message m, she splits the message/document

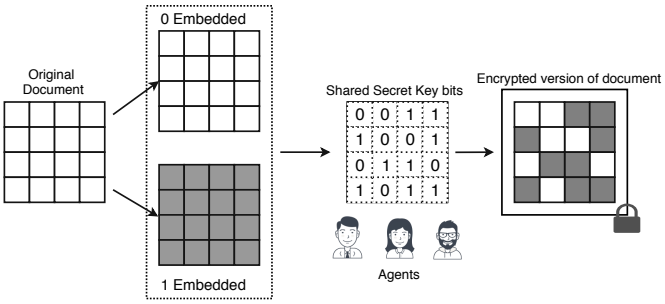

Fig. 2: Watermarking the document blocks and transferring them to the agents. Two versions of each block are obtained by watermarking $0, 1$. The secret key bits are shared among the agents. Depending on the bit of the secret key, the transferred document block consists of the corresponding bit as the watermark. Thus final transferred encrypted document contains the whole secret key as the watermark.

into λ parts (λ is the bit length of secret key sk) and watermarks each part with robust bit watermarking to generate two versions of each part; i.e., for each part m_i for $i \in [0, \lambda-1]$, she computes two watermarked parts $(m_{i,0}, m_{i,1})$ with watermarks corresponding to bit 0 and 1. She symmetric-key encrypts each part $m_{i,b}$ for $i \in [0, \lambda-1]$ and $b \in \{0,1\}$ using randomly sampled keys $\mathsf{k}_{i,b}$ to obtain the ciphertexts $c_{i,b}$.

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

The aim of the user is to transfer an encrypted version of each document part such that the transferred part is watermarked with a secret key bit that is shared among all the agents. If the encrypted document part is decrypted, the decrypted part would reveal a secret key bit to whoever can read the watermark. At a later point of time, when the agents decrypt the encryption c_{i,sk_i} of the watermarked message m_{i,sk_i}, the detection key would be necessary to detect the watermark. The user forwards this detection key to all the agents 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 $\mathsf{k}_{i,sk_i}, i \in [0, \lambda-1]$ by combining the shares $[\![\mathsf{k}_{i,sk_i}]\!]$. The cipher texts c_{i,sk_i} are decrypted using the reconstructed keys to reveal the message parts m_{i,sk_i}. All the revealed message parts are combined to form a watermarked version m' of the message/-document m.

Collusion and Key revelation. The agents may decide to collude and decrypt the message m by reconstructing the keys $\mathsf{k}_{(\cdot,\cdot)}$ even before the user condition is met. However, the decrypted message version m' would contain the secret key sk as a watermark. The two-party computation of oblivious transfer functionality ensures that each version of the message part that the agents can decrypt contains the secret key bit sk_i as a watermark. The secret key is revealed when all the watermarked bits are read from the message parts. Any agent who has access to the watermark detection key can read the secret key sk (and any information encrypted to the public key pk) and transfer all the deposits to an address of his choice. This is the *revealing* and penalizing property of the protocol. When the funds are publicly transferred on the blockchain before the user condition is met, all the agents except the agent who performed the transfer are banned from the system.

The agents may also collude to reconstruct the secret key from the shares and transfer to an address different from any of the agents' addresses. In this case, the transfer is publicly visible and *all the agents* are banned from the system. Any evidence of collusion, including signature generated through the embedded key by multi party computation can result in banning of all the agents – if the deposit is not transferred by a single agent to his/her address. In case the agents try to attack by removing the watermark in the received documents, the robustness of the watermarking ensures that when the agents try to remove the watermark, the data itself is damaged or rendered useless. Thus robustness of the watermarking is critical to our protocol. If the watermarking scheme is not robust, the agents would decrypt the document blocks, remove the watermarks and reconstruct the whole document. With the watermark is removed, no single agent would be able to transfer the deposit and hence the penalizing property of the protocol is lost. However, it may be noted that, in practice a weaker version of 'robustness' is sufficient for our protocol, which states that the watermarking scheme should have no known attacks. See Section III-B for further details.

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

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

III. Building Blocks

Here, we describe the basic building blocks employed by the proposed Collusion Deterrent Escrow protocol.

A. MPC Primitives

Multi-party computation (MPC) [11], [33], [69], [77] is an approach allowing mutually distrusting parties to compute some functions with their private input collaboratively. We use MPC modules for distributed key-generation, bit-decomposition, and two-party computation between the user and each agent. 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 [12] are used to multiply two secret shares.

Distributed key generation—DKG. A $(n, t+1)$ DKG [45], [79] 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 a size less than $t+2$ has any knowledge of it. A DKG mechanism is defined by two phases, the sharing phase at the end of which every party holds a share $[\![sk]\!]_j$ of the key sk, and the reconstruction phase involving every node broadcasting their share and running the reconstruction algorithm on the collected shares. The two algorithms for share generation and reconstruction are:

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

Bit decomposition. A bit decomposition protocol [24], [86], [89] takes a secret share as input and transforms the share into bit-wise shared values *i.e.,* for a value sk, upon input of all the shares $[\![sk]\!]$, the protocol outputs the shares $[\![sk_i]\!]$, where $sk_i, i \in [0, \lambda-1]$ are the bits of the value sk. It is defined by two algorithms:

- $\mathsf{bit.decomp}(\lambda, [\![sk]\!])$ takes in the total number of users and the shares of the secret key sk and returns to every agent A_j a vector $[\![sk_i]\!]_j$ of shares of the bits of the secret key.
- $\mathsf{bit.recon}([\![sk_i]\!])$ takes in the vector of shares of a particular bit and returns the reconstructed bit sk_i.

Oblivious linear function evaluation—OLE. OLE is a two-party computation protocol where the sender has inputs $a, b \in \mathbb{Z}_p$ and the receiver has the input $x \in \mathbb{Z}_p$. After running the protocol, the receiver obtains the value $a+(b-a)\cdot x \in \mathbb{Z}_p$ and the sender obtains $\top$ if the protocol run is successful (refer Fig. 3). The sender does not have any information about x or the value obtained by the receiver. There are several constructions of OLE [32], [39], [40], [48], [58]; in this work we consider OLE as a UC-secure blackbox. The ideal functionality of OLE is provided in Fig. 8 in the Appendix.

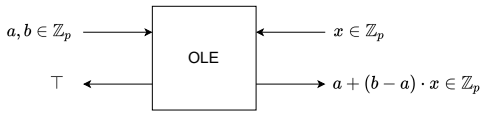

Fig. 3: Oblivious linear function evaluation (OLE) [39]. The sender has two inputs $a, b \in \mathbb{Z}_p$, the reciever has input $x \in \mathbb{Z}_p$, the receiver obtains the value $a+(b-a)\cdot x \in \mathbb{Z}_p$ obliviously.

B. Robust Bit Watermarking

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

- $\mathsf{wm.gen}(\eta)$: Given η, outputs keys $k_{emd}, k_{det} \in \mathcal{K}$ probabilistically.
- $\mathsf{wm.embed}(M, w, k_{emd})$: Takes the document M, watermark $w \in \mathcal{W}$ and embedding key k_{emd} as inputs and generates a watermarked document M'.
- $\mathsf{wm.detect}(M', k_{det}, w)$: 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 [4], [52], [68], [85]. We assume a given similarity function $sim(M, M')$ which returns $\perp$ if the two documents M and M' are not similar and $\top$ if they are.

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

Watermarking can be viewed as a mechanism to create a communication channel that is multiplexed into original document data [25], [55]. Here, robustness requires that the capacity of the watermark channel degrades as a smooth function of the degradation of the marked content. Since our watermark message is only one bit of data, when the watermark channel is fully utilized, corruption of the watermarked bit (by distorting full watermark-channel capacity) corrupts the whole document data rendering it unusable. Cayre *et al.* [25] provide a security analysis of watermarking schemes and show that schemes like substitutive watermarking provide perfect covering where no watermarking information is leaked or the attacker can not modify the watermarking signal when he has access to only watermarked copies of the data. In our work, the user forwards only watermarked copies of data to the agents during the transfer of the document.

The CDE protocol uses a robust watermarking scheme to watermark just either the bit 0 or the bit 1. The actual

watermarking scheme varies depending on the type of the data being watermarked. Our robustness definition is a weaker notion of traditional notion of robustness [25] which states that the any attempt to remove the watermark should result is destruction of watermarked data. In terms of real-world applicability, not all watermarking schemes that are currently used in practice for different kinds of data [70], [76], [78] have a theoretical proof of robustness. However, for our protocol to be applicable we do need such a proof as long as there is no known attack (till the end of expiration of escrow condition) on the watermarking scheme being employed. This is reflected in the robustness definition provided above. While theoretically, an algorithm may exist which can remove the watermark from the data, we just require that such an algorithm should not be available or known to humans; Rogaway [83] formalized this approach. We also need the watermarking to be imperceptible and the watermarked data to be 'similar' to un-watermarked data implying no loss of information from the original data. We do not need cryptographic indistinguishability of watermarked and un-watermarked data because the agent is aware that the data is watermarked, they do not (and should not) know what the bit that is watermarked in the data.

C. Claim-or-refund deposit

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

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

Algorithm 1 Claim-or-refund contract

```
if Escrow Condition == True then
    Direct the locked funds back to the contract creator
else
    if signature corresponding to public key pk of protocol instance is valid then
        Direct the funds to the mentioned recipient
    else
        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* [1] which interact with the outside world with different APIs for information like weather parameters etc. Depending on the trust imposed on the oracles by the user and the agents, they can agree on the oracles and the deposit value before starting the protocol.

IV. 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 $\mathsf{A}_j, j \in [1, n]$ constitutes the following algorithms:

- $\mathsf{KeySetup}(n, t, \lambda)$ 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.
- $\mathsf{MessageSetup}(m, \lambda, pk)$ outputs λ pairs of encryptions of binary-watermarked parts of the message m encrypted to the public key pk.
- $\mathsf{DROT}\left((\mathsf{k}_{i,0}, \mathsf{k}_{i,1}), [\![sk_i]\!]\right)$ The DROT protocol takes the keys $\mathsf{k}_{i,0}, \mathsf{k}_{i,1}$ from the sender and the shares of the bit sk_i from the receivers and transfers sharing $[\![\mathsf{k}_{i,sk_i}]\!]$ corresponding to the bit sk_i to the receivers for $i \in [0, \lambda - 1]$.
- $\mathsf{Open}\left([\![sk_i]\!], [\![\mathsf{k}_{i,sk_i}]\!], c_{i,b}, b \in \{0,1\} i \in [0, \lambda - 1]\right)$ takes the shares of secret key bits along with the cipher texts, decrypts the ciphertexts forming the message blocks and outputs the final combined message.

A. Cryptographic Setup

$\mathsf{KeySetup}(n, t, \lambda, \eta)$. 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 $\mathsf{grp.gen}(\cdot)$ which takes the security parameter λ as input. It generates the cyclic group $\mathbb{G}$ of prime order p and two generators g, h. The two generators are used for Pedersen commitments. The agents run the distributed key generation (DKG) algorithm $\mathsf{dkg.gen}(\cdot)$ to generate the public key pk and the vector of secret key shares $[\![sk]\!]$, with each agent A_j, $j \in [1, n]$ obtaining the share $[\![sk]\!]_j$. The agents run a bit decomposition algorithm $\mathsf{bit.decomp}(\cdot)$ with the shares to obtain the bit-wise threshold-shares $[\![sk_i]\!]_j$ of the bits $sk_i, i \in [0, \lambda - 1]$ for each agent. Here, η is the security parameter and $\lambda = poly(\eta)$. Algorithm 2 describes $\mathsf{KeySetup}$.

$\mathsf{MessageSetup}(m, \lambda, k_{emd})$. It is run by the sender who takes the message m and uses an algorithm $\mathsf{split}(m, \lambda)$ to divide it into λ parts $(m_0, \ldots, m_{\lambda-1})$ where λ is the bit-length of the secret key sk. The sender forms the robust binary watermarked versions of the message parts (using $\mathsf{wm.embed}(\cdot)$) with the watermark embedding key k_{emd}, watermarked with $\{0,1\}$ to obtain $\{(m_{0,0}, m_{0,1}), \cdots, (m_{\lambda-1,0}, m_{\lambda-1,1})\}$ and encrypts them using the randomly chosen symmetric keys $\mathsf{k}_{i,b}$ to produce $c_{i,b}, i \in [0, \lambda - 1], b \in \{0,1\}$. The $\mathsf{split}(\cdot)$ is a simple split/chopping of the message into parts. It is not a cryptographic secret sharing of the message. This is because, reconstruction of the data from watermarked secret shares may not result in the same or meaningful reconstruction of the data.

Algorithm 2 KeySetup (n, t, λ, η)

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

Algorithm 3 MessageSetup(m, λ, k_{emd})

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

Fig. 4: KeySetup and MessageSetup algorithms

The watermark may also be affected upon such a reconstruction. Algorithm 3 depicts algorithm MessageSetup.

B. Distributed Receiver Oblivious Transfer—DROT

Oblivious transfer is a 2-party computation protocol, in which the sender has two messages m_0, m_1 and the receiver has the bit c; at the end of the protocol, the receiver receives the message m_c. The protocol ensures that the sender has no information of c and the receiver has no information about m_{1-c}. We developed a multi-party version of oblivious transfer called the Distributed Receiver Oblivious Transfer protocol (DROT) 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 $\llbracket s \rrbracket$ for the bit s. At the end of the computation, each of the receivers receives the shares $\llbracket \mathsf{k}_s \rrbracket$. 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.

Ideal functionality of DROT. The functionality (see Figure 5) interacts with the sender S and the n receivers $\mathsf{A}_j, j \in [1, n]$. S has the messages $\mathsf{k}_0, \mathsf{k}_1$ and each receiver A_j has the share $\llbracket s \rrbracket$ of the bit s. The adversary $\mathcal{A}$ can corrupt a total of t parties in the system — the sender or upto t receivers or sender and $t-1$ receivers. The sender initiates by forwarding the input by sending the $\texttt{inputS}$ message with the two messages $\mathsf{k}_0, \mathsf{k}_1$ and the session id sid to $\mathcal{F}_{\mathsf{DROT}}$. The functionality stores them and informs the adversary by sending the $(\texttt{input}, sid)$ message. It also informs the receivers that the protocol has been initiated by sending the (intd, sid) messages. The receivers forward their shares $\llbracket s \rrbracket_j$ using the message $\texttt{inputR}$ to $\mathcal{F}_{DROT}$, whose id j is stored in the set $\mathcal{I}$. After receiving at least $t+2$ shares, the functionality computes the bit s by combining the shares $\llbracket s \rrbracket_j$. When $\mathcal{A}$ sends the $(\texttt{deliverS}, sid)$ message, the functionality checks if the inputs from the sender and at least $t+2$ receivers have been received. If not, the message is ignored. The sender is informed by forwarding the $(\texttt{delivered}, sid)$ message to S. $\mathcal{A}$ sends the $(\texttt{deliverR}, sid)$ message to release the output

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

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

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

Fig. 5: Ideal Functionality Of DROT

to the receivers. On receiving the $\texttt{deliverR}$ message, the functionality checks if $|\mathcal{I}| \geq t+2$, if at least $t+2$ receivers have forwarded to release the output. If yes, it generates shares of the value k_s and forwards the share $\llbracket \mathsf{k}_s \rrbracket_j$ to the receivers $\mathsf{A}_j, j \in \mathcal{I}$.

Protocol. We realize DROT using multiple instances of two-party computation, realizing an oblivious linear function evaluation (OLE) [39] between the sender and the receivers. In DROT, the OLE computation is run by the sender with all the n receivers $\mathsf{A}_j, j \in [1, n]$. The input of the sender is messages $(\mathsf{k}_0, \mathsf{k}_1)$ and the input of each receiver is share $\llbracket s \rrbracket_j$ of the bit s. $\llbracket s \rrbracket_j$ are $(n, t+1)$ shared values of the value s. After the DROT protocol run, the receivers obtain the value $\mathsf{k}_0 + (\mathsf{k}_1 - \mathsf{k}_0) \cdot \llbracket s \rrbracket_j$. These are the shares $\llbracket \mathsf{k}_s \rrbracket_j, j \in [1, n]$ of the value k_s, such that any $t+2$ or more parties can reconstruct the value k_s. The receivers prove in zero-knowledge that the input shares indeed correspond to a bit s.

The Collusion Deterrent Escrow (CDE) uses the DROT protocol to transfer the encrypted message blocks and the corresponding keys' shares to the agents. The user encrypts the message blocks $m_{i,b}$, using keys $\mathsf{k}_{i,b}$ to obtain $c_{i,b}$ for $i \in [0, \lambda - 1]$ and $b \in \{0, 1\}$ and broadcasts the ciphertexts to all the agents. User transfers shares of keys k_{i,sk_i} to the agents using DROT. As a part of DROT the user runs λ instances of OLE— for each instance i, the input of the user is $(\mathsf{k}_{i,0}, \mathsf{k}_{i,1})$ where the inputs of the agents would be shares $\llbracket sk_i \rrbracket_j$ of the bit sk_i of the signing key sk. At the end of the runs, each agent A_j has the values $\llbracket \mathsf{k}_{i,sk_i} \rrbracket_j$ which are shares of the values $\mathsf{k}_{i,sk_i} = \mathsf{k}_{i,0} - (\mathsf{k}_{i,1} - \mathsf{k}_{i,0}) \cdot sk_i$. While running DROT for CDE, the agents prove to the user that the input bit shares indeed correspond to bits of the secret key sk by forwarding a zero knowledge proof π_{sk}; this is a NIZK proof [20] of a discrete log statement. This way of directly computing the shares of the keys k_{i,sk_i} prevents the agents

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

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

from learning the other key corresponding to $\mathsf{k}_{i,1-sk_i}$. The oblivious transfer functionality can be realized using standard oblivious transfer primitives, however, they involve computing hash functions in a distributed manner which is computationally intensive. Realizing the oblivious transfer functionality as oblivious linear function evaluation using secret sharing based 2-party computation avoids the computational bottlenecks.

The security analysis of DROT is discussed in Section VI. For security, DROT employs multiple instances of secure two party OLE protocol.

C. 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 $[\![\mathsf{k}_{i,sk_i}]\!], i \in [0, \lambda-1]$ for each i and the corresponding ciphertext c_{i,sk_i} is decrypted to m_{i,sk_i}. After all parts of the messages m_{i,sk_i} 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. Algorithm 4 presents algorithm Open.

D. Escrow deposits

Before starting the protocol, the user and the agents agree on the deposit value D. The agents deploy smart contracts embedding the condition τ specified by the user to the address corresponding to the public key pk. Each agent A_j creates a deposit transaction TX_j for the value D and the user verifies that the agreed-on values of funds have been held in the deposit. When the condition τ is met, the deposits can be immediately transferred back to the agents. Any agent with sk can claim/transfer all the deposits to the address of his choice before the condition τ is met. If the deposit is ever transferred before τ, all the agents except the agent that transferred the deposit are banned from offering any future service thereby penalizing them. This ban can be either permanent or for a specified period.

In the non-colluding scenario, after the condition τ is met, we expect the agents to take a non-zero time to compute the secret key sk. Indeed, we expect that the deposits are transferred back instantaneously to all the agents by the smart contracts before the sk is computed by the agents once τ is met. Once τ is met, there is no secret information in the protocol instance anymore.

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

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

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

A. Considerations and assumptions

Before we analyze the system, we mention some of the assumptions under which the proposed protocol is applicable and practical:

- When multiple agents try to transfer the deposit after collusion, each agent transfers (wins the race) with equal probability. We make this assumption for ease of analysis. However, if the probabilities are different, it deters the agents further to collude since they do not know if they have a higher/lower probability of success.

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

- All the agents are similar. No agent(s) can impose negative pay-offs on any other agent outside the protocol, for example, using blackmail etc. However, they may offer positive pay-offs like bribes to encourage a certain strategy.

- At any point in time, the expected reward for offering the service in the future for the agents is F. The expected reward of all future receivable payments (F) for the services to be offered is greater than the value v and d of individual documents encrypted for any time period. F is not just the receivable payments from future payments of the *current* service offering but any other future business offering by the agents. It is because the identities are physically verifiable, and any banning is public. Also, any deviation from the protocol by early deposit transfer is publicly recorded on the blockchain and serves as verifiable evidence in the future.

B. Games and Pay-offs

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 I. 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 I with only the *accept* and *reject* strategies. *accept* indicates the strategy of offering the service without collusion. Each

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

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

agent makes a deposit of value D before the user interacts with them.

The agents are free to interact; they can collude 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 I 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 [75], and hence both the agents play the *accept and collude* strategy at equilibrium.

To prevent such a collusion attack and to prevent *accept and collude* as the equilibrium strategy, the user who acts as a principal/regulator designs a mechanism implemented by offering the grand contract. In our protocol, the agents collude to attack the system either by decrypting the document or by reconstructing the secret key. A collusion game is induced among the agents in either of these scenarios.

In the implementation of the game as a protocol, $D > 0$ is the value of each agent's conditional deposit, and F is the approximate sum of future payments to be received if the agent/node is not banned from the system. 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 $F \gg 0$.

C. Collusion-game

A Collusion game is induced when the agents collude and decrypt the document either by threshold decryption or secret key reconstruction. The following parameters define the game.

- The set of agents $N = \{A_1, A_2\}$
- The strategy space of each agent j is S_j = {*wait*, *transfer*}
- The utilities corresponding to the strategies of the agent $u_j(s_j, s_{-j}), s_j \in S_j$. s_{-j} indicates the strategy(ies) of player(s) other than player j.

When the agents collude and decrypt the document, the colluding agent can detect the watermark and can transfer the deposit. However, he can choose not to transfer the deposit and wait. These two actions are indicated by *transfer* and *wait*.

Pay-offs. The expected pay-offs of the agents under Collusion is captured by the pay-off matrices in Table II. When the agents decrypt the user document and decide not to transfer the deposit (play *wait*), they both can accrue a pay-off of $\alpha = v + \frac{d}{2}$. If one of the agents transfers (plays *transfer*) 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 $\alpha - (F + D)$. When the agents attempt to transfer the deposit simultaneously, since only one agent can succeed in the transfer, it creates a race condition and we assume that each will succeed in the transfer with equal probability [3]; hence in expectation they accrue a pay-off $0.5(\alpha + D) + 0.5(\alpha - F - D) = \alpha - \frac{F}{2}$. Here, $u_j(transfer, s_{-j}) > u_j(wait, s_{-j}) \forall s_{-j} \in \{transfer, wait\}$ (see Table II). For an agent *transfer* is a strictly dominant strategy, he always plays *transfer*. The dominant strategies have been indicated in grey in Table II; one can observe that the pay-offs are similar to the pay-offs in well-known prisoners' dilemma [62]. The agents playing their dominant strategy indeed forms the Nash Equilibrium as defined below.

Definition 1 (Nash Equilibrium [61])**.** *Given a strategic form game $\Gamma = (N, (S_i), (u_i))$, the strategy profile $s^* = (s_1^*, s_2^*, \cdots, s_n^*)$ is called pure strategy Nash equilibrium of Γ if $u_i(s_i^*, s_{-i}^*) \geq u_i(s_i, s_{-i}) \forall s_i \in S_i \forall i$. In words, no single player, by changing his own part of strategy profile can obtain higher utility if the others stick to theirs.*

Both the agents play *transfer* as their dominant strategy, accruing a pay-off of $\beta = \alpha - \frac{F}{2} = v + \frac{d}{2} - \frac{F}{2}$. As F is the sum of all future payments from many users, we have, $F \gg D, d, v$. The pay-off from the collusion game β is strictly less than v making collusion unviable.

n agents. Similar to the two agent scenario in Collusion it is a dominant strategy for the agents to play *transfer*. If one agent plays *transfer*, he obtains a pay-off of $\alpha + (n-1)D$ [4] and every other agent obtains $\alpha - (F + D)$. When more than one agent plays *transfer*, each agent obtains $\alpha + (n-1)D$ with equal probability. With $t+1$ other agents in the system each playing their dominant strategy *transfer*, each agent obtains a positive pay-off of $\alpha + (n-1)D$ with a probability of $\frac{1}{t+2}$ and obtains a pay-off of $\alpha - F - D$ with probability $\frac{t+1}{t+2}$. Hence the expected pay-off of an agent playing his dominant strategy with $t+1$ *other* agents is:

$$\beta = \frac{1}{t+2}(\alpha + (n-1)D) + \frac{t+1}{t+2}(\alpha - F - D)$$

$$= \frac{(t+2)\alpha + (n-(t+2))D - (t+1)F}{t+2}$$

Similar to the two agent scenario, with $F \gg d, v, D$, we have $\beta < v$.

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

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

TABLE II: Pay-offs of rational agents under different strategies under Collusion scenarios - threshold decryption and secret key reconstruction. The agents always play the dominant strategies shown in grey with $F \gg D, \alpha$.

A_2 \ A_1	*wait*	*transfer*
wait	(α, α)	$(\alpha + D, \alpha - F - D)$
transfer	$(\alpha - F - D, \alpha + D)$	$(\alpha - \frac{F}{2}, \alpha - \frac{F}{2})$

Bargaining: Apart from the collusion scenario defined, each agent may indulge in bargaining/bribing by offering a positive payment through the side contract to other agents to prevent them from playing *transfer*. The other agents can not impose a negative pay-off on the whistle blower (playing *transfer*); hence the agent will *always* play his dominant strategy of transfer to improve his pay-off even after accepting a payment from the other agent. From the strictly dominant strategies, it is clear that whenever the agents decide to collude, they always attempt to transfer the deposit and act as whistle-blowers in the protocol irrespective of the actions of other agents.

Theorem 1. *In CDE protocol under Nash equilibrium (ref Definition 1) with $n = 2$, the agents do not collude; for $n > 2$, no more than threshold $t+1$ agents collude when the secret key is shared in a $(n, t+1)$ threshold structure (at least $t+2$ agents are required to obtain secret information) under the assumptions listed in Section V-A and $F \gg D, d, v$.*

Proof: For $n = 2$ when the agents collude, the pay-off of each agent from dominant strategies is $\beta = (\alpha - \frac{F}{2})$. 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 I, under the CDE protocol mechanism changes to Table III. Since $\beta < v$, it is evident from Table III 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 (see Fig. 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'. ■

TABLE III: Pay-off matrix of the two agents under CDE. v, β are the pay-off when agents do not collude and collude. The agents do not collude since $\beta \ll v$

A_2 \ A_1	*reject*	*accept (not collude)*	*accept and collude*
reject	(0,0)	(0,0)	(0,0)
accept (not collude)	(0,0)	(v, v)	(v, v)
accept and collude	(0,0)	(v, v)	(β, β)

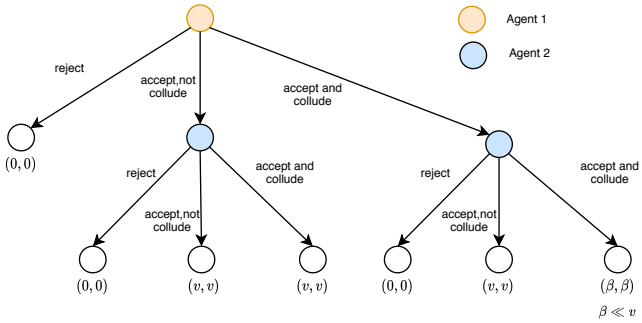

Fig. 6: Extensive form game induced by the user in CDE between two agents. v is the pay-off when agents do not collude and $\beta \ll v$ is the pay-off when the agents collude

In the current protocol design, the user forwards the watermark detection key to all the users after participating in the transfer protocol. We arrive at this setup from game-theoretic analysis of a setup where the user forwards the detection key to a random subset of agents. Here the user chooses to send the detection key to each agent with the probability p. This divides the agents, the players of the collusion game, into two types: players with the detection key called efficient agents and agents without called inefficient agents. The corresponding collusion games analyzing the two attacks scenarios and their corresponding pay-offs in this setup are provided Appendix C.

After collusion. Table II shows that during collusion irrespective of what the other agents' strategy is, the dominant strategy of any rational agent is to attempt to transfer the deposit to himself. Thus whenever collusion occurs (with $> t+2$ parties) irrespective of which set of agents collude during the collusion phase, every colluding rational agent attempts to transfer to himself leading to an expected pay-off strictly lower than obtained without collusion. The adversary controlling t agents can approach any rational agent to reveal all the secret shares with the adversary, however, since the threshold of secret information is $t+1$, at least 2 rational agents need to participate in collusion along with the adversary to transfer all the deposit. If more than $t+1$ agents participate in collusion, the expected pay-off is strictly less than without collusion. Thus no rational agent participates in the collusion.

In works involving rational secret sharing [49], [50], 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 (non-collusion) survives the iterated deletion of weakly dominant strategies and hence an equilibrium. What is a road-block in works like [49], [50], [57] 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.

D. Practical Issues

Multiple-rounds. The analysis considered only one round of play of the game. However, 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 (because of higher reputation), 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. Since the message blocks are segments of the message, the receivers may try to decrypt the document partially (only few message blocks and not the full document). This reveals only few bits of secret key instead of the full secret key to the agents. For example, when the document is divided into $\lambda = 256$ parts, the agents can collude and decrypt, say 26 parts or $\sim 10\%$, of the message. This will reveal 26 bits of the secret key. Thus when the key is embedded once, decryption of any $\ell < 256$ parts reveals only ℓ bits to the agents with the detection key. To prevent such a decryption, the user does the following: she divides the document in multiples γ of λ. Say for $\gamma = 64$, this would imply embedding 64 copies of the secret key into the document. Now the user divides the document into $64 * 256 = 16384$ parts, randomizes their order and inputs them into the protocol. Now if the agents decrypt $10\% = 1638$ parts randomly, in expectation they reveal 255 bits.

The user can divide the document among multiple instances of the protocol rather than one. When an agent with the detection key obtains the key partially, he can brute force the remaining bits of the key whenever possible. For valid decryption after the escrow condition is met, an index is appended to the the randomized message parts before encryption. The exact number of copies of the secret key to be embedded depends on the entropy of the data, the useful information it contains and the watermark channel capacity of the data – the analysis of which is beyond the scope of this document.

Choosing D, d, v. The agents agree on the value of D, d before the start of the protocol. One simpler practical approach would be to fix the values of D, v for any instance. This would put a limit on the maximum document value d that the user wishes to deposit. If the document it too valuable, the user may divide it among multiple runs of the protocol. Also, since the user is anonymous, the agents would not know how the user data is distributed across different instances of the protocol run. The service fee v can be paid in parts over the period of the escrow starting with an initial partial payment at the beginning of the escrow instance.

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

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

Init Session and user input:

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

Open message:

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

Release message:

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

Delivery:

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

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

Further issues. It may appear that some rational parties are at a risk of losing their deposit if other parties collude. However, for the collusion to occur at least two rational parties should participate in the collusion. Since the expected pay-off with collusion is less than pay-off without collusion, no rational party attempts to collude. Hence such scenario does not occur.

VI. Security Analysis

A. Security Definition

The system consists of n agents $\mathsf{A}_j, j \in [1, n]$ 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 a corrupt interface that takes an agent/user identifier and returns the internal state of the agent to the adversary. All subsequent incoming and outgoing communication of the agent is then routed through the adversary. The adversary is t-bounded, and

can corrupt up to t agents and the user. For formal security, as discussed earlier, we consider the static corruption model i.e., the adversary commits to the identifiers of the agents it wishes to corrupt ahead of time. The adversary is also informed whenever some communication happens between two agents and it can arbitrarily delay the delivery of the message between honest parties; however, it cannot drop messages between two honest agents or between the honest user and the honest agent.

Ideal Functionality. In our ideal functionality $\mathcal{F}_{CDE}$ (See Figure 7), the user chooses a set of agents $\mathcal{S}_{\mathcal{U}} = \{\mathsf{A}_{u_1}, \cdots, \mathsf{A}_{u_q}\}$ where $\mathcal{U} = \{u_1, \cdots, u_q\}$ is the set of indices. The user initiates the document transfer using the `inputU` message and forwards the data m_{U} and the set $\mathcal{S}_{\mathcal{U}}$ to $\mathcal{F}_{CDE}$ with the session id sid. $\mathcal{F}_{CDE}$ receives and stores $\langle \mathsf{U}, \mathcal{S}_{\mathcal{U}}, m_{\mathsf{U}}, sid \rangle$. The `intd` message is sent to all the agents indicating that the session has been initiated by the user. $\mathcal{U}$ is the set of indices of all agents to whom detection key is forwarded in the protocol implementation.

Each agent A_j can wish to open the user message, they forward the message $(\texttt{openR}, j, sid, \mathsf{U})$ to indicate they wish to open the message of the user U. The functionality stores the index in the set $\mathcal{Q}_{sid}$. Similarly, the users may wish to release the user message and forward $(\texttt{releaseR}, j, sid, \mathsf{U})$ to the functionality. Opening the message indicates the selective opening of the message, using threshold decryption in the protocol implementation. Releasing the message implies reconstructing the secret key using the shares of secret key bits among the agents and decrypting the user message. When the agents forward `openR` and `releaseR` messages from agents, the functionality informs the adversary by forwarding `open` and `release` messages respectively.

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

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

Towards analyzing the security of our protocols under the mixed-behaviour model [71], we offer theorem statements and proof sketches for ideal-real world security paradigm. We employ all our functional blocks in a black-box manner; we first consider the security of the DROT protocol from Section IV-B, which is the key cryptographic construction of CDE. DROT uses multiple instances of UC-Secure OLE protocol for forwarding the key shares to the agents. We present the simulator $\mathcal{S}_{DROT}$ of the DROT functionality in Fig. 9 in Appendix.

Theorem 2. *Assuming a secure two-party computation of the OLE protocol, the DROT protocol (Section IV-B) securely implements the ideal functionality $\mathcal{F}_{DROT}$ (in Fig. 5) under the mixed-behaviour model.*

Theorem 3. *Let dkg, bit-decomp be secure MPC protocols, wm$(\cdot)$ is a robust bit watermarking algorithm, and DROT is secure (as in Theorem 2). The CDE protocol π_{CDE} securely realizes the ideal functionality $\mathcal{F}_{CDE}$ under the mixed-behaviour model.*

The theorem proofs are postponed to Appendix A.

We provide the implementation details with experimental results in Appendix B.

VII. Related Work

Timed-release encryption (TRE) was first introduced by May [73]. Several applications and approaches using TRE have been proposed including sealed bids [26] electronic voting systems [27], spam and denial of service preventions [41] and proof-of-work systems [53]. For time-lock puzzles, a well known puzzle was created by Rivest *et al.* in 1996 [82] 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 [19], signatures [43], [44] and key escrow [13], [14]. Besides the RSA-based construction of a time-lock puzzles recent results from Bitansky et al. show constructions based on random encodings [8], [15]. 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. [82] 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 [80] 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.* [37] which uses a trusted time-server and a newly created primitive called 'Conditional oblivious transfer'. They create an efficient protocol based on the quadratic residuosity assumption which in addition offers sender anonymity. Watanabe *et al.* used secret sharing where the dealer can choose a time. The shareholders can not reconstruct the secret shared before the time specified is over [90]. Many schemes based on identity based encryption were proposed, where the trusted key distribution center is also used for ensuring the correct time [18], [28], [29], [74]. 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 [65], [67], [42] present different audio watermarking schemes while works like [64], [92] deal with robust video watermarking. For software watermarking, schemes suggested in [88], [59] can be considered. The proposed IE protocol admits any robust watermarking scheme with no known attacks [83].

Halpern and Teague [50] introduce rational secret sharing where the agents sharing a secret are rational rather than honest or malicious. They show that at equilibrium, the agents do not contribute shares for reconstruction and propose a randomized protocol for performing the reconstruction. Gordon *et al.* [49] improve on the randomized protocol of [50] by overcoming the impossibility result. Lysyanskaya *et al.* [71] introduces the mixed behaviour model where the parties are either rational or adversarial, the authors provide a framework for multi-party computation in the proposed model. We adopt the mixed-behaviour model in this work.

Collusion in a multi-party setting to achieve collusion free MPC has been considered in works like [6], [7], [66], however, these works have stringent assumptions including the parties being co-located and communication only through a mediator. Recently, Ciampi *et al.* [30] define collusion proofness for multi-party functionalities, they assume availability of stateful trusted hardware tokens with each of the agents and parties communicate through authenticated broadcast channels. However, none of these works can be used for information escrows as agents can communicate over external channels and reconstruct the stored documents. In this work, we make no assumptions on communication or existence of envelopes and hardware tokens; we allow the agents to communicate freely on external channels and design the mechanism such that non-collusion is a Bayesian Nash Equilibrium for the system.

VIII. Conclusion and Future Work

We propose a novel Collusion Deterrent Escrow (CDE) mechanism to realize a distributed approach for information escrows that disincentivizes 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 disincentivizes any collusion among the agents offering the service to decrypt the user's data selectively. The agents make a conditional deposit on a cryptocurrency system before 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 until now. We analyze 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 not to collude. The cryptographic construction of CDE protocol employs robust watermarking, a 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. The proposed mechanism significantly raises the bar from state of the art by deterring collusion in the threshold escrow service.

While this work focuses on realizing secure information escrow using MPC, we find that our work can be extended to other MPC applications especially those using MPC-as-a-service for several users such as allegation escrow [9], [63], 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 Sri Aravinda Krishna Thyagarajan and Tian Tian Gong for helpful comments and feedback on the document. We also thank Simon Heinzel for his efforts with a preliminary manuscript associated with this 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] Chain link. https://chain.link/.

[2] Escrow tech. https://www.escrowtech.com/.

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

[4] Adelsbach, A., and Sadeghi, A.-R. Zero-knowledge watermark detection and proof of ownership. In *Information Hiding* (2001).

[5] Akinyele, J. A., Garman, C., Miers, I., Pagano, M. W., Rushanan, M., Green, M., and Rubin, A. D. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering* (2013), 111–128.

[6] Alwen, J., Katz, J., Lindell, Y., Persiano, G., shelat, a., and Visconti, I. Collusion-free multiparty computation in the mediated model. In *CRYPTO 2009* (2009), S. Halevi, Ed., pp. 524–540.

[7] Alwen, J., Shelat, A., and Visconti, I. Collusion-free protocols in the mediated model. In *CRYPTO 2008* (2008), D. Wagner, Ed., pp. 497–514.

[8] Applebaum, B. Randomized encoding of functions. In *Cryptography in Constant Parallel Time*, Information Security and Cryptography. 2014, pp. 19–31.

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

[10] Ayres, I., and Unkovic, C. Information escrows. *Mich. L. Rev. 111* (2012), 145.

[11] Barak, A., Hirt, M., Koskas, L., and Lindell, Y. An end-to-end system for large scale p2p mpc-as-a-service and low-bandwidth mpc for weak participants. In *ACM CCS* (2018), pp. 695–712.

[12] Beaver, D. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology – Crypto* (1991), pp. 420–432.

[13] Bellare, M., and Goldwasser, S. Encapsulated key escrow. Tech. rep., University of California, San Diego, 1996.

[14] Bellare, M., and Goldwasser, S. Verifiable partial key escrow. CCS '97, pp. 78–91.

[15] Bitansky, N., Goldwasser, S., Jain, A., Paneth, O., Vaikuntanathan, V., and Waters, B. Time-lock puzzles from randomized encodings. Cryptology ePrint Archive, Report 2015/514, 2015.

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

[17] BONEH, D., BOYEN, X., AND HALEVI, S. Chosen ciphertext secure public key threshold encryption without random oracles. In *CT-RSA 2006*, D. Pointcheval, Ed., vol. 3860 of *Lecture Notes in Computer Science*. 2006, pp. 226–243.

[18] BONEH, D., AND FRANKLIN, M. K. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology* (2001), CRYPTO '01, pp. 213–229.

[19] BONEH, D., AND NAOR, M. Timed commitments. In *CRYPTO 2000*, M. Bellare, Ed., vol. 1880 of *Lecture Notes in Computer Science*. 2000, pp. 236–254.

[20] CAMENISCH, J., AND STADLER, M. Proof systems for general statements about discrete logarithms. Tech. rep., Dept. of Computer Science, ETH Zurich, 1997.

[21] CANETTI, R. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science* (2001), pp. 136–145.

[22] CANETTI, R., GENNARO, R., JARECKI, S., KRAWCZYK, H., AND RABIN, T. Adaptive security for threshold cryptosystems. In *Annual International Cryptology Conference* (1999), Springer, pp. 98–116.

[23] CATRINA, O., AND DE HOOGH, S. Improved primitives for secure multiparty integer computation. In *International Conference on Security and Cryptography for Networks* (2010), Springer, pp. 182–199.

[24] CATRINA, O., AND SAXENA, A. Secure computation with fixed-point numbers. In *Financial Cryptography* (2010), vol. 6052.

[25] CAYRE, F., FONTAINE, C., AND FURON, T. Watermarking security: theory and practice. *IEEE Transactions on Signal Processing 53*, 10 (2005), 3976–3987.

[26] CHALKIAS, K., AND STEPHANIDES, G. Timed release cryptography from bilinear pairings using hash chains. In *Communications and Multimedia Security*, vol. 4237. 2006, pp. 130–140.

[27] CHEN, H.-C., AND DEVIANI, R. A secure e-voting system based on rsa time-lock puzzle mechanism. In *BWCCA* (2012), IEEE, pp. 596–601.

[28] CHEON, J. H., HOPPER, N., KIM, Y., AND OSIPKOV, I. Timed-release and key-insulated public key encryption. *IACR Cryptology ePrint Archive 2004* (2004), 15.

[29] CHEON, J. H., HOPPER, N., KIM, Y., AND OSIPKOV, I. Provably secure timed-release public key encryption. *ACM Trans. Inf. Syst. Secur. 11*, 2 (2008).

[30] CIAMPI, M., LU, Y., AND ZIKAS, V. Collusion-preserving computation without a mediator. IEEE CSF 2022 (to appear), 2022.

[31] DAMGÅRD, I., FITZI, M., KILTZ, E., NIELSEN, J. B., AND TOFT, T. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography* (2006), pp. 285–304.

[32] DAMGÅRD, I., PASTRO, V., SMART, N., AND ZAKARIAS, S. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology – CRYPTO 2012* (2012), pp. 643–662.

[33] DAMGÅRD, I., PASTRO, V., SMART, N. P., AND ZAKARIAS, S. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - CRYPTO 2012* (2012), pp. 643–662.

[34] DAS, S. DWT-DCT-Digital-Image-Watermarking. https://github.com/diptamath/DWT-DCT-Digital-Image-Watermarking.

[35] DEB, K., AL-SERAJ, M. S., HOQUE, M. M., AND SARKAR, M. I. H. Combined dwt-dct based digital image watermarking technique for copyright protection. In *2012 7th International Conference on Electrical and Computer Engineering* (2012), pp. 458–461.

[36] DESMEDT, Y. G., AND FRANKEL, Y. Threshold cryptosystems. CRYPTO '89, pp. 307–315.

[37] DI CRESCENZO, G., OSTROVSKY, R., AND RAJAGOPALAN, S. Conditional oblivious transfer and timed-release encryption. In *EUROCRYPT '99*, J. Stern, Ed., vol. 1592 of *Lecture Notes in Computer Science*. 1999, pp. 74–89.

[38] DOERNER, J., KONDI, Y., LEE, E., AND A. SHELAT. Secure two-party threshold ecdsa from ecdsa assumptions. In *2018 IEEE Symposium on Security and Privacy (SP)* (2018), pp. 595–612.

[39] DOTTLING, N., GHOSH, S., NIELSEN, J. B., NILGES, T., AND TRIFILETTI, R. Tinyole: Efficient actively secure two-party computation from oblivious linear function evaluation. In *2017 ACM CCS* (New York, NY, USA, 2017).

[40] DÖTTLING, N., KRASCHEWSKI, D., AND MÜLLER-QUADE, J. David & goliath oblivious affine function evaluation-asymptotically optimal building blocks for universally composable two-party computation from a single untrusted stateful tamper-proof hardware token.

[41] DWORK, C., AND NAOR, M. Pricing via processing or combatting junk mail. In *CRYPTO' 92*. 1993.

[42] ERFANI, Y., AND SIAHPOUSH, S. Robust audio watermarking using improved ts echo hiding. *Digital Signal Processing 19*, 5 (2009), 809 – 814.

[43] GARAY, J. A., AND JAKOBSSON, M. Timed release of standard digital signatures. In *FC'02: Financial cryptography* (2003), Springer-Verlag, pp. 168–182.

[44] GARAY, J. A., AND POMERANCE, C. Timed fair exchange of standard signatures. In *Financial Cryptography*, R. Wright, Ed. 2003.

[45] GENNARO, R., JARECKI, S., KRAWCZYK, H., AND RABIN, T. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptol. 20*, 1 (Jan. 2007), 51–83.

[46] GENNARO, R., RABIN, M. O., AND RABIN, T. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *ACM Symposium on Principles of Distributed Computing* (1998), PODC '98, pp. 101–111.

[47] GENNARO, R., RABIN, M. O., AND RABIN, T. 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* (1998).

[48] GHOSH, S., NIELSEN, J. B., AND NILGES, T. Maliciously secure oblivious linear function evaluation with constant overhead. In *Advances in Cryptology – ASIACRYPT 2017* (Cham, 2017), T. Takagi and T. Peyrin, Eds., Springer International Publishing, pp. 629–659.

[49] GORDON, S. D., AND KATZ, J. Rational secret sharing, revisited. Cryptology ePrint Archive, Report 2006/142, 2006.

[50] HALPERN, J. Y., AND TEAGUE, V. Rational secret sharing and multiparty computation: Extended abstract. *CoRR abs/cs/0609035* (2006).

[51] HARSANYI, J. C. Games with incomplete information played by "bayesian" players, i–iii part i. the basic model. *Management science 14*, 3 (1967), 159–182.

[52] HE, J., AND ZHANG, H. Digital right management model based on cryptography and digital watermarking. In *2008 International Conference on Computer Science and Software Engineering* (2008), vol. 3, pp. 656–660.

[53] JAKOBSSON, M., AND JUELS, A. Proofs of work and bread pudding protocols(extended abstract). In *Secure Information Networks*, B. Preneel, Ed., vol. 23 of *IFIP — The International Federation for Information Processing*. Springer US, 1999, pp. 258–272.

[54] JARECKI, S., AND LYSYANSKAYA, A. Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In *International Conference on the Theory and Applications of Cryptographic Techniques* (2000), Springer, pp. 221–242.

[55] KALKER, T. Considerations on watermarking security. In *2001 IEEE Fourth Workshop on Multimedia Signal Processing (Cat. No.01TH8564)* (2001), pp. 201–206.

[56] KATZ, J., MAURER, U., TACKMANN, B., AND ZIKAS, V. Universally composable synchronous computation. In *Theory of Cryptography Conference* (2013), Springer, pp. 477–498.

[57] KAWACHI, A., OKAMOTO, Y., TANAKA, K., AND YASUNAGA, K. General constructions of rational secret sharing with expected constant-round reconstruction. Cryptology ePrint Archive, Report 2013/874, 2013.

[58] KELLER, M., ORSINI, E., AND SCHOLL, P. Mascot: Faster malicious arithmetic secure computation with oblivious transfer. In *2016 ACM CCS* (New York, NY, USA, 2016), p. 830–842.

[59] KIM, S., AND WU, D. J. Watermarking cryptographic functionalities from standard lattice assumptions. In *Advances in Cryptology – CRYPTO 2017* (Cham, 2017), J. Katz and H. Shacham, Eds., Springer International Publishing, pp. 503–536.

[60] KOKORIS KOGIAS, E., MALKHI, D., AND SPIEGELMAN, A. *Asynchronous Distributed Key Generation for Computationally-Secure Randomness, Consensus, and Threshold Signatures.* 2020, p. 1751–1767.

[61] KREPS, D. M. *Nash Equilibrium*. 1989, pp. 167–177.

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

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

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

Fig. 8: Ideal Functionality of the OLE protocol [39], [48]

[62] KUHN, S. Prisoner's dilemma.

[63] KUYKENDALL, B., KRAWCZYK, H., AND RABIN, T. Cryptography for #metoo. *Proc. Priv. Enhancing Technol. 2019*, 3 (2019), 409–429.

[64] LANCINI, R., MAPELLI, F., AND TUBARO, S. A robust video watermarking technique in the spatial domain. In *International Symposium on VIPromCom Video/Image Processing and Multimedia Communications* (2002), pp. 251–256.

[65] LEI, B. Y., SOON, I. Y., AND LI, Z. Blind and robust audio watermarking scheme based on svd?dct. *Signal Processing 91*, 8 (2011), 1973 – 1984.

[66] LEPINSKI, M., MICALI, S., PEIKERT, C., AND SHELAT, A. Completely fair sfe and coalition-safe cheap talk. In *ACM Symposium on Principles of Distributed Computing* (2004), p. 1–10.

[67] LIE, W.-N., AND CHANG, L.-C. Robust and high-quality time-domain audio watermarking based on low-frequency amplitude modification. *IEEE Transactions on Multimedia 8*, 1 (Feb 2006), 46–59.

[68] LIU, F., AND WU, C.-K. Robust visual cryptography-based watermarking scheme for multiple cover images and multiple owners. *IET Information Security 5*, 2 (2011), 121–128.

[69] LU, D., YUREK, T., KULSHRESHTHA, S., GOVIND, R., KATE, A., AND MILLER, A. Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication. In *Proceedings of the 2019 ACM CCS* (2019), pp. 887–903.

[70] LUO, M., AND BORS, A. G. Surface-preserving robust watermarking of 3-d shapes. *IEEE Transactions on Image Processing 20*, 10 (2011), 2813–2826.

[71] LYSYANSKAYA, A., AND TRIANDOPOULOS, N. Rationality and adversarial behavior in multi-party computation. In *Annual International Cryptology Conference* (2006), Springer, pp. 180–197.

[72] MANGIPUDI, E. V., AND KATE, A. D-kode: Mechanism to generate and maintain a billion keys. Cryptology ePrint Archive, Report 2022/161, 2022. https://ia.cr/2022/161.

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

[74] MONT, M. C., HARRISON, K., AND SADLER, M. The hp time vault service: Innovating the way confidential information is disclosed, at the right time, 2002.

[75] MYERSON, R. B. *Game theory*. Harvard university press, 2013.

[76] NAMBA, R., AND SAKUMA, J. Robust watermarking of neural network with exponential weighting. In *2019 ACM Asia CCS* (2019), Asia CCS '19, Association for Computing Machinery, p. 228–240.

[77] NIELSEN, J. B., NORDHOLT, P. S., ORLANDI, C., AND BURRA, S. S. A new approach to practical active-secure two-party computation. Cryptology ePrint Archive, Report 2011/091, 2011. https://eprint.iacr.org/2011/091.

[78] NOORKAMI, M., AND MERSEREAU, R. M. A framework for robust watermarking of h.264-encoded video with controllable detection performance. *IEEE Transactions on Information Forensics and Security 2*, 1 (2007), 14–23.

[79] PEDERSEN, T. P. Non-interactive and information-theoretic secure verifiable secret sharing. CRYPTO '91.

[80] RABIN, M. O., AND THORPE, C. Time-lapse cryptography. Tech. rep., Harvard University School of Engineering and Applied Sciences, 2006.

[81] RAJAN, A., QIN, L., ARCHER, D. W., BONEH, D., LEPOINT, T., AND VARIA, M. Callisto: A cryptographic approach to detecting serial perpetrators of sexual misconduct. In *ACM COMPASS 2018* (2018), E. W. Zegura, Ed., pp. 49:1–49:4.

[82] RIVEST, R. L., SHAMIR, A., AND WAGNER, D. A. Time-lock puzzles and timed-release crypto. Tech. rep., Massachusetts Institute of Technology, 1996.

[83] ROGAWAY, P. Formalizing human ignorance. In *VIETCRYPT 2006* (2006), P. Q. Nguyen, Ed., pp. 211–228.

[84] SHAMIR, A. How to share a secret. *Commun. ACM 22*, 11 (Nov. 1979), 612–613.

[85] SINGH, P., AND CHADHA, R. S. A survey of digital watermarking techniques, applications and attacks. *International Journal of Engineering and Innovative Technology (IJEIT) 2*, 9 (2013), 165–175.

[86] TOFT, T. *CT-RSA 2009*. 2009, ch. Constant-Rounds, Almost-Linear Bit-Decomposition of Secret Shared Values, pp. 357–371.

[87] TOFT, T., ET AL. Primitives and applications for multi-party computation. *Doctoral dissertation, University of Aarhus, Denmark* (2007).

[88] VENKATESAN, R., VAZIRANI, V., AND SINHA, S. A graph theoretic approach to software watermarking. In *Information Hiding* (2001), I. S. Moskowitz, Ed., pp. 157–168.

[89] VEUGEN, T. Linear round bit-decomposition of secret-shared values. *Information Forensics and Security, IEEE Transactions on 10*, 3 (March 2015), 498–506.

[90] WATANABE, Y., AND SHIKATA, J. Timed-release secret sharing scheme with information theoretic security. *CoRR abs/1401.5895* (2014).

[91] WATANABE, Y., AND SHIKATA, J. Timed-release computational secret sharing and threshold encryption. *Designs, Codes and Cryptography 86*, 1 (2018), 17–54.

[92] ZHANG, J., HO, A. T. S., QIU, G., AND MARZILIANO, P. Robust video watermarking of h.264/avc. *IEEE Transactions on Circuits and Systems II: Express Briefs 54*, 2 (Feb 2007), 205–209.

APPENDIX A
SECURITY ANALYSIS

Theorem 2. *Assuming a secure two-party computation of the OLE protocol, the DROT protocol (Section IV-B) securely implements the ideal functionality $\mathcal{F}_{DROT}$ (in Fig. 5) under the mixed-behaviour model.*

Proof: The simulator $\mathcal{S}_{\mathsf{DROT}}$ in Fig. 9 interacts with the user U and n agents $\mathsf{A}_j, j \in [1, n]$, the adversary $\mathcal{A}$ corrupts a maximum of t parties. The simulator presents an indistinguishable view to the adversary in the real-world ideal-world paradigm. In the DROT protocol, the user inputs pair of of keys $\mathsf{k}_0, \mathsf{k}_1$ where as the agents input shares $[s]$ of a secret key bit s which is $(n, t+1)$ threshold shared among the agents.

The transfer of secret key shares from user to the agents is realized using the secure 2-party computation of oblivious linear function evaluation (OLE) between the user and each of the agents. The simulator $\mathcal{S}_{\mathsf{DROT}}$ invokes $\mathcal{S}_{OLE}$ [48] with corresponding inputs while generating the view for the adversary. The simulator for the OLE protocol $\mathcal{S}_{OLE}$ provided by Ghosh *et al.* [48] is a generalization where the inputs are t element vectors. This can be used in a straightforward manner for $t = 1$. $\mathcal{S}_{DROT}$ uses $\mathcal{S}_{OLE}$ in a black box manner while

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

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

Corrupt Sender

The simulator simulates n agents to the sender.

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

Corrupt Receivers:

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

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

Corrupt Sender and $t-1$ corrupt receivers

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

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

Fig. 9: DROT simulator

interacting with corrupt parties, invoking the corresponding side (corrupt sender or corrupt receiver) of it.

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

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

When the sender and $t-1$ receivers are corrupted, $\mathcal{S}_{DROT}$ simulates $n-t+1$ receivers to the sender and forwards all the messages between the corrupted sender and the receivers without any modifications. $\mathcal{S}_{DROT}$ invokes $n-t+1$ instances of $\mathcal{S}_{OLE}$ which simulate the receivers. $\mathcal{S}_{OLE}$ extracts the sender inputs $\mathsf{k}_0, \mathsf{k}_1$ which are forwarded to $\mathcal{F}_{DROT}$ using the message $(\texttt{inputS}, sid, \mathsf{k}_0, \mathsf{k}_1)$.

The indistinguishability of the view follows from the UC-security of $\mathcal{F}_{OLE}$. In the case when the sender and $t-1$ receivers are corrupted, all the messages are forwarded between them without modification.

From Theorem 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 V-C and Table III. 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 3. *Let dkg, bit-decomp be secure MPC protocols, wm$(\cdot)$ is a robust bit watermarking algorithm, and DROT is secure (as in Theorem 2). The CDE protocol π_{CDE} securely realizes the ideal functionality $\mathcal{F}_{CDE}$ under the mixed-behaviour model.*

Proof Sketch: The system consists of user U and n agents $\mathsf{A}_j, j \in [1,n]$; at any instance of time a maximum of t parties can be corrupted by the adversary $\mathcal{A}$. The CDE protocol involves distributed key generation, bit-decomposition and DROT protocols, where DKG and bit-decomposition are performed among the agents and DROT protocol is run between the user and the agents. We use these three protocols in a black-box manner with their corresponding simulators. We refer the reader to the works [45] and [31], [87] for the simulators for DKG and bit-decomposition algorithms. We refer to them as $\mathcal{S}_{DKG}$ and $\mathcal{S}_{\mathsf{BitDec}}$.

The simulator $\mathcal{S}_{CDE}$ generates an indistinguishable view for the adversary in the real-world ideal-world paradigm. It invokes the three simulators $\mathcal{S}_{DKG}$, $\mathcal{S}_{\mathsf{BitDec}}$ and $\mathcal{S}_{\mathsf{DROT}}$ for each of the phases of the protocol run. The DKG protocol [45] 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 $\mathcal{S}_{DKG}$, $\mathcal{S}_{\mathsf{BitDec}}$ 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 query values q_i. These queries are used by the user to sample keys to encrypt the different message

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

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

blocks. The simulator $\mathcal{S}_{CDE}$ invokes λ instances of $\mathcal{S}_{\mathsf{DROT}}$ once for each of the bits of the secret key. In cases with corrupted sender, the simulator computes the keys used for encryption k_i by checking over all the stored queries q_i and inputs those keys.

Since the rational parties have an incentive to obtain the correct key shares for the protocol to proceed, they have an incentive to not deviate from the protocol during the setup. After obtaining the watermarked and encrypted user data and the corresponding key shares through DROT, the agents do not collude at equilibrium as was proved by From Theorem 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. ■

APPENDIX B
IMPLEMENTATION

We implement the CDE protocol using HoneybadgerMPC [69], SCALE-MAMBA [33], and Charm cryptographic library [5]. Our implementation includes realizing the DROT protocol to transfer encrypted watermarked images and their corresponding key shares. Each run of the protocol involves splitting the data into blocks, watermarking the blocks, and transferring them. The DKG and bit-decomposition protocols are run by the agents as setup before the user enters the system.

Distributed Key Generation—DKG. The DKG protocol is realized using HoneybadgerMPC [69], a secret sharing based python MPC framework supporting malicious security. HoneybadgerMPC supports the robust reconstruction of secret shared values and requires the adversary to control up to $t < n/3$ parties. Honeybadger using a partial synchronous network model requires $n \geq 3t + 1$, however, assuming bounded synchrony this reduces to $n \geq 2t + 1$. All test cases we run on honeybadgerMPC satisfy this threshold. We implement the DKG protocol proposed by Gennaro *et al.* [45], we realize Pedersen verifiable secret sharing (VSS) leveraging the communication layer of HoneybadgerMPC. The DKG is realized by letting each agent perform VSS of random values, sum them up to get the shared private key, and compute the public key from the commitments obtained. The agents generate shares of the $256-$bit secret key with the key pair on the curve secp256k1. The DKG protocol has a message complexity of $O(n^3)$ for n parties. When adversary can corrupt t agents, the threshold of secret sharing is $t+1$ (at-least $t+2$ agents are needed for reconstruction), requiring $n \geq 3t+1$ for safety and liveness of the DKG protocol. Refer Mangipudi *et al.* [72] for benchmarks on DKG protocol timings for higher number of nodes.

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.* [24]. 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 [87]. Moreover, we use the prefix multiplication protocol introduced in [23] to replace the prefix AND sub-protocol in [87]. 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 IV-B) and we implement it using SCALE-MAMBA [33] 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 [33] is leveraged in the online phase. The whole two-party computation involves one multiplication and one reconstruction. The computation is performed by the user with each agent so the total communication complexity is $O(n)$ for n agents.

Watermarking. The user splits the data into blocks and generates two versions of each block by watermarking with bits $0, 1$. For a 256-bit secret key of the agents, the user divides his data into 256 blocks. While the data can be of any form including multi-media and document data, we use bit map images for the prototype. For image watermarking we use the combined DWT-DCT watermarking algorithm proposed by Ali Al-Haj [35] and the implementation based on [34]. 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.

A. Experimental results

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

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

TABLE V: 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

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 V 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.

APPENDIX C
PROVIDING DETECTION KEY TO A SUBSET OF AGENTS

After transferring the document data to the agents using the cryptographic protocol, the user transfers auxiliary information to the agents. The information provided to the agents induces a typeset among the agents. With the typeset induced among the agents, the two collusion scenarios of threshold decryption and secret key reconstruction should be analyzed separately because in secret key reconstruction attack, the auxiliary information is not used. The collusion games arising from the two attack scenarios are named Collusion-1 and Collusion-2.

The auxiliary information provided to the agents is modelled as belonging to the information types $\{\mathcal{I}, \mathcal{D}\}$. 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 user forwards the information $\mathcal{I}$ (resp. $\mathcal{D}$) with probability p (resp. $1-p$) to each of the agents.

In the implementation of the protocol, the information $\mathcal{I}$ would correspond to the correct detection key and $\mathcal{D}$, an incorrect detection key. $D > 0$ is the value of conditional deposit made by each agent and F is the approximate sum of future payments to be received if the agent/node is not banned from the system. The two collusion scenarios Collusion-1, Collusion-2 would be collusion scenarios using document decryption and secret key (sk) reconstruction respectively. In the 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 $F \gg 0$.

A. Collusion-game-1

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

- The set of agents $N = \{\mathsf{A}_1, \mathsf{A}_2\}$
- The typeset of the agents $\Theta = \{e, ie\}$
- The strategy space of each agent j is $S_j = \{wait_1, transfer_1\}$
- The utilities corresponding to the strategy and the type of the agent $u_j(\theta_j, s_j, s_{-j}), \theta_j \in \Theta, s_j \in S_j$. s_{-j} indicates the strategy(ies) of player(s) other than player j.

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

Payoffs. The expected pay-offs of the agents under Collusion-1 is captured by the pay-off matrices in Table VI. An efficient agent (e) has the correct detection key and inefficient agent (ie) does not. When both agents are efficient and decrypt the user document and not transfer the deposit (play $wait_1$), they both can accrue a pay-off of $\alpha = v + \frac{d}{2}$. If one of the agents transfers (plays $transfer_1$) the deposit, he gets a pay-off of $\alpha + D$ where as, the other agent loses his deposit and gets banned thereby accruing a pay-off of $\alpha - (F + D)$. In the case when both the efficient agents attempt to transfer the deposit simultaneously, since only one agent can succeed in the transfer, it creates a race condition and we assume that each will succeed in the transfer with equal probability [5]; hence in expectation they accrue a pay-off $0.5(\alpha+D)+0.5(\alpha-F-D) = \alpha-\frac{F}{2}$. $u_j(e, transfer_1, s_{-j}) > u_j(e, wait_1, s_{-j}) \forall s_{-j} \in \{transfer_1, wait_1\}$ (from Table VI). For an efficient agent $transfer_1$ is a strictly dominant strategy, he always plays $transfer_1$. For the inefficient agent, $wait_1$is the only strategy available to play in Collusion-1, trivially, it is the dominant strategy. The dominant strategies have been indicated in grey in Table VI, the pay-off submatrix of Table VI when both the agents are efficient are similar to the pay-offs in well known prisoners' dilemma [62]. The agents playing their dominant strategy for the given type according to the payoff matrix of Table VI indeed forms the Bayesian Nash Equilibrium as defined below.

Definition 2 (Bayesian Nash Equilibrium [51])**.** *A strategy profile $s(\cdot)$ is a Bayesian Nash equilibrium if $Eu_j(s_j|s_{-j}, \theta_j) \geq Eu_j(s'_j|s_{-j}, \theta_j)$ for all $\theta_j \in \Theta_j$, for all $s'_j(\theta_j) \in S_j$ where the expected utility $Eu_j(s_j|s_{-j}, \theta_j) = \sum_{\theta_{-j} \in \Theta_{-j}} u_j(s_j, s_{-j}(\theta_{-j}), \theta_j, \theta_{-j}) p(\theta_{-j}|\theta_j)$.*

Expected payoff. The regulator chooses each agent and transfers information $\mathcal{I}$ with probability p independently. Hence the

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

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

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

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

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

expected payoff of each agent playing their dominant strategies (Table VI) is:

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

B. Collusion-game-2

In the collusion scenario where the agents reconstruct the secret key (Collusion-2), the game induced is Collusion-game-2. The agents can either choose to transfer the deposit – play $transfer_2$ or wait without transferring – play $wait_2$. The actions do not depend on the availability of the detection key and hence do not depend on the type of the agent. The following parameters define the game.

- The set of agents $N = \{A_1, A_2\}$
- The strategy space of each agent j is $S_j = \{wait_2, transfer_2\}$
- The utilities corresponding to the strategy.

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

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

Maximum pay-off from collusion. It can be seen from Eq. (1), Eq. (2) the maximum expected pay-off that can be obtained by each agent through either collusion scenarios is

$$\beta = \max(\hat{a}, \hat{b}) = \max\Big((\alpha - F(p - \frac{p^2}{2})), (\alpha - \frac{F}{2})\Big) \tag{3}$$

As F is the sum of all future payments from many users, we have, $F \gg D, F \gg d, v$. The regulator or the user sets the value of p such that $\beta < v$.

The value of p is lower bounded such that any subset of $t + 1$ agents has at least one agent with the detection key. However, since the pay-off from collusion Collusion-1 is minimum when $p = 1$, the user provides the detection key to all the agents. This further results in all the agents being efficient agents and simplifies the game theoretic analysis for the Collusion-1 game. With all the agents being efficient, the pay-off matrix of Collusion-1 is similar to Collusion-2. We present the resulting pay-off matrix as Table II along with the corresponding analysis in Section V.