Cassiopeia: Practical On-Chain Witness Encryption

Schwinn Saereesitthipitak and Dionysis Zindros

Stanford University

Abstract. Witness Encryption is a holy grail of cryptography that remains elusive. It asks that a secret is only revealed when a particular computational problem is solved. Modern smart contracts and blockchains make assumptions of "honest majority", which allow for a *social* implementation of Witness Encryption. The core idea is to make use of a partially trusted committee to carry out the responsibilities mandated by these functionalities – such as keeping the secret private, and then releasing it publicly after a solution to the computational puzzle is presented. We propose Cassiopeia, a smart contract Witness Encryption scheme (with public witness security) and provide an open source composable implementation that can be utilized as an oracle by others within the broader DeFi ecosystem. We devise a cryptoeconomic scheme to incentivize honest participation, and analyze its security under the honest majority and rational majority settings. We conclude by measuring and optimizing gas costs and illustrating the practicality of our scheme.

Keywords: Witness Encryption · Publicly Verifiable Secret Sharing · Smart Contract.

1 Introduction

Witness Encryption [16] is a cryptographic scheme where a message is encrypted such that it can only be decrypted when a solution (also called the "witness") to a computational puzzle is presented. For example, a message may be witness-encrypted such that it can only be decrypted if for some connected graph, a Hamiltonian cycle is found. A particularly useful instantiation of Witness Encryption in the blockchain setting is Timelock Encryption [25, 27, 29], where a message is encrypted such that it can only be decrypted after a set unlock time. This enables important use cases in DeFi such as sealed bid auctions and front-running prevention [1].

Though Witness Encryption is still impractical under standard assumptions, the "honest majority" assumption in modern blockchains can be leveraged to make Witness Encryption practical. We do so by asking a committee to keep the secret and release it for us when the witness is presented. As long as a majority of the committee is honest, the secret is revealed at the right time. Our goal is to make Witness Encryption practical and composable for use in DeFi. Furthermore, we hold committee members accountable for releasing correct decryption keys in a timely manner.

Our Contributions. We summarize our contributions as follows:

1. We put forth a smart contract-based construction for Witness Encryption.
2. We explore whether our scheme is correct and secure under the honest majority and rational majority settings.
3. We implement the construction as an open-source Solidity smart contract, benchmark gas usage on Ethereum, and provide our parameterization of the scheme.

Construction Overview. Suppose Alice wants to encrypt a secret with an instance of a puzzle. Alice will entrust a fixed committee to keep her secret and release it if someone presents a solution to the puzzle to the smart contract. She trusts that the majority of the committee are honest. Alice splits her secret into shares, one for each committee member, such that a majority of them are required to reconstruct the original secret and each committee member is only able to decrypt its own share. Alice then calls a smart contract function to publish the encrypted shares for them to access. She also attaches a zero-knowledge proof of knowledge of the secret (in the form of a zkSNARK) to ensure she is honest and actually knows the secret. The contract verifies that Alice split her secret into shares correctly. When the contract receives a solution to the puzzle, it enables committee members to submit their decrypted shares to the contract. When a committee member submits a share, the contract verifies that it is consistent with the corresponding encrypted share Alice posted before accepting the share. As long as at least a majority of the committee members revealed their shares, Alice's secret can be reconstructed by anyone.

We also propose an incentivized version of the above scheme which includes fees and slashing. Each committee member deposits collateral into the contract. If a committee member correctly reveals its decrypted shares, it is awarded a fee. Otherwise, its collateral is slashed. We ensure the fee is sufficient to incentivize honest committee member participation, using the risk-free rate as a benchmark for the opportunity cost. Note that the incentivized version of the scheme requires Alice to provide a maximum secret lifespan upfront after which the secret can be decrypted even without a witness.

Related Work. General-purpose Witness Encryption was introduced by Garg et al. [16]. The standard security definition of Witness Encryption is *extractable security* [18]. Most current Witness Encryption schemes are based on multilinear maps [16, 17] or on indistinguishability obfuscation (iO) [15]. Both construction paths require strong assumptions and are computationally impractical. One alternative is to use weaker variants of Witness Encryption that encompass only a specific subset of NP [4].

Another alternative to build practical Witness Encryption, which we explore in this work, is to do *social witness encryption* where a committee is entrusted up to an adversarial threshold. Goyal et al. [19] were the first to propose an extractable social Witness Encryption scheme constructed from Proactive Secret Sharing (PSS) that leverages a blockchain and its Proof-of-Stake validators as

the committee. Though there have been multiple previous works proposing social WE equivalents [8, 12, 19], we are the first to provide an incentivized smart contract instantiation. One particularly useful application of Witness Encryption and blockchains is Timelock Encryption [25, 27, 29]. Social Timelock Encryption leveraging Identity-Based Encryption (IBE) [6] has been put forth in i-TiRE [1] and tlock [24]. Dottling et al. [12] proposed a social Timelock Encryption scheme using a weaker Signature-based Witness Encryption (SWE) scheme.

The critical building block for social Timelock Encryption and Witness Encryption is a Secret Sharing Scheme [32]. In the blockchain setting, threshold encryption was previously explored in works by Benhamouda et al. [3] and Goyal et al. [19]. For accountability, we use Publicly Verifiable Secret Sharing (PVSS) [10, 33]. For our implementation, we use SCRAPE [9] as the underlying PVSS scheme.

Applications. Witness Encryption can be used to build powerful cryptographic primitives such as succinct Functional Encryption [7,28] for Turing machines [18]. When combined with a blockchain, Witness Encryption can also be leveraged to achieve fairness against dishonest majority in a secure multi-party computation protocol [11]. More practically, it can also be used to instantiate Timelock Encryption, which can be applied towards building new forms of wallets [34] among other applications.

2 Preliminaries

2.1 Symmetric Encryption

Let s be the secret key and m be the message to encrypt. A symmetric encryption scheme contains an encryption algorithm $Enc_s(m)$ that produces a ciphertext c and a decryption algorithm $Dec_s(c)$ that produces the original message m.

2.2 Secret Sharing

A threshold secret sharing scheme [32] consists of a dealer D, a secret s, a committee of n participants $p_1, p_2, \ldots, p_n$, and a threshold t. To share the secret, D calls a function $\mathsf{share}(s)$ which splits s into shares $s_1, s_2, \ldots, s_n$ and distributes these to participants such that each participant p_i only has access to s_i. A set of at least t participants can work together to combine their shares $s_{i_1}, s_{i_2}, \ldots$, using the function $\mathsf{recover}(s_{i_1}, s_{i_2}, \ldots)$ to obtain the original secret. On the other hand, if less than t participants work together to combine their shares, no information can be learned about the secret.

2.3 Publicly Verifiable Secret Sharing (PVSS)

Publicly verifiable secret sharing schemes [9, 10, 23, 30, 31, 33] are augmentations of secret sharing schemes such that any verifier V (who does not need to be the dealer or part of the committee) can verify that the secret shares are valid. PVSS schemes generally follow the steps below [9]:

– **Setup**: Given a security parameter λ, each participant p_i calls $\mathsf{Gen}(1^\lambda)$ to generate a public-private key pair (pk_i, sk_i) and shares pk_i publicly.
– **Distribution**: Let s be the secret to be shared by the dealer. The dealer calls $\mathsf{genDist}(s, [pk_i])$ to generate shares $s_1, \ldots, s_n$ and encrypts each share s_i with pk_i to produce $\hat{s}_i$. The dealer also generates a proof π_D that all $\hat{s}_i$ are valid and consistent with one another.
– **Verification**: Any external verifier V calls $\mathsf{verifyDist}([\hat{s}_i], [pk_i], \pi_D)$ to non-interactively verify, using all publicly available information, that the encrypted shares are consistent with one another.
– **Reconstruction**:
  • Each p_i publishes $s'_i = \mathsf{decrypt}(\hat{s}_i, sk_i)$, along with a proof π_i that $s'_i = s_i$.
  • V calls $\mathsf{verifyShare}(i, \hat{s}_i, s'_i, \pi_i)$ for every participant's contribution to verify that all $s'_i = s_i$.
  • With all shares known, anybody can call $\mathsf{reconstruct}(s'_{i_1}, \ldots, s'_{i_t})$ on the revealed shares to obtain the secret s.

PVSS schemes generally have the following properties [23]:

– **Correctness**: If the dealer is honest and a set of at least t parties are honest, then verifyDist passes during distribution, verifyShare passes during reconstruction for all honest parties, and reconstruct yields the original secret.
– **Verifiability**: If verifyDist passes, then $\hat{s}_i$ are consistent secret shares with overwhelming probability, and if verifyShare passes, then $\hat{s}'_i = s_i$ with overwhelming probability.
– **Secrecy**: Prior to reconstruction, any set of less than t participants cannot learn any information about the secret.

2.4 Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zkSNARK)

Let C be an efficiently computable boolean circuit. We define the relation $\mathcal{R}_C = \{(x_C, w_C) \mid C(x_C, w_C) = 1\}$, where (x_C, w_C) is an instance-witness pair. A zk-SNARK consists of a triple of probabilistic polynomial time algorithms (G, P, V) as follows [5, 21]:

– G is a generator that, upon receiving a security parameter input λ, generates a *reference string* σ and a *verification state* τ.
– A prover $P(\sigma, x_C, w_C)$ that outputs a proof π for the instance x_C and the witness w_C.
– A verifier $V(\tau, x_C, \pi)$ that outputs 1 if the proof is valid and 0 otherwise.

zkSNARKs have the following properties:

– **Correctness**: An honestly-generated proof π is always accepted by an honest verifier V.
– **Soundness**: A proof π generated with an invalid witness will not be accepted by an honest verifier V with overwhelming probability.
– **Zero Knowledge**: The interaction between prover and verifier reveals negligible information about the witness.

2.5 Witness Encryption

In a Witness Encryption scheme, a message is encrypted such that the ciphertext can only be decrypted if a witness is presented as a solution to an NP problem [16]. For example, a message can be encrypted such that it can only be decrypted with a solution to an instance of the 3-SAT problem. Concretely, let $\mathcal{R}$ be a relation of an NP language $\mathcal{L}$ such that for each $x \in \mathcal{L}$, there exists some w such that $(x, w) \in \mathcal{R}$ and for all $x \notin \mathcal{L}$ such a witness does not exist. The scheme has public witness security if the witness is public during decryption, and the message is revealed publicly. A Witness Encryption scheme, parameterized by a security parameter λ, consists of PPT encryption and decryption functions $\mathsf{WE.Enc}_{\mathcal{R}}(1^{\lambda}, x, m)$ and $\mathsf{WE.Dec}_{\mathcal{R}}(c, x, w)$ such that the following properties hold [25]:

- *Correctness*: For any plaintext message m, instance $x \in \mathcal{L}$, and witness w such that $(x, w) \in \mathcal{R}$, $\mathsf{WE.Dec}_{\mathcal{R}}(\mathsf{WE.Enc}_{\mathcal{R}}(1^{\lambda}, x, m), x, w) = m$.
- *Extractable Security*: A PPT adversary given $c = \mathsf{WE.Enc}_{\mathcal{R}}(1^{\lambda}, x, m)$ is only able to extract information about m if she can also produce a witness w such that $(x, w) \in \mathcal{R}$, except with negligible probability.

2.6 Risk-Free Rate

The fees awarded to committee members should be at least the return on an exogenous risk-free source of yield, otherwise rational committee members will stop participating in the protocol. Therefore, we assume there exists a per-block risk-free rate r agreed upon by all parties participating in the protocol. At any block, anyone can deposit M and earn $M(1+r)$ after one block via an exogenous risk-free source of yield. The opportunity cost for not earning risk-free yield on M for k blocks is $M(1+r)^k - M$.

3 Construction

The Cassiopeia smart contract is instantiated for a fixed committee with known public keys of size n and a threshold t. The threshold t is a public parameter indicating the minimum number of honest committee members required for the construction to be correct and secure. Correctness means that the secret can be decrypted if the witness becomes available. Security means no information about the secret can be obtained unless a witness becomes available.

Suppose the dealer wants to perform *social* Witness Encryption with public witness security on a message m (Algorithm 1). First, the dealer chooses a relation $\mathcal{R}$ and a corresponding instance x. Then, the dealer generates a random bit string s that can be simultaneously used as the secret in a PVSS scheme and the key of a symmetric encryption scheme. The dealer runs $\mathsf{PVSS.genDist}(s, [pk_i])$ to generate the encrypted secret shares $[\hat{s}_i]$ along with a proof π_D that the generated encrypted secret shares are consistent with one another. Note that a valid proof π_D guarantees that $\mathsf{Cassiopeia.encrypt}$ prevents committee members

from maintaining shares of invalid secrets. It becomes vital to correctness in the incentivized construction in Section 4.

Denote $c = ([\hat{s}_i], \pi_D)$ as the *PVSS ciphertext*. Using a symmetric encryption scheme, the dealer encrypts m with the key s to produce $\hat{c} = Enc_s(m)$. The dealer calls the smart contract function encrypt to register the PVSS ciphertext and instance on chain. The contract checks whether the proof π_D is valid, and if so, makes the encrypted secret shares available to the committee members. A unique identifier for the secret id is returned. Subsequently, $\hat{c}$ is dispersed to *public storage* (using the function disperse), either off-chain to optimize gas costs (e.g. IPFS) or on-chain for data availability.

Algorithm 1 Off-chain procedure run by dealer to witness-encrypt a secret

function $\mathsf{WE.Enc}_{\mathcal{R}}(1^{\lambda}, x, m)$
  $s \xleftarrow{\$} \{0,1\}^{\lambda}$
  $\hat{c} \leftarrow Enc_s(m)$
  $c \leftarrow \mathsf{PVSS.genDist}(s, [pk_i])$
  $y \leftarrow H(s \,\|\, c \,\|\, \mathcal{R} \,\|\, x)$
  $\pi \leftarrow P(\sigma, (c, \mathcal{R}, x, y, [pk_i]), s)$
  $id \leftarrow \mathsf{Cassiopeia.encrypt}(c, \mathcal{R}, x, y, \pi)$
  $\mathsf{disperse}(id, \hat{c})$
end function

Anybody who obtains a valid witness w can call the smart contract function claim to start decryption (Algorithm 2). The smart contract checks that w is indeed a valid witness such that $(x, w) \in \mathcal{R}$. If so, a flag M_{id} is set indicating the secret has been claimed. Note that to support private decryption, one could provide a zkSNARK proof as the witness w for a boolean circuit corresponding to x and $\mathcal{R}$ [19]. This way, w reveals nothing about the actual secret witness while still retaining the same correctness and security properties.

Now, committee members will decrypt their encrypted shares $\hat{s}_i$ and submit the result on chain (Algorithm 3). A committee member does so by first using PVSS.decrypt to obtain a decryption s'_i and a proof π_i that s'_i is a valid decryption, i.e. $s'_i = s_i$. The committee member then submits the share on-chain by calling the smart contract function submitShare, which verifies the proof and stores s'_i in the set of decrypted shares S_{id} inside the contract. Once $|S_{id}| \geq t$, anyone can reconstruct the secret s using PVSS.reconstruct. To obtain the original message, $\hat{c}$ is fetched from *public storage* (using the function fetch) and decrypted using s as the key to produce $m = Dec_s(\hat{c})$. Note that for our correctness proof to follow, we assume committee members submit their shares within Δ blocks of the secret being claimed, where Δ is a fixed parameter. The full decryption procedure is written in pseudocode below.

However, this protocol still vulnerable to malleability attacks. Concretely, let x' be an instance of relation $\mathcal{R}'$ for which the adversary already knows a valid witness w'. The adversary can act as a malicious dealer, calling encrypt with c

Algorithm 2 Off-chain decryption procedure run by anyone in possession of the witness w

1: $id \leftarrow$ identifier of secret to decrypt
2: **function** $\mathsf{WE.Dec}_{\mathcal{R}}(1^{\lambda}, c, x, w)$
3: $\quad \mathsf{Cassiopeia.claim}(id, w)$
4: $\quad \triangleright$ *Wait for committee members to submit shares*
5: $\quad$ **upon** $|\mathsf{Cassiopeia}.S_{id}| \geq t$ **do** $\quad \triangleright$ Monitor smart contract for state change
6: $\quad\quad s \leftarrow \mathsf{reconstruct}(\mathsf{Cassiopeia}.S_{id})$
7: $\quad\quad \mathsf{fetch}(\hat{c})$
8: $\quad\quad m \leftarrow Dec_s(\hat{c})$
9: $\quad$ **end upon**
10: **end function**

Algorithm 3 Off-chain procedure run by any committee member submitting shares

1: $i \leftarrow$ index of own public key in $[pk_i]$
2: **upon** $\mathsf{Cassiopeia}.M_{id} = \textsc{claimed}$ **do** $\quad \triangleright$ Monitor smart contract for state change
3: $\quad ([\hat{s}_i], \pi_D) \leftarrow \mathsf{Cassiopeia}.C_{id}.c$
4: $\quad (s'_i, \pi_i) \leftarrow \mathsf{PVSS.decrypt}(\hat{s}_i, sk_i)$
5: $\quad \mathsf{Cassiopeia.submitShare}(s'_i, \pi_i, id, i)$
6: **end upon**

and x' instead of x, then call claim with w' to notify committee members to start submitting their shares (and thereby start decryption). Therefore, the adversary can bypass the requirement of finding a valid witness w for x to decrypt the secret encoded by c.

To mitigate this issue, we ask the dealer to provide a proof in zero knowledge that he knows s and he intends to encrypt s with the instance x. In particular, the dealer generates a commitment to the secret and ciphertext, tying it to $\mathcal{R}$ and x by computing $y = H(s \,\|\, c \,\|\, \mathcal{R} \,\|\, x)$, where H is a hash function modeled as a random oracle. The dealer then generates a zkSNARK proof of knowledge of s such that if the encrypted shares were decrypted and recombined, the result would be s, and that $y = H(s \,\|\, c \,\|\, \mathcal{R} \,\|\, x)$. More formally, the dealer generates the proof $\pi = P(\sigma, (c, \mathcal{R}, x, y, [pk_i]), s)$ in Line 6 of Algorithm 1 using the boolean circuit in Algorithm 4. Without knowledge of s, an adversarial dealer cannot use the same ciphertext c with another instance x'.

Algorithm 4 zkSNARK circuit defined by $\mathcal{R}_C$

Require: $x_C = (c, \mathcal{R}, x, y, [pk_i])$, $w_C = s$
1: $y' \leftarrow H(s \,\|\, c \,\|\, \mathcal{R}, x)$
2: $c' \leftarrow \mathsf{PVSS.genDist}(s, [pk_i])$
3: **return** $y' = y \wedge c' = c$

When the dealer calls encrypt, he must include y and π. The contract verifies that the zero knowledge proof π is valid with respect to c, $\mathcal{R}$ and y, on top of already verifying the PVSS ciphertext as outlined above. Concretely, the π must be valid according to $V(\tau, (c, \mathcal{R}, x, y, [pk_i]), \pi) = 1$. Without a valid π, an adversary would not be able to carry out a malleability attack. The Cassiopeia smart contract is written in pseudocode in Algorithm 5.

Algorithm 5 Cassiopeia Smart Contract

```
1: contract Cassiopeia
2:     C, S, M ← ∅
3:     function encrypt(c, R, x, y, π)
4:         ([ŝ_i], π_D) ← c
5:         require(V(τ, (c, R, x, y, [pk_i]), π) = 1 ∧ PVSS.verifyDist([ŝ_i], [pk_i], π_D))
6:         id ← H(c, R, x)
7:         C_id ← (c, R, x)
8:         return id
9:     end function
10:    function claim(id, w)
11:        {R, x, ...} ← C_id
12:        require((x, w) ∈ R)
13:        M_id ← CLAIMED
14:    end function
15:    function submitShare(s'_i, π_i, id, i)
16:        require(M_id = CLAIMED)
17:        ([ŝ_i], ⊥) ← C_id.c
18:        PVSS.verifyShare(i, ŝ_i, s'_i, π_i)
19:        S_id,i ← s
20:    end function
21: end contract
```

4 Incentives

Here, we augment the scheme in Section 3 to incentivize committee members to act honestly. We will create incentives for the committee to reveal on time so that the secret is recoverable. This will be done by paying out a reward to the committee members who reveal at the right time, while slashing committee members who do not. If every committee member is honest, everyone will be rewarded. This way, we will ensure correctness. Unfortunately, we cannot use slashing to ensure security, as malicious committee members can always reveal confidential information off-chain and the smart contract has no way of knowing this, so our goal will only be correctness.

Initially, the dealer chooses a *reparation price*, which they are guaranteed to be paid in case the secret is irrecoverable. Next, the dealer calculates a *holding fee* which is a function of the reparation price. The larger the reparation price,

the larger the holding fee must be. He begins the Witness Encryption procedure by paying the holding fee into the contract. The holding fee is the incentive for committee members to participate honestly in the protocol. At the same time, each committee member puts in a certain *collateral*, which is held by the contract in escrow until the completion of protocol. If the majority of committee members are dishonest and the secret is irrecoverable, the slashing amounts are sufficient to add up to the reparation price which is used to appease the dealer in the case of failure.

Consider the happy path, where the dealer and all committee members are honest. After a call to claim with a valid witness, committee members submit their shares. As soon as a committee member submits a valid share, they receive their reward. Let f be the holding fee of the secret. The holding fee is split equally amongst all committee members to cover their reward payments, so each committee member's reward is $\frac{f}{n}$.

Now consider the scenario where the committee has at least t honest members, but not all of them are honest. Dishonest committee members may choose to not submit their shares. If any committee member does not submit their share, they do not receive their reward of $\frac{f}{n}$. Instead, it is transferred back to the dealer.

Now consider the case where the secret is irrecoverable. In particular, there are less than t honest committee members who submit their shares. Every dishonest committee member has their collateral slashed equally, on top of already not receiving their reward. Note that we mentioned in Section 3 that the proof π_D is needed to ensure correctness incentives in this scheme. This is because if we did not check that the PVSS output was consistent, the adversary could generate invalid secret shares, yet the committee members would be slashed.

Suppose only t' committee members reveal valid shares and the reparation price is a. The sum of every dishonest committee member's slashed collateral and forfeited reward must add up to a. Therefore, the amount of collateral slashed per committee member $b = \frac{a}{n-t'} - \frac{f}{n}$. The contract keeps track of the collateral balance cl_i for each committee member i that is deducted when slashed and added to when rewards are earned from submitting valid shares. We introduce a function slash (Algorithm 6) that slashes committee members and transfers the reparation price to the dealer as outlined above.

To ensure the contract can use a committee member's collateral to pay the reparation price, the committee member must have deposited at least b inside the contract before encrypt can be called. However, the number of honest committee members who will submit valid shares t' is unknown at the time of an encrypt request. Therefore, the contract must ensure that each committee member has deposited at least the maximum slashable amount given the reparation price. Let $\hat{b}$ be the amount of funds a committee member is required to deposit.

$$\hat{b} = \max_{0 \leq t' \leq t-1} b = \frac{a}{n-t+1} - \frac{f}{n} \tag{1}$$

Notice that each committee member's collateral is locked inside the contract for the lifetime of the secret. However, the locked funds do not earn interest,

Algorithm 6 Cassiopeia slash function

```
1: function slash(i)
2:     require(M_id = CLAIMED ∧ block.number ≥ D_id)
3:     {dealer, a, f, ...} ← C_id
4:     t' ← |S_id|
5:     G ← 0
6:     b ← a/(n−t') + f/n
7:     b̂ ← a/(n−t+1) + f/n                          ▷ Equation 1
8:     for i ∈ [n] do
9:         if S_id,i = ⊥ then
10:            if t' < t then
11:                cl_i ← cl_i − b
12:                G ← G + b
13:            end if
14:            G ← G + f/n
15:        end if
16:    end for
17:    dealer.send(G)
18:    l ← l − b̂
19:    M_id ← SLASHED
20: end function
```

which introduces an opportunity cost for committee members. To incentivize committee members to participate in the protocol honestly, the reward must be higher than the opportunity cost. Concretely, let d be the *maximum lifespan* of the secret in blocks. The opportunity cost for locking $\hat{b}$ as collateral inside the contract for d blocks is

$$o = \hat{b}((1+r)^d - 1) \tag{2}$$

where r is the per-block risk-free rate agreed upon by all committee members.

In fact, we will see that the reparation price a is limited by the fee and the risk free rate. Intuitively, the higher the risk free rate is and the lower the fee is, the lower the maximum reparation price will be. Suppose committee members agree upon a target time-valued net profit β for honest committee members, where $\beta = \frac{f}{n} - o$. Combining this with Equation 1 and Equation 2, we have that the maximum value of a is:

$$a = \frac{(\frac{f}{n}(1+r)^d - \beta)(n-t+1)}{(1+r)^d - 1} \tag{3}$$

Instead of letting the user specify the reparation price, we would like the contract to compute the reparation price a directly from the holding fee. To do so, the maximum lifespan of the secret d must be known. However, for an arbitrary relation $\mathcal{R}$ the witness may take an arbitrary amount of time to be found. Therefore, we require the dealer to also provide the maximum number of blocks T that committee members will keep the secret for. Let st be the block where the encryption request was first made. We modify the original claim

function to start decryption at block $st + T$ even if no valid witness has been revealed. Alternatively, the protocol could also be defined such that all secret shares are destroyed after time $st + T$.

The lifespan of the secret d also includes the number of blocks between when the secret is claimed and when committee members are slashed, after which the secret can be decrypted. For the maximum lifespan of the secret to be known, everyone must agree upon a share submission deadline D_{id} after which committee members are slashed. The deadline D_{id} is set to Δ blocks (corresponding to the liveness parameter of the blockchain [14]) after the secret is claimed. Honest committee members must submit their shares by block D_{id}. Therefore, the secret's maximum lifespan is $d = T + \Delta$, and is known at encryption time. We modify the claim function as in Algorithm 7.

Algorithm 7 Modified Cassiopeia claim function

```
1: function claim(id, w)
2:     require(M_id = ⊥)
3:     {st, x, T} ← C_i
4:     require(block.number ≥ st + T ∨ (x, w) ∈ R)
5:     D_id ← block.number + Δ
6:     M_id ← CLAIMED
7: end function
```

We call a secret *active* if slash has not been called for the secret. Because many secrets may be active, the contract must ensure it has sufficient funds in escrow to cover all potential reparations. Let l be the sum of $\hat{b}$ for all secrets. In encrypt, the contract ensures the remaining collateral $cl_i - l$ is at least the new secret's $\hat{b}$. The fully modified encrypt function is shown below in Algorithm 8.

Lastly, we must allow committee members to deposit and withdraw their collateral to collect fees or stop participating in the protocol. The only requirement is that the remaining collateral is sufficient to cover the worst case reparation price for all active secrets. Concretely, committee member i can withdraw δ_{cl} only if $cl_i - \delta_{cl} \geq l$.

5 Analysis

We assume a synchronous network model where there exists a probabilistic polynomial time adversary. The adversary controls some committee members, which can do whatever they like. Let PVSS = (genDist, verifyDist, decrypt, verifyShare, reconstruct) be a correct and verifiable PVSS scheme that has secrecy, (G, P, V) be a correct and extractably secure zkSNARK scheme, (Enc, Dec) be a correct and secure symmetric encryption scheme, $\mathbb{L}$ be a safe and live ledger, and H be a random oracle. We will analyze the security properties of both the non-incentivized and incentivized constructions.

Algorithm 8 Modified Cassiopeia encrypt function

function encrypt$(c, \mathcal{R}, x, T, y, \pi)$ **payable**
  ▷ *Incentives*
  $d \leftarrow T + \Delta$
  $f \leftarrow$ msg.value
  $a \leftarrow \frac{(\frac{f}{n}(1+r)^d - \beta)(n-t+1)}{(1+r)^d - 1}$ ▷ Equation 3
  $\hat{b} \leftarrow \frac{a}{n-t+1} - \frac{f}{n}$ ▷ Equation 1
  for $i \in [n]$ **do**
    require$(cl_i \geq l + \hat{b})$
  end for
  $l \leftarrow l + \hat{b}$
  $st \leftarrow$ block.number
  ▷ *Verify PVSS and zkSNARK*
  $([\hat{s}_i], \pi_D) \leftarrow c$
  require$(V(\tau, (c, \mathcal{R}, x, T, y, [pk_i]), \pi) = 1 \wedge$ PVSS.verifyDist$([\hat{s}_i], [pk_i], \pi_D))$
  $id \leftarrow H(c, \mathcal{R}, x, T)$
  $C_{id} \leftarrow \{c, \mathcal{R}, x, T, [\hat{s}_i], f, a, st\}$
  return id
end function

5.1 Security Analysis of Non-incentivized Construction

Theorem 1. *Correctness (Informal). Consider an honest dealer and a committee of size n such that at least t members are honest. The social Witness Encryption construction in Section 3 is correct. See Appendix A for proof.*

Theorem 2. *Security (Informal). Consider an honest dealer, and a committee of size n such that less than t members are adversarial. The social Witness Encryption construction in Section 3 is extractably secure. See Appendix B for proof.*

At the heart of the proof of security lies the following Lemma. Intuitively, the attack that Lemma 1 ensures protection from is a malleability attack. In particular, an adversary may use the arguments $(c, \mathcal{R}, x, y, \pi)$ of a previous encrypt call to generate arguments $(c', \mathcal{R}', x', y', \pi')$ such that the decryption of c' can be used to infer the decryption of c (i.e. the original secret).

Lemma 1. *Suppose an adversary is given the values $c, \mathcal{R}, x, y, \pi$ for secret s generated by an honest dealer, and the adversary interacts with the contract. The adversary cannot obtain s without a witness w such that $(x, w) \in \mathcal{R}$. See Appendix B for proof.*

Now, we analyze the security properties of the non-incentivized scheme under honest majority. Let h be the number of honest committee members. Assume that a majority of committee members are honest (i.e. $h \geq \frac{n}{2} + 1$). If $t = \frac{n}{2}$, then the construction is correct, since $h \geq t$. Furthermore, the construction is also extractably secure, because the number of adversaries $n - h \leq \frac{n}{2} - 1 < t$.

Note that the construction may be instantiated with different values of t. Since the minimum number of honest committee members needed to ensure correctness and security is t and $n-t+1$ respectively, higher t ensures a higher degree of correctness for lower security and vice versa. When both correctness and security are desired with the lowest possible minimum required number of honest committee members, we optimize for $\arg\min_t \min(t, n-t+1)$, which occurs when $t = \frac{n}{2}$, corresponding with the honest majority case.

5.2 Security Analysis of Incentivized Construction

We assume there exists a fixed per-block risk-free rate r, as outlined in Section 2.6.

Lemma 2. *If a committee member submits a valid share within the grace period, its time-valued net compensation is at least β.*

Proof (Sketch). Since the PVSS scheme is verifiable, the contract will only accept the dealer's PVSS ciphertext c if it is valid. Furthermore, the contract will only accept a committee member's encrypted share if it is correct. Therefore, a committee member will only be paid its reward of $\frac{f}{n}$ if it submits a valid share. Let d be the maximum lifetime of the secret. Since the committee member also deposits its collateral for d blocks, the committee member's time-valued net compensation is

$$\frac{f}{n} - o = \frac{f}{n} - (\frac{a}{n-t+1} - \frac{f}{n})((1+r)^d - 1) = \beta$$

Theorem 3. *Correctness (Informal). Consider an honest dealer, and a committee of size n such that all maintain sufficient collateral for the holding f and at least t members are rational. The social Witness Encryption construction in Section 4 is correct.*

Proof (Sketch). We follow the same proof as that of Theorem 1 but with added steps. If a committee member does not submit a valid share by the deadline, they forfeit both their reward and are slashed. By Lemma 2, a committee member who submits their shares will receive a net gain of β. Since we assume that β is a sufficient reward, rational committee members will choose to submit valid shares of the secret on time. Because we assume all committee members hold sufficient collateral in the contract, $\mathsf{WE.Enc}_{\mathcal{R}}$ completes in polynomial time. Since there are at least t rational committee members, S_{id} contains at least t valid shares. As in the proof of Theorem 1 above, the correctness of PVSS ensures that $\mathsf{reconstruct}(S_{id}) = s$, and by the correctness of the symmetric encryption scheme, $Dec_s(\hat{c}) = m$. Therefore, $\mathsf{WE.Dec}_{\mathcal{R}}$ correctly recovers m in polynomial time.

Theorem 4. *Payout (Informal). Consider an honest dealer, and a committee of size n such that all maintain sufficient collateral for the holding f. If less than t committee members submit valid shares, the dealer receives a.*

Proof (Sketch). Let t' be the number of committee members who did not submit valid shares. Committee members who do not submit valid shares do not receive their reward, which is given to the dealer in Line 14 of Algorithm 6, along with their share of the reparation fee $\frac{a}{|t'|} - \frac{f}{n}$. The reparation fee can always be deducted from each committee member's collateral because $\hat{b} \geq \frac{a}{|t'|} - \frac{f}{n}$. Therefore, the dealer receives a in total.

6 Implementation

We implement all on-chain components of the non-incentivized scheme in Solidity and off-chain components in Typescript and Rust. We use SCRAPE [9] as the underlying PVSS scheme by adapting the implementation given by Gurkan et al. [22]. Since SCRAPE relies on Type 3 pairings, our PVSS ciphertext is instantiated using the curve BN254 [2], as at the time of writing only it is the only curve with Type 3 pairings available as precompiles on Ethereum. Similar to the scheme presented by Schoenmakers [31], SCRAPE's dealer shares the secret s but the committee recovers h^s, where h is the generator of $\mathbb{G}_2$ in BN254. Therefore, our implementation of the scheme requires the dealer to produce $\hat{c}$ by encrypting m with the truncated SHA256 hash of a h^s rather than use s directly.

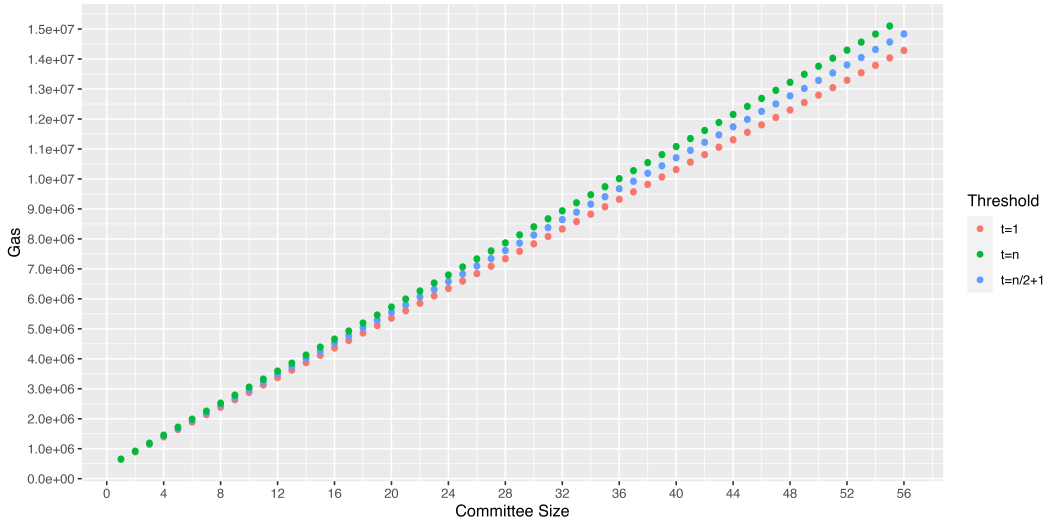
Though BN254 is not a SNARK friendly curve, we have optimized the proving time of the zkSNARK circuit from Algorithm 4 by leveraging the fact that SCRAPE's instantiation of $\mathsf{PVSS.verifyDist}$ allows us to be convinced that $\mathsf{PVSS.genDist}(s, [pk_i]) = c$ just by checking $F_0 = g^s$ (where g is the generator of $\mathbb{G}_1$). Since $\mathsf{PVSS.verifyDist}$ already checks consistency of F_0 with the rest of the ciphertext, we only need to check whether $\log F_0 = s$. This involves only one BN254 exponentiation inside of Algorithm 4 and makes the circuit size constant, allowing proving time to be constant. To optimize the proving time of the zkSNARK further, we compress the problem instance by first hashing $(c, \mathcal{R}, x)$ before passing it into H, and instantiate H using Poseidon [20], a SNARK friendly hash function. The primary bottleneck of performance is $\mathsf{PVSS.verifyDist}$, which occurs on chain and has optimal $O(n)$ complexity using SCRAPE.

We use a Hardhat node to measure the on-chain gas cost paid for by the dealer in $\mathsf{WE.Encrypt}_{\mathcal{R}}$, relative to n and t. As can be seen in Figure 6, the maximum committee size that does not consume more gas than the block gas limit is 56. The code for the implementation can be found at `https://github.com/galletas1712/cassiopeia`.

7 Conclusion

In this work, we propose Cassiopeia, a social Witness Encryption scheme instantiated as an on-chain smart contract, along with an incentivized version of the scheme. Our construction combines a publicly verifiable secret sharing scheme and zkSNARKs to provide security against malleability attacks. In doing so,

Fig. 1. Gas cost of `encrypt` vs committee size



the scheme is also resistant to front-running attacks which are widespread when interacting with smart contracts. We also provide a Solidity smart contract implementation of the non-incentivized scheme. The non-incentivized construction is correct and secure under honest majority, and the incentivized construction is correct under rational majority.

Future work will focus on ensuring security for the incentivized scheme under rational majority. Threshold Information Escrows have been proposed with incentivized security [26], and could potentially be integrated with our work. Furthermore, we could also allow for a dynamic committee by using proactive secret sharing, which would make our protocol more robust to security attacks.

References

1. L. Baird, P. Mukherjee, and R. Sinha. i-tire: Incremental timed-release encryption or how to use timed-release encryption on blockchains? Cryptology ePrint Archive, Paper 2021/800, 2021. `https://eprint.iacr.org/2021/800`.
2. P. S. L. M. Barreto and M. Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. In B. Preneel and S. Tavares, editors, *Selected Areas in Cryptography*, pages 319–331, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
3. F. Benhamouda, C. Gentry, S. Gorbunov, S. Halevi, H. Krawczyk, C. Lin, T. Rabin, and L. Reyzin. Can a public blockchain keep a secret? Cryptology ePrint Archive, Paper 2020/464, 2020. `https://eprint.iacr.org/2020/464`.
4. F. Benhamouda and H. Lin. Multiparty reusable non-interactive secure computation. Cryptology ePrint Archive, Paper 2020/221, 2020. `https://eprint.iacr.org/2020/221`.

5. N. Bitansky, A. Chiesa, Y. Ishai, O. Paneth, and R. Ostrovsky. Succinct non-interactive arguments via linear interactive proofs. In A. Sahai, editor, *Theory of Cryptography*, pages 315–333, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
6. D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, page 213–229, Berlin, Heidelberg, 2001. Springer-Verlag.
7. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. Cryptology ePrint Archive, Paper 2010/543, 2010. `https://eprint.iacr.org/2010/543`.
8. M. Campanelli, B. David, H. Khoshakhlagh, A. Konring, and J. Nielsen. Encryption to the future: A paradigm for sending secret messages to future (anonymous) committees. In *Advances in Cryptology – ASIACRYPT 2022*, Lecture Notes in Computer Science. Springer, Jan. 2023.
9. I. Cascudo and B. David. Scrape: Scalable randomness attested by public entities. Cryptology ePrint Archive, Paper 2017/216, 2017. `https://eprint.iacr.org/2017/216`.
10. B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 383–395, 1985.
11. A. R. Choudhuri, M. Green, A. Jain, G. Kaptchuk, and I. Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 719–728, New York, NY, USA, 2017. Association for Computing Machinery.
12. N. Döttling, L. Hanzlik, B. Magri, and S. Wohnig. Mcfly: Verifiable encryption to the future made practical. Cryptology ePrint Archive, Paper 2022/433, 2022. `https://eprint.iacr.org/2022/433`.
13. M. Ende. *Momo*. Puffin, München, 2009.
14. J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In E. Oswald and M. Fischlin, editors, *EUROCRYPT (2)*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.
15. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 40–49, 2013.
16. S. Garg, C. Gentry, A. Sahai, and B. Waters. Witness encryption and its applications. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, page 467–476, New York, NY, USA, 2013. Association for Computing Machinery.
17. C. Gentry, A. B. Lewko, and B. Waters. Witness encryption from instance independent assumptions. Cryptology ePrint Archive, Paper 2014/273, 2014. `https://eprint.iacr.org/2014/273`.
18. S. Goldwasser, Y. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. How to run turing machines on encrypted data. Cryptology ePrint Archive, Paper 2013/229, 2013. `https://eprint.iacr.org/2013/229`.
19. V. Goyal, A. Kothapalli, E. Masserova, B. Parno, and Y. Song. Storing and retrieving secrets on a blockchain. In *Public-Key Cryptography – PKC 2022: 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8–11, 2022, Proceedings, Part I*, page 252–282, Berlin, Heidelberg, 2022. Springer-Verlag.

20. L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *Proceedings of the 30th USENIX Security Symposium*, Proceedings of the 30th USENIX Security Symposium, pages 519–535, United States, 2021. USENIX Association. 30th USENIX Security Symposium : USENIX Security 2021, USENIX Security '21 ; Conference date: 11-08-2021 Through 13-08-2021.
21. J. Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *Proceedings of the 12th International Conference on Theory and Application of Cryptology and Information Security*, ASIACRYPT'06, page 444–459, Berlin, Heidelberg, 2006. Springer-Verlag.
22. K. Gurkan, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu. Aggregatable distributed key generation. Cryptology ePrint Archive, Paper 2021/005, 2021. `https://eprint.iacr.org/2021/005`.
23. S. Heidarvand and J. L. Villar. Public verifiability from pairings in secret sharing schemes. In *Selected Areas in Cryptography*, 2008.
24. P. Labs. tlock: Timelock encryption/decryption made practical. `https://github.com/drand/tlock`.
25. J. Liu, T. Jager, S. A. Kakvi, and B. Warinschi. How to build time-lock encryption. *Des. Codes Cryptography*, 86(11):2549–2586, November 2018.
26. E. V. Mangipudi, D. Lu, A. Psomas, and A. Kate. Collusion-deterrent threshold information escrow. Cryptology ePrint Archive, Paper 2021/095, 2021. `https://eprint.iacr.org/2021/095`.
27. W. Mao. Timed-release cryptography. Cryptology ePrint Archive, Paper 2001/014, 2001. `https://eprint.iacr.org/2001/014`.
28. C. Mascia, M. Sala, and I. Villa. A survey on functional encryption, 2021.
29. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, USA, 1996.
30. A. Ruiz and J. L. Villar. Publicly verifiable secret sharing from paillier's cryptosystem. In C. Wulf, S. Lucks, and P.-W. Yau, editors, *WEWoRC 2005 – Western European Workshop on Research in Cryptology*, pages 98–108, Bonn, 2005. Gesellschaft für Informatik e.V.
31. B. Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, page 148–164, Berlin, Heidelberg, 1999. Springer-Verlag.
32. A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, nov 1979.
33. M. Stadler. Publicly verifiable secret sharing. In U. Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, pages 190–199, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.
34. D. Zindros. Hours of horus: Keyless cryptocurrency wallets. Cryptology ePrint Archive, Paper 2021/715, 2021. `https://eprint.iacr.org/2021/715`.

8 Appendix A: Proof of Correctness For Non-incentivized Construction

Theorem 5. *Correctness (Informal). Consider an honest dealer and a committee of size n such that at least t are honest. The social Witness Encryption construction in Section 3 is correct.*

Proof (Sketch). First, we will prove that $\mathsf{WE.Enc}_{\mathcal{R}}(1^{\lambda}, x, m)$ successfully returns the secret identifier id in polynomial time. Since $Enc, \mathsf{PVSS.genDist}, H, P$ are all PPT algorithms, they all return in polynomial time. Since the PVSS scheme is correct and c was honestly generated, the $\mathsf{PVSS.verifyDist}$ check in Line 5 of $\mathsf{encrypt}$ passes. Similarly, since the zkSNARK scheme is correct and π was honestly generated, the zkSNARK verifier V also accepts the proof π. Since $\mathbb{L}$ is a safe and live ledger, $\mathsf{encrypt}$ returns id in polynomial time without reverting. By extension, $\mathsf{WE.Enc}_{\mathcal{R}}$ also returns id in polynomial time.

Now, we will prove that $\mathsf{WE.Dec}_{\mathcal{R}}(1^{\lambda}, x, id)$ returns m in polynomial time. The random oracle H ensures that no secret identifiers are duplicate with overwhelming probability, so the secret with identifier id corresponds to the same one encrypted in $\mathsf{WE.Enc}_{\mathcal{R}}$. Since $\mathbb{L}$ is a safe and live ledger, claim will update M_{id} will be updated to CLAIMED in polynomial time. Furthermore, since at least t committee members are honest, at least t of them will have submitted their shares Δ blocks after the secret was claimed. By the correctness of the PVSS scheme, the shares of all honest committee members pass the $\mathsf{PVSS.verifyShare}$ check in Line 18 of $\mathsf{submitShare}$. Therefore, $|S_{id}|$ will be at least t after all honest committee members submit their shares. Furthermore, reconstruction of the secret is also polynomial time. Since the symmetric encryption scheme is correct and $\hat{c}$ was produced honestly, $Dec_s(\hat{c})$ returns m in polynomial time. By extension, $\mathsf{WE.Dec}_{\mathcal{R}}$ also returns m in polynomial time.

9 Appendix B: Proof of Security For Non-incentivized Construction

Lemma 3. *Suppose an adversary is given the values $c, \mathcal{R}, x, y, \pi$ for secret s generated by an honest dealer, and the adversary interacts with the contract. The adversary cannot obtain s without a witness w such that $(x, w) \in \mathcal{R}$.*

Proof (Sketch). The only way the adversary can obtain any information about s interacting with the contract is to first call $\mathsf{encrypt}$ with $(c', \mathcal{R}', x', y', \pi')$, then performing $\mathsf{WE.Dec}_{\mathcal{R}'}(1^{\lambda}, c', x', w')$ to retrieve the secret s', where w' is a witness for $\mathcal{R}'$ and x' already known to the adversary. The adversary aims to deduce non-negligible information about s from s' and $(c, \mathcal{R}, x, y, \pi)$.

Suppose the adversary can extract non-negligible information about s. Because the PVSS scheme has secrecy, s cannot be extracted from c, since c is an encryption of s. Since y is the output of a random oracle, all preimages are equally probable, so y contains no information about s. By the zero knowledge property of the zkSNARK scheme, π does not reveal any information about the private input s. Therefore, the adversary deduces information about s using s'.

Consider when $s' = s$ and $c' = c$. When the adversary calls $\mathsf{encrypt}$, the ledger's safety and liveness ensures she must specify $y' = H(s \,\|\, c \,\|\, \mathcal{R}' \,\|\, x')$ and produce a valid zkSNARK proof π for the circuit in Algorithm 4. By the extractable soundness property of the zkSNARK, the adversary must know the entire preimage of y' prior to $\mathsf{encrypt}$. The adversary must also provide s such

that $c = \mathsf{PVSS.genDist}(s, [pk_i])$. Furthermore, since the preimage of y and y' are not equal, y cannot be used in place of y', nor is it related to y' is any way. Therefore, the adversary must know s prior to encryption. However, since the dealer is honest, the adversary cannot have access to s prior to encryption, which is a contradiction.

Now consider when $s' \neq s$ and $c' \neq c$. Since s' is the decryption of c', c' must have encrypted some information about s. But y contains no information about s, c' must have been derived from c. When the adversary calls encrypt, the ledger's safety and liveness ensures she must specify $y' = H(s' \,\|\, c' \,\|\, \mathcal{R}' \,\|\, x')$ and produce a valid zkSNARK proof π for the circuit in Algorithm 4. Because the zkSNARK is extractably sound, y' is the output of a random oracle function, and $c' = \mathsf{PVSS.genDist}(s', [pk_i])$ is enforced in the zkSNARK circuit, the adversary must know s' prior to the computation of y' (i.e. before the call to encrypt). But this means the adversary can know s with knowledge of only $c, \mathcal{R}, x, y, \pi$, which is a contradiction.

In either case, the adversary cannot obtain s even by interacting with the contract.

Theorem 6. *Extractable Security (Informal). Consider an honest dealer, and a committee of size n such that less than t are adversarial. The social Witness Encryption construction in Section 3 is extractably secure.*

Proof (Sketch). First, consider the case where the contract will accept only one secret to encrypt in its lifetime. The dealer is honest, so no other party (including the adversary) has access to s. Since s is a bit string of length λ and the symmetric encryption scheme is secure, $\hat{c}$ reveals nothing about s. Similarly, since the PVSS scheme has secrecy, the PVSS ciphertext c reveals nothing about s. Furthermore, because less than t committee members are adversarial, no coalition of adversarial committee members can learn any information about the secret. The random oracle H ensures that y is uniformly distributed and $Pr[y = H(s \,\|\, c \,\|\, \mathcal{R} \,\|\, x) \mid c, \mathcal{R}, x]$ is negligible, so a PPT adversary cannot learn anything about s from y. By the zero knowledge property of the zkSNARK, the proof π also reveals nothing about s. Therefore, all public values reveal nothing about s.

Now, suppose the contract may accept multiple secrets in its lifetime. By Lemma 3, no matter what other encryption requests the adversary makes, she cannot learn any information about s, even if she has access to the public arguments of all encrypt calls that will ever occur.

Without having submitted a valid witness w, the adversary does not learn any information about s, whether from public data or from interacting with the contract. Therefore, if the adversary learns any information about s, they must know a valid witness w, and the construction is extractably secure.