Dead Man's Switch Cryptography

Shalini Banerjee[⋆‡], Andrey Bozhko[†], Raphael Heitjohann[†⋆], Andy Rupp[†‡]

[⋆]Karlsruhe Institute of Technology
[‡]KASTEL Security Research Labs
shalini.banerjee@kit.edu
[†]University of Luxembourg
{andrey.bozhko, andy.rupp}@uni.lu
[†⋆]University of Wuppertal
heitjohann@uni-wuppertal.de

July 1, 2026

Abstract

We initiate the formal study of "dead man switch cryptography" — the problem of cryptographically enforcing the fate of a long-lived secret upon the death of its *sole* keeper — either by releasing it exclusively to designated nominees or by provably destroying it. Despite the natural practical demand, no prior work formalizes this primitive.

We identify Dead Man's Switch (DMS) as a standalone cryptographic primitive with two modes: release and delete, capturing these dual functionalities. Our abstraction isolates the core conceptual properties of this primitive, and gives its first rigorous treatment in the UC framework. To realize the notion, we introduce *Dealer-Controlled Evolving-Committee Proactive Secret Sharing* (DC-ECPSS), an extension of the Evolving-Committee Proactive Secret Sharing by Benhamouda *et al.* (TCC '20). We provide the UC formalization of DC-ECPSS and instantiate it generically using the nested YOSO framework of Abraham, Chouatt, Damgard, Gilad, Stern, and Yakoubov.

Building on DC-ECPSS, we realize DMS-release by coupling it with a multi-recipient authorized recovery mechanism based on a PKI infrastructure, a multi-recipient PKE, a key committing SKE, and NIZK proof systems, while DMS-delete directly follows from DC-ECPSS instantiated in the delete mode. We prove the security of both constructions in the UC framework under standard assumptions.

Keywords: Dead man's switch, YOSO, evolving-committee PSS

Contents

1 Introduction **3**
- 1.1 Our Contributions . . . 5
- 1.2 Our Approach . . . 5
- 1.3 Further Discussions . . . 8

2 Preliminaries **9**
- 2.1 Notation . . . 9
- 2.2 Secret Sharing Schemes . . . 9
- 2.3 Re-sharing . . . 10
- 2.4 Linear Secret Sharing Schemes . . . 11
- 2.5 Re-sharing for Linear Secret Sharing Schemes . . . 11
- 2.6 Nested Shamir Secret Sharing . . . 12

3 Dealer-Controlled EPSS **13**
- 3.1 Modeling DC-ECPSS . . . 13
- 3.2 The protocol $\pi_{\mathsf{DC\text{-}ECPSS}}$ . . . 14
- 3.3 Security of $\pi_{\mathsf{DC\text{-}ECPSS}}$ . . . 17
- 3.4 A Direct Realization of DMS-delete . . . 19
- 3.5 From Abstract to Real World . . . 19

4 Dead Man's Switch - Release **20**
- 4.1 Modeling DMS . . . 20
- 4.2 Protocol π_{DMS} . . . 21
- 4.3 Security of π_{DMS} . . . 24

A Additional Preliminaries **28**
- A.1 Commitment Schemes . . . 28
- A.2 Public Key Encryption . . . 28
- A.3 Multi-Recipient Public Key Encryption . . . 28
- A.4 Symmetric-Key Encryption . . . 29
- A.5 Universal Composability Framework . . . 29

B Proof of Theorem 1 **32**

C Proof of Theorem 2 **35**

1 Introduction

What happens to a secret when its sole keeper unexpectedly dies? In 2018, this question came alive with the alleged death of Gerard Cotten, the founder of the Canadian cryptocurrency exchange QuadrigaCX. More than 100,000 investors were permanently locked out of their cryptocurrency holdings worth millions of dollars, as Cotten was the sole keeper of the key to the firm's cold wallet[1]. No backup existed, no recovery mechanism were triggered — the secret simply became unreachable. This irreversible loss is not due to any adversarial hack, but because there is no immediate way to prevent secrets from falling into oblivion.

In fact, the QuadrigaCX collapse is not merely a problem unique to cryptocurrency. A government dignitary may hold a password granting access to classified information critical to national security that must be seamlessly transferred to a designated nominee on the event of their death, incapacitation, or capture. A whistleblower may possess secret information that must reach a journalist or a legal advocate upon their death. An entrepreneur may possess proprietary algorithms or business-critical secrets that must transfer to a successor upon sudden incapacitation. In each of these scenarios, a fundamental question arises: not merely what happens to a secret when its keeper dies, but also can we use cryptography to guarantee what *should* happen?

The answer depends on what must become of a secret once its keeper, whom we henceforth call "testator" (following the convention of [SST20]), is no longer able to protect it. In precisely the settings discussed above, the desired outcome is clear: transfer a secret from a testator to a designated nominee(s) on the event of the testator's death. A complementary yet fundamentally unified requirement arises in another setting: a secret must be provably destroyed after a testator's death — not because there is no intended recipient, but because its existence poses a greater risk than its loss. For example, a researcher who holds sensitive personal data about human subjects may require the data to be provably destroyed after their death. The closest study in this domain is the Vanish project [Wik25] which makes a secret unrecoverable after a fixed time horizon, rather than an unpredictable death event.

In fact, the need is already recognized in practice: Google's Inactive Account Manager allows a user to designate upto ten trusted contacts who may access their account on their prolonged inactivity, or request account deletion, nevertheless at the cost of Google acting as a trusted third party (TTP).

Despite the apparent importance, the requirements remain largely unaddressed by the existing literature. Understanding this gap requires revisiting the cryptographic primitives that closely resemble our setting — the most immediate choice being the family of time-based cryptographic primitives. The intuitive idea was formalized by May [Tim93], where a TTP hands over a secret after τ period of time has elapsed. While conceptually elegant, this exposes a deeper tension: the absence of cryptographic primitives for time-based release necessitates external enforcement mechanisms. The natural question is why does the testator simply not hand over the secret to the nominee. The answer is precisely what motivates the trusted third party — the TTP exists just to enforce the temporal condition.

This shifts the focus from protecting the secret itself to enforcing the time constraint cryptographically. Time-lock puzzles and the broader primitive of timed-released cryptography by Rivest, Shamir and Wagner [RSW96] replace TTP by an efficient computation as follows: encrypt a secret w.r.t. a puzzle, solving which essentially requires time τ.

Yet, a fundamental mismatch remains: τ must be fixed at the setup time, whereas the event we wish to condition on — "testator's death" — is not known a priori. If an arbitrary τ is chosen too small, a secret may be released while a testator is still alive; if τ is too large, then a nominee may be withheld from the secret long after testator's death, thereby defeating the purpose altogether. Even setting this limitation aside, in both May's construction and time-lock puzzles, the secret is released to whoever obtains the decryption key or solves the puzzle first. Neither construction captures the notion of an authorized nominee. Finally, a third and more subtle failure remains — what if a testator is malicious? We say a testator is malicious if they participate in a protocol but register a fraudulent secret that does not correspond to the intended bequest, leaving nominees with no recourse. We stress that this is not merely a theoretical

[1]The collapse of QuadrigaCX became the subject of the Canadian documentary "Dead Man's Switch: A Crypto Mystery" [Wik22]. The circumstances surrounding Cotten's death and the inaccessibility of the funds remain disputed.

concern: the QuadrigaCX collapse precisely illustrates this risk. Worth mentioning, there exists literature on timed commitments [BN00] which requires committing to a value at setup time such that the commitment is binding, however the remaining failures remain unchanged. In fact, the existing time-based primitives fundamentally model a different problem which precisely is not what our setting demands; rather we need a different trigger conditioned on an event that takes place at an a priori unknown time.

One might hope that verifiable delay functions (VDFs) introduced by Boneh, Bonneau, Benedikt, Bunz, and Fisch [BBBF18] may offer a way forward. Specifically, a VDF computes $y = f(x, \tau)$ that requires exactly τ period of time to evaluate, and produces a short proof π that anyone can verify in $\log(\tau)$ time without evaluating f. McCadden and McGuinn [MM22] build upon this to design a framework, where a testator computes a VDF as $y = f(r, \tau)$ for a random public challenge r using a secret trapdoor key tk, derives an encryption key as $k = \mathsf{KDF}(y, r)$, performs an AES encryption of the secret s as $\mathsf{ct} = \mathsf{AES}(k, s)$, and finally secret-shares ct among static servers *at every epoch* — which is precisely the alive signal. If a testator fails to do so, one can compute $y = f(x, \tau)$ and reconstruct s, which requires time τ without the knowledge of tk. Thus τ no longer denotes a fixed release deadline but defines the epoch duration within which a testator must provide an alive signal — an intuition aptly captured by the authors in the title "A Cryptographic Dead Person's Switch".

At a first glance, this appears to come remarkably close to capturing the functionality required in our setting — death is abstracted as an absence of alive signal and trust is distributed via an m-out-of-n secret sharing scheme. However, upon closer inspection, we observe several gaps. Anyone who computes a valid VDF output (y, π), can receive a share of ct, i.e. there is no way to identify an authorized nominee. Adding to this, the system cannot verify if a server has honestly deleted an old share upon receiving a new one. Also, the trapdoor key tk needs to be protected along with s. Furthermore, the construction does not provide a way to identify a malicious testator. Finally, the alive signal is heavyweight — a full execution of the sharing phase, requiring the testator to maintain persistent access to the secret throughout their lifetime.

Essentially, we require a formal treatment of the primitive itself, one that defines an honest testator, authorized nominees, lightweight alive signals, along with properties that such a primitive should satisfy. This is precisely what Seres, Shlomovits and Tiwari [SST20] attempt to answer in their CryptoWills framework. Their core idea is to allow a testator to hand over a secret to a set of nominees upon their death — a standalone cryptographic primitive where death is defined as an absence of response signal within an epoch of duration τ — with precisely the following guarantees: *correctness* which ensures that nominees procure correct secret following a testator's death, *soundness* which says that only authorized nominees can procure a secret, *heir-hiding* which ensures that the set of nominees are not revealed, and *bequest-hiding* which ensures that a secret is well-hidden before testator's death. Their primary construction relies on a TEE, while a second construction is based on a threshold committee via CP-ABE.

However, their guarantees rest on assumptions that are fragile in our setting: the authors explicitly mention "if the testator is corrupt, then there is no meaningful CryptoWills", but we take into account a malicious testator (recall the QuadrigaCX collapse). Additionally, the heir-hiding property restricts a nominee from knowing whether they are indeed one until the testator's death, whereas in our setting, this guarantee is often too strong: a nominee (such as an investor) may require ex-ante cryptographic assurance that they have indeed been nominated. We depart from the heir-hiding property and introduce a strictly stronger notion of *nominee privacy* that allows a nominee to independently verify their inclusion before the testator's death. Finally, the handover phase relies on static servers that maintain a secret from setup until the testator's death, providing no security against a mobile adversary, and there is no way to achieve provable destruction of the secret upon testator's death. Consequently, it seems natural to ask:

Can we design a cryptographic dead man's switch that conditions the fate of a secret on a testator's death — either by releasing it to authorized nominees, or by provably destroying it — and do so without the aforementioned limitations?

We believe this is a promising direction to explore in the sense that a positive result will indicate a cryptographic foundation that controls secrets beyond the lifetime of their keeper[2].

[2] `DeadMan'sSwitch.sol` on Ethereum already show its practical significance, but offer no confidentiality of secret, nominee

Towards an affirmative direction, we introduce Dead Man's Switch (DMS), a cryptographic primitive with two *dual* modes: release - a secret is released to authorized nominees upon a testator's death, and delete - a secret is provably destroyed upon a testator's death.

1.1 Our Contributions

We now summarize our main contributions:

- The first contribution of this work is to identify "Dead Man's Switch" as a new cryptographic primitive, and give its first formal treatment. Our DMS notion captures two dual modes: DMS-release, in which a long-term secret is released to authorized nominees upon the testator's death, and DMS-delete, where a secret is provably destroyed, instead. We capture the modes in an unified ideal functionality $\mathcal{F}_{\mathsf{DMS}}$ in the UC framework that captures correctness, soundness, bequest-hiding, nominee privacy, and liveness. While we follow the notational convention of CryptoWills [SST20], we depart from their heir-notion and instead use nominee-privacy as nominees themselves can verify if they are included in a bequest at any point during the testator's lifetime. Our formalism further treats the testator as potentially malicious, enforcing well-formedness of the committed secret at the setup via a relation $\mathcal{R}$ checked by the functionality.

- We introduce a dealer-controlled variant of ECPSS introduced by Benhamouda, Gentry, Gorbunov, Halevi, Krawczyk, Lin, Rabin and Reyzine [BGG+20], which we call DC-ECPSS — a new primitive that additionally enables dealer control in epoch-wise propagation of a long-term secret. In particular, a current holding committee may execute the handover phase only upon receiving an authenticated alive signal from the dealer, absence of which triggers transition to a terminal phase. We formalize DC-ECPSS in the UC framework via an ideal functionality $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$. We believe that DC-ECPSS may be of independent interest as a general abstraction for *dealer controlled secret management*. Our re-sharing protocol adapts the nested YOSO committee structure of [ACD+25], originally designed for general secure computation, to the PSS setting. The nested structure tolerates higher number of adversaries compared to [BGG+20]. This improvement is particularly interesting in a dead man's switch setting, where a testator may live for decades.

- In formalizing DC-ECPSS, we additionally obtain the first rigorous UC treatment of ECPSS, to the best of our knowledge. Our formalization corresponds to a special case of our dealer-controlled functionality $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$, where dealer control is absent, and may be of independent interest for future work on evolving-committee protocols.

- We generically construct both DMS-release and DMS-delete from DC-ECPSS. Our DMS-delete is a straightforward adaptation of our DC-ECPSS protocol. To realize DMS-release, we augment DC-ECPSS with a nominee encryption layer: rather than maintaining the secret itself, DC-ECPSS now maintains a compact cryptographic state that can uncover the secret, while the later remains encrypted publicly throughout its lifetime. Our DMS-release construction additionally assumes an IND-Anon-CPA secure multi-recipient PKE [KKPP20], an IND-CPA secure PKE, a key-committing symmetric key encryption scheme SKE, and an NIZK proof system that binds publicly available ciphertexts to values committed in DC-ECPSS. We prove that the resulting protocol π_{DMS} UC-realizes $\mathcal{F}_{\mathsf{DMS}}$ in the $(\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{DC\text{-}ECPSS}})$ hybrid model under these assumptions.

1.2 Our Approach

Having established our goal, two natural starting points present themselves. The first candidate is witness encryption (WE), introduced by Garg, Gentry, Sahai and Waters [GGSW13], which allows encrypting a secret relative to an NP statement such that decryption is possible by anyone who holds a valid witness. In

privacy, or security from a malicious testator.

our setting, one might hope to encode testator's death as a statement e.g. presence of an external attestation. The witness is then the certificate itself, and decryption is possible when the certificate is issued — albeit at the expense of a TTP — the assumption we seek to eliminate.

The second approach is to encode death as an absence of an alive signal — which gives no compact locally verifiable object, and lacks a succinct witness. This would crucially require additional machinery such as authentication logs, consensus certificates, etc. What's more, this approach treats death as an one-time satisfiable statement fixed at setup, whereas we require a long-term cryptographic state that must remain protected for an unknown duration. Finally, no efficient witness encryption construction exists under standard assumptions. We therefore, look ahead in a different direction — toward storing a secret securely until death occurs.

Secret sharing schemes split a secret into multiple shares such that no coalition holding less than threshold t shares can recover the secret. We note that the standard static corruption model where an adversary fixes the set of corrupt parties at the start of the protocol might not be apt in our long-term setting. Instead we need security against a *mobile* adversary, who may adaptively corrupt up to $t-1$ parties at any point of time, and over sufficiently long time may corrupt every party in the system.

Proactive secret sharing (PSS) [HJKY95] precisely addresses this threat by periodically refreshing shares over epochs, preventing an adversary from gradually accumulating t corrupted shares to reconstruct a secret. Nevertheless, PSS does not scale well — it requires all the parties to engage in the share-refresh protocol. Moreover the shareholders in each epoch, whom we henceforth refer to as "holding committees", are static and publicly known throughout the lifetime of the secret. A natural remedy is to replace static committees with small dynamically evolving holding committees - the central idea behind dynamic proactive secret sharing (DPSS) [DJ97, ZSVR05]. However, this does not ally all the fears, a mobile adversary may fully corrupt a small committee, smaller than its corruption budget, and recover the secret.

In their seminal paper [GHK+21], Gentry, Halevi, Krawczyk, Magri, Nielsen, Rabin, and Yakoubov introduce the YOSO model to address this. In a nutshell, they consider an open distributed system consisting of a large population of nodes $\mathcal{U}$ jointly maintaining a long-lived cryptographic computation. At each epoch, a small subset of nodes $n << |\mathcal{U}|$ form an anonymous holding committee who does some partial computation, hands over the state to the next anonymous holding committee by speaking exactly once, while erasing their state before speaking. Benhamouda, Gentry, Gorbunov, Halevi, Krawczyk, Lin, Rabin and Reyzin [BGG+20] introduce evolving-committee PSS (ECPSS) in the same spirit: whereas the YOSO model considers general secure computation, ECPSS specializes it to the PSS setting by allowing epoch-wise handover of a secret by re-sharing it among dynamically changing holding committees that are majorly honest (due to auxiliary role assigning "nominating committees"). The assignment of roles is random, and completely hidden from a mobile adversary who cannot predict the holding committee for the current epoch until they speak during the handover phase. Nonetheless, the holding committee speaks *only once* and erases their states, giving no advantage to a mobile adversary who has a corruption budget p and can control at most $p \cdot |\mathcal{U}| > t$ parties. *This forms the starting point of our construction.*

We observe that the [BGG+20] construction does not enable any interaction between a dealer ("testator" in our setting) and the system, beyond the initial secret sharing phase. Once a secret is shared, re-sharing progresses regardless of whether the dealer is alive or dead. On the contrary, our DMS setting requires a dealer to provide some "alive" signal at every epoch — such that the current holding committee cannot re-share until so. Put differently, we impose a *dealer-controlled* variant of ECPSS in the UC framework: a dealer inputs an alive signal to the functionality at each epoch, and the holding committee is permitted to re-share only upon receiving it. In absence of an alive signal — indicating testator's death — the holding committee declares itself as final, and the protocol transitions to its terminal phase. We call this primitive *dealer controlled* ECPSS (DC-ECPSS). This functionality can be realized, for example via a challenge-response protocol: a nominating committee publishes a fresh challenge at each epoch (alongside the usual ephemeral key-pairs required for secret propagation as in [BGG+20]), dealer computes a response to this challenge, which the holding committee verifies before executing the handover phase.

Furthermore, we consider the nested YOSO structure by Abraham, Chouatt, Damgard, Gilad, Stern, and Yakoubov [ACD+25]. Specifically, each nominator chooses a holding sub-committee consisting of m parties.

The idea is that corrupt nominators will choose sub-committees which are entirely corrupt, while honest nominators will choose sub-committees which are mostly honest. As long as more than half the nominators are honest, the majority of subcommittees will be mostly honest. This supposedly has the advantage that the number of committee members allowed to be corrupt is significantly higher than for the non-nested approach. The improvement is particularly valuable in a dead man's switch setting, where a testator may live for decades — giving a mobile adversary an extended window to corrupt the committee members.

To realize a secret sharing scheme for such a nested committee setting, [ACD+25] propose a secret sharing construction based on the following intuition: First, the secret is split into m shares, one for each subcommittee, using a t-out-of-m threshold scheme. Each subcommittee then splits its share again among its m members, again using the t-out-of-m threshold scheme. The authors of [ACD+25] also provide a specific secret sharing construction based on linear integer secret sharing [DT06]. Note that our application does not require secret sharing over the integers, therefore we make use of more efficient linear secret sharing over the field $\mathbb{Z}_q$. We also need a way for the entire holding committee to re-share the secret with the next epoch's holding committee. For this we rely on the re-sharing techniques presented in [BGG+20]. They only show how to realize re-sharing for the regular Shamir t-out-of-m threshold scheme scheme, while we require a re-sharing protocol for the nested extension of the Shamir scheme. To address this, we use the fact that the Shamir scheme and it's nested extension can both be represented as linear secret sharing schemes (LSSS), and show that the re-sharing approach from [BGG+20] can be extended to general LSSS. The result is a secret sharing scheme for nested committees with re-sharing functionality, exactly what we require for our construction of DC-ECPSS.

DC-ECPSS as DMS-delete: Our DC-ECPSS primitive directly realizes DMS-delete: upon receiving no alive signal from the testator in the delete mode, the holding committee for the current epoch broadcasts $(\texttt{finish}, \perp)$, and erases their local state — the secret simply stops to exist in the system. Additionally, the YOSO anonymity ensures that at any point, a mobile adversary cannot determine which committee to corrupt for reconstructing the secret.

Augmenting DC-ECPSS to realize DMS-release: Towards constructing DMS-release, we first observe that, while DC-ECPSS allows us to maintain a long-lived cryptographic state under adaptive corruption, the holding committees entrusted with maintaining the state must never learn the s — even at the release time. At the same time, only designated nominees should be able to recover s upon testator's death. Therefore we need a clean separation between one who maintains state and one who recovers secret. A natural approach is to encrypt s under a symmetric key and store the resulting ciphertext publicly (e.g. off-chain considering a blockchain setting), while only maintaining inside DC-ECPSS the state required to recover the key.

Following this intuition, a testator generates an ephemeral PKE pair $(\mathsf{epk}, \mathsf{esk})$, and encrypts a freshly sampled symmetric key K under epk to obtain c_0. The ciphertext c_0 is then passed to DC-ECPSS to be hidden until there is no alive signal. The secret s is then encrypted under K resulting in a ciphertext c_2, which is publicly broadcasted. The final step is to guarantee esk is only available to the nominees; for that the ephemeral secret key esk is encrypted towards the nominees' public keys using a multi-recipient PKE (mPKE) scheme, yielding c_1, published along with c_2.

Next, to prevent a malicious testator from storing a fraudulent secret, the pair (c_1, c_2) is published along with a NIZK proof π certifying that c_2 indeed encrypts a well-formed secret satisfying some relation $\mathcal{R}$. We note that $\mathcal{R}$ may be the trivially true relation — in applications where no well-formedness constraint on s is required.

Upon receiving no alive signal from a testator in the release mode, a current holding committee broadcasts $(\texttt{finish}, \mathsf{sh})$, and erases their local state. An authorized nominee then decrypts c_1 to recover esk, extracts K from c_0, and finally recovers the secret as $s = \mathsf{SKE.Dec}(K, c_2)$.

A notable consequence of this design is that a potential nominee can independently verify whether they have indeed been designated by the testator, at any point during the testator's lifetime. Since c_1 is published at the setup, anyone may locally try to decrypt it using their nominee secret key. Crucially, recovering esk before testator's death does not reveal s, as c_0 remain withheld by DC-ECPSS. This provides an ex-ante

assurance of inclusion — a guarantee absent in CryptoWills [SST20], where a nominee learns inclusion only after the testator's death. We note that for applications that do not require this assurance, one may use an mPKE scheme with an implicit rejection functionality.

Finally, unlike the VDF-based construction [MM22], where alive signal is a full execution of the sharing phase, requiring re-encrypting and re-sharing the secret at each epoch, our construction simply uses a single authenticated broadcast per epoch. Thus, there is no need to access the secret or do any heavyweight computation.

1.3 Further Discussions

Other Related Work. Beyond the works discussed above, several primitives and constructions relevant in our setting are worth mentioning.

Goyal, Kothapalli, Masserova, Parno and Song [GKM+22] propose extractable WE on Blockchain (eWEB) using a DPSS variant to simulate extractable WE. Section 10 of their paper discusses a dead man's switch application, where a secret is released to any party that satisfies a public predicate. We deviate from this in that we impose a strictly stronger requirement of releasing the secret to authorized nominees. Additionally, they do not consider a malicious testator that is essential in our setting. Finally, there is no formal cryptographic treatment of the primitive itself.

In [CCN+23], Cerulli, Connolly, Neven, Preiss, and Shoup show how a static committee can maintain a long-term master secret key that derives decryption keys corresponding to public identities or labels on demand. The authors sketch a dead man's switch application where a testator encrypts secret under a BLS signature, and stores it in a dApp that mostly does an application level monitoring of alive signals — without any cryptographic abstraction of death.

In [AFP25], Agarwal, Fernando and Pinkas propose batched IBE, which allows encrypting messages to specific identities and batch labels, where a single succinct decryption key is issued for an arbitrary subset of entities in a given batch. However, the label is known at the time of encryption, there is no nominee privacy, and the construction gives no interface for alive signals, which we cannot do without.

Küpçü and Lysyanskaya [KL12] study optimistic fair exchange, where two parties exchange digital items and invoke an arbitrator to resolve any dispute. While this might resemble our setting in the sense that both involve conditional release of digital assets, the analogy breaks down in a fundamental way: fair exchange does not stop both parties to act at the time of exchange, whereas our testator is absent at the moment of secret release and hence cannot act. Therefore, a committee must play a strictly stronger role. In other words, our requirements fall strictly outside the scope of fair exchange.

Future Work. We now highlight several directions for future research.

- To begin, our construction allows an authorized nominee to independently recover a secret upon testator's death. In many practical scenarios, however, unilateral recovery might not be desirable — a corporate secret held by an executive may require a quorum of board members to participate in a t-out-of-n anonymous recovery policy, upon the executive's death. Hence a natural extension is to ask whether our framework can support threshold recovery among nominees while preserving nominee privacy.

- Our current design assumes a fixed nominee set. Nevertheless, in practice, nominee relationships may evolve over time — new nominees may be added or revoked, or their public keys may be rotated. Supporting "dynamic nominee sets" without re-initializing the long-lived secret is an interesting direction for future work.

- DC-ECPSS captures a general cryptographic notion — dealer controlled secret management — that may be of interest beyond a dead man's switch application. Natural settings where this abstraction arises include digital asset inheritance, recovery of cryptocurrency wallet keys, archival decryption, among others. Exploring the applications of DC-ECPSS and instantiating the primitive in each of this settings, is left as a future work.

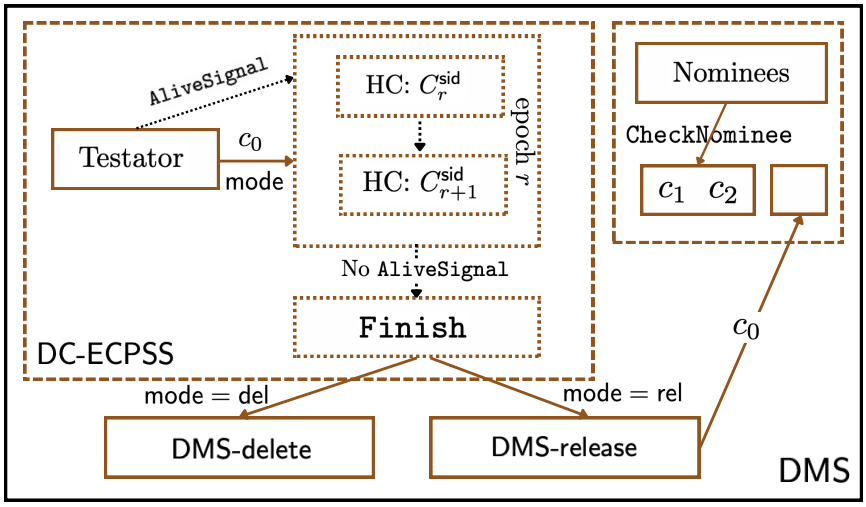


Figure 1: Illustration of the DMS architecture: DC-ECPSS maintains c_0 across epochs, while (c_1, c_2) is broadcast by the testator at the setup phase. Nominees can locally check their inclusion using `CheckNominee`. When there is no `AliveSignal`, c_0 is released to nominees in the release mode, while c_0 is erased in the delete mode; "HC" denotes holding committees.

- Our work realizes DMS through evolving committees and PSS. An interesting open question is whether comparable guarantees can be achieved through different cryptographic paradigms.

2 Preliminaries

2.1 Notation

We write λ for the security parameter and 1^λ for its unary encoding; PPT abbreviates probabilistic polynomial-time, and $\mathsf{negl}(\lambda)$ denotes a negligible function. For $n \in \mathbb{N}$ we write $[n] := \{1, \ldots, n\}$, and for a set $\mathcal{X}$ we write $2^{\mathcal{X}}$ for its power set. We use $|\mathcal{X}|$ for the cardinality of a set and $|x|$ for the bit-length of a string x; ε denotes the empty string and $\perp$ a distinguished failure ("no value") symbol.

We write $x \leftarrow\!\!\$\ \mathcal{X}$ for sampling x uniformly from a finite set $\mathcal{X}$, and $y \leftarrow A(x)$ for assigning to y the output of a (possibly randomized) algorithm A on input x. To make randomness explicit we write $y \leftarrow A(x; \rho)$ for running A on input x with random tape ρ, and $(y, \rho) \leftarrow A(x)$ for saving the random tape used into ρ; with ρ fixed, A is deterministic, which lets a relation or a proof refer to the exact coins an algorithm used. For a probabilistic experiment we write $\Pr[\, E : \mathsf{exp} \,]$ for the probability of event E after running exp. Two distribution ensembles X, Y are *identically distributed*, written $X \equiv Y$, if they coincide, and *computationally indistinguishable*, written $X \approx_c Y$, if no PPT distinguisher separates them with non-negligible advantage.

2.2 Secret Sharing Schemes

Secret sharing schemes allow a party called the dealer to split a secret into shares, which will then be distributed among n parties, such that only authorized sets of said shares allow reconstruction of the secret, i.e., only an authorized set of the n parties can recover the secret.

Definition 1 (Secret Sharing Schemes) *A secret sharing scheme with a secret domain K for an access structure $\mathbb{A}$ on m parties is given by a tuple of PPT algorithm $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Recon})$ with the following syntax.*

$(\mathsf{sh}_1, \ldots, \mathsf{sh}_m) \leftarrow \mathsf{Share}(s)$**:** *Given a secret $s \in K$, this algorithm outputs a set of shares $(\mathsf{sh}_1, \ldots, \mathsf{sh}_m)$.*

$s' \leftarrow \mathsf{Recon}(\{\mathsf{sh}_i\}_{i\in S})$**:** *Given a set of shares* $\{\mathsf{sh}_i\}_{i\in S}$*, this algorithm outputs a secret* $s' \in K$*.*

Access structures encode which sets of parties should be able to reconstruct the shared secret.

Definition 2 ((Monotone) Access Structures) *A (monotone) access structure on* m *parties is given by a set* $\mathbb{A} \subseteq 2^{[m]}$ *of sets of parties, where* $B \in \mathbb{A}$ *and* $B \subseteq C$ *implies that* $C \in \mathbb{A}$ *as well. We call the sets in* $\mathbb{A}$ *authorized and the sets not in* $\mathbb{A}$ *unauthorized.*

A relevant example of a (monotone) access structure is the t-out-of-n threshold access structure, denoted by $\mathbb{A}_{\mathsf{th}(t,m)}$ which contains all subsets $A \subseteq 2^{[m]}$ with size $|A| \geq t$. We are mainly interested in the nested t-out-of-m threshold access structure, denoted by $\mathbb{A}_{\mathsf{Nstd}(t,m)} \subseteq 2^{[m^2]}$. $\mathbb{A}_{\mathsf{Nstd}(t,m)}$ contains a set $S \subseteq [m^2]$ if and only if S contains at least t parties from at least t blocks of parties $\{i \cdot m, \ldots, (i+1) \cdot m - 1\}$ for $i \in \{0, 1, \ldots, m-1\}$. An authorized example is the set $\bigcup_{i\in\{0,1,\ldots,t-1\}} \{i \cdot m, \ldots, i \cdot m + t - 1\}$ which contains t parties from each of the first t blocks of m parties.

A secret sharing scheme should satisfy the properties of *correctness*, *privacy*, and *share simulatability*. Correctness ensures that any authorized set can reconstruct the correct secret.

Definition 3 (Correctness) *We call* $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Recon})$ correct *if for every authorized set* $S \in \mathbb{A}$ *and secret* $s \in K$*, it holds that*

$$\Pr\left[\mathsf{Recon}(\{\mathsf{sh}_i\}_{i\in S}) = s \mid (\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(s)\right] = 1.$$

Privacy formalizes the notion that any unauthorized set of shares leaks no information about the secret.

Definition 4 (Privacy) *We say that* $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Recon})$ *has* privacy *if for every unauthorized set* $S \notin \mathbb{A}$*, for every two secrets* $s_1, s_2 \in K$*, and for* $(\mathsf{sh}_1, \ldots, \mathsf{sh}_m) \leftarrow \mathsf{Share}(s_1)$ *and* $(\mathsf{sh}'_1, \ldots, \mathsf{sh}'_m) \leftarrow \mathsf{Share}(s_2)$*, the distributions of* $\{\mathsf{sh}_i\}_{i\in S}$ *and* $\{\mathsf{sh}'_i\}_{i\in S}$ *are identical.*

Share simulatability, [KRY22], requires the existence of an efficient share simulation algorithm $\mathsf{ShareSim}$ that takes in an unauthorized set of shares for some secret s_1 and extends it to a full set of shares for a different target secret s_2.

Definition 5 (Share Simulatability) *We say that* $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Recon})$ *is* share simulatable *if there exists a PPT algorithm* $\mathsf{ShareSim}$ *such that for every unauthorized set* $S \notin \mathbb{A}$*, for every pair of secrets* $s_1, s_2 \in K$*, for* $(\mathsf{sh}_1, \ldots, \mathsf{sh}_m) \leftarrow \mathsf{Share}(s_1)$*, and* $\{\mathsf{sh}'_i\} \leftarrow \mathsf{ShareSim}(\{\mathsf{sh}_i\}_{i\in S}, s_2)$*, it holds that* $\{\mathsf{sh}_i\}_{i\in S} \cup \{\mathsf{sh}'_i\}_{i\in[m]\setminus S}$ *is distributed identically to a fresh sharing output by* $\mathsf{Share}(s_2)$*.*

2.3 Re-sharing

Following [BGG+20], we require secret sharing schemes with special re-sharing functionality. Specifically, we augment secret sharing schemes with two algorithms, $\mathsf{Handover}$, and $\mathsf{Takeover}$. Assume that we shared a secret s with m parties such that each party P_i holds share sh_i. The parties would now like to re-share their secret with another, possibly distinct, set of m parties, denote these by $P'_1, \ldots, P'_m$. To do this, each party P_i runs the algorithm $\mathsf{Handover}$ on its share sh_i, generating a set of re-sharing shares $\mathsf{rsh}_{i,1}, \ldots, \mathsf{rsh}_{i,m}$ which are then shared with the next set of parties. Specifically, P'_i then receives re-sharing shares $\mathsf{rsh}_{1,i}, \ldots, \mathsf{rsh}_{m,i}$ one from each party $P_1, \ldots, P_m$. By running the $\mathsf{Takeover}$ algorithm on their re-sharing shares, the parties $P'_1, \ldots, P'_m$ can then recover a fresh sharing of the same secret s.

Definition 6 (Secret Sharing Schemes with Re-sharing) *Let* $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Recon})$ *be a secret sharing scheme. We can extend* SSS *to a* secret sharing scheme with re-sharing *by providing two PPT algorithms* $\mathsf{Handover}$ *and* $\mathsf{Takeover}$ *with the following syntax.*

$(\mathsf{rsh}_1, \ldots, \mathsf{rsh}_n) \leftarrow \mathsf{Handover}(\mathsf{sh})$**:** *Given a a share* sh*, this algorithm outputs a set of re-sharing shares* $(\mathsf{rsh}_1, \ldots, \mathsf{rsh}_m)$*.*

$\mathsf{sh} \leftarrow \mathsf{Takeover}(\{\mathsf{rsh}_i\}_{i\in S})$**:** *Given a set* $\{\mathsf{rsh}_i\}_{i\in S}$ *of re-sharing shares, this algorithm outputs a share* sh*.*

For brevity we may denote this combination using the tuple $(\mathsf{Share}, \mathsf{Recon}, \mathsf{Handover}, \mathsf{Takeover})$.

These additional algorithms should satisfy the properties of *re-sharing correctness* and *re-sharing privacy*.

Re-sharing correctness formalizes the intuition, that as long as the parties that run $\mathsf{Handover}$ honestly form an authorized set, $\mathsf{Takeover}$ will result in a fresh sharing of the original secret.

Definition 7 (Re-sharing Correctness) *We say that* $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Recon}, \mathsf{Handover}, \mathsf{Takeover})$ *has re-sharing correctness if for every authorized set* $S \in \mathbb{A}$ *and secret* $s \in K$*, it holds that after running* $(\mathsf{sh}_1, \ldots, \mathsf{sh}_m) \leftarrow \mathsf{Share}(s)$ *and* $(\mathsf{rsh}_{i,1}, \ldots, \mathsf{rsh}_{i,m}) \leftarrow \mathsf{Handover}(\mathsf{sh}_i)$ $\forall i \in S$*, the shares* $(\mathsf{sh}'_1, \ldots, \mathsf{sh}'_m)$ *constructed via* $\mathsf{sh}'_j \leftarrow \mathsf{Takeover}(\{\mathsf{rsh}_{i,j}\}_{i \in S})$ *for all* $j \in [n]$ *are distributed identically to a fresh sharing output by* $\mathsf{Share}(s)$.

Re-sharing privacy is essentially privacy for the secret sharing scheme rephrased for the $\mathsf{Handover}$ and $\mathsf{Takeover}$ algorithms, implying that a set of re-sharing shares is independent of the share sh.

Definition 8 (Re-sharing Privacy) *We say that* $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Recon}, \mathsf{Handover}, \mathsf{Takeover})$ *has re-sharing privacy if for every unauthorized set* $S \notin \mathbb{A}$*, for every two shares* $\mathsf{sh}, \mathsf{sh}'$*, and for* $(\mathsf{rsh}_1, \ldots, \mathsf{rsh}_m) \leftarrow \mathsf{Handover}(\mathsf{sh})$ *and* $(\mathsf{rsh}'_1, \ldots, \mathsf{rsh}'_m) \leftarrow \mathsf{Handover}(\mathsf{sh}')$*, the distributions of* $\{\mathsf{rsh}_i\}_{i \in S}$ *and* $\{\mathsf{rsh}'_i\}_{i \in S}$ *are identical.*

2.4 Linear Secret Sharing Schemes

One way to instantiate secret sharing schemes with re-sharing is using linear secret sharing schemes. Let $q \in \mathbb{N}$ be a prime and $\mathbb{A}$ a (monotone) access structure on m parties. A (monotone) linear secret sharing scheme (LSSS) $\mathsf{LSSS} = (\mathbf{M}, \rho)$ over $\mathbb{Z}_q$ for $\mathbb{A}$ can be completely described by a *share-generating matrix* $\mathbf{M} \in \mathbb{Z}_q^{\ell \times d}$, and a *label function* $\rho : [\ell] \to [m]$ that labels the rows of $\mathbf{M}$ with a party index from $[m]$. In this work we will only work with LSSS where the label function ρ is the identity function (such that also $\ell = m$), which means we can represent an LSSS only by its sharing matrix. For some $i \in [\ell]$ we write $\mathbf{m}_i$ to denote the i-th row of $\mathbf{M}$. This then implies the following secret sharing scheme with secret domain $\mathbb{Z}_q$:

$\mathsf{Share}(s \in \mathbb{Z}_q)$**:** Sample $v_2, \ldots, v_d \leftarrow\!\!\$\, \mathbb{Z}_q$ and let $\mathbf{v} = [s, v_2, \ldots, v_d]^\top \in \mathbb{Z}_q^d$. Finally, compute $\mathbf{z} = \mathbf{M} \cdot \mathbf{v} \in \mathbb{Z}_q^{m \times 1}$, and let each share be $\mathsf{sh}_i \leftarrow z_i$. Output $\mathsf{sh}_1, \ldots, \mathsf{sh}_m$.

$\mathsf{Recon}(\{\mathsf{sh}_i\}_{i \in S})$**:** Compute $r_i \in \mathbb{Z}_q$ for all $i \in S$ such that $\sum_{i \in S} r_i \mathbf{m}_i = [1, 0, \ldots, 0] \in \mathbb{Z}_q^d$, and output $\sum_{i \in S} r_i \mathsf{sh}_i = s'$.

2.5 Re-sharing for Linear Secret Sharing Schemes

Benhamouda et al. [BGG+20] show how re-sharing can be realized for the Shamir secret sharing scheme specifically. Their approach generalizes easily to LSSS, as we show now. Let $\mathsf{LSSS} = \mathbf{M} \in \mathbb{Z}_q^{m \times d}$ be an LSSS with implied sharing and reconstruction algorithms Share and Recon which satisfy correctness, privacy, and share simulatability. Then the $\mathsf{Handover}$ and $\mathsf{Takeover}$ algorithms work as follows.

$(\mathsf{rsh}_1, \ldots, \mathsf{rsh}_n) \leftarrow \mathsf{Handover}(\mathsf{sh})$**:** Output the result of running $\mathsf{Share}(\mathsf{sh})$.

$\mathsf{sh} \leftarrow \mathsf{Takeover}(\{\mathsf{rsh}_i\}_{i \in S})$**:** Output the result of running $\mathsf{Recon}(\{\mathsf{rsh}_i\}_{i \in S})$.

The following Lemma shows that the above re-sharing algorithms satisfy re-sharing correctness and re-sharing privacy.

Lemma 1 Let $\mathsf{LSSS} = (\mathsf{Share}, \mathsf{Recon})$ be an LSSS that satisfies correctness (Definition 3, privacy (Definition 4), and share simulatability (Definition 5). Then the LSSS formed by augmenting LSSS with the above re-sharing algorithms $\mathsf{Handover}, \mathsf{Takeover}$ satisfies re-sharing correctness (Definition 7), and re-sharing privacy (Definition 8).

Proof. Re-sharing privacy follows immediately from privacy of the underlying LSSS since $\mathsf{Handover}$ just runs Share. It remains to show re-sharing correctness. By definition of Share, each share sh_i for $i \in [m]$ is computed as

$$\mathsf{sh}_i \leftarrow \mathbf{m}_i \cdot \mathbf{v},$$

where $\mathbf{v} = [s, v_2, \ldots, v_d]^\top \in \mathbb{Z}_q^d$ with $v_2, \ldots, v_d \leftarrow\!\!\$\ \mathbb{Z}_q$. Then $\mathsf{Handover}(\mathsf{sh}_i)$ computes the re-sharing shares $\mathsf{rsh}_{i,j}$ for $j \in [m]$ by first sampling $v_{i,2}, \ldots, v_{i,d} \leftarrow\!\!\$\ \mathbb{Z}_q$ and then setting

$$\mathsf{rsh}_{i,j} \leftarrow \mathsf{Share}(\mathsf{sh}_i) = \mathbf{m}_j \cdot \begin{bmatrix} \mathsf{sh}_i \\ v_{i,2} \\ \vdots \\ v_{i,d} \end{bmatrix} = \mathbf{m}_j \cdot \begin{bmatrix} \mathbf{m}_i \cdot \begin{bmatrix} s \\ v_2 \\ \vdots \\ v_d \end{bmatrix} \\ v_{i,2} \\ \vdots \\ v_{i,d} \end{bmatrix}.$$

Then, for some authorized set $S \in \mathbb{A}$, denote the corresponding reconstruction vector by $\mathbf{r}(\mathbf{M}, S)$, which implies that $\mathsf{Takeover}(\{\mathsf{rsh}_{i,j}\}_{i \in S})$ computes its output share sh'_j via

$$\mathsf{sh}'_j \leftarrow \sum_{i \in S} \mathbf{r}(\mathbf{M}, S)_i \cdot \mathsf{rsh}_{i,j} = \sum_{i \in S} \mathbf{r}(\mathbf{M}, S)_i \cdot \mathbf{m}_j \cdot \begin{bmatrix} \mathbf{m}_i \cdot \begin{bmatrix} s \\ v_2 \\ \vdots \\ v_d \end{bmatrix} \\ v_{i,2} \\ \vdots \\ v_{i,d} \end{bmatrix}$$

$$= \mathbf{m}_j \cdot \begin{bmatrix} \sum_{i \in S} \mathbf{r}(\mathbf{M}, S)_i \cdot \mathbf{m}_i \cdot \begin{bmatrix} s \\ v_2 \\ \vdots \\ v_d \end{bmatrix} \\ \sum_{i \in S} \mathbf{r}(\mathbf{M}, S)_i \cdot v_{i,2} \\ \vdots \\ \sum_{i \in S} \mathbf{r}(\mathbf{M}, S)_i \cdot v_{i,d} \end{bmatrix} = \mathbf{m}_j \cdot \begin{bmatrix} s \\ \sum_{i \in S} \mathbf{r}(\mathbf{M}, S)_i \cdot v_{i,2} \\ \vdots \\ \sum_{i \in S} \mathbf{r}(\mathbf{M}, S)_i \cdot v_{i,d} \end{bmatrix},$$

where the second-to-last inequality follows by linearity of the inner product, and the last equality follows by definition and existence of the reconstruction vector $\mathbf{r}(\mathbf{M}, S)$ since S is an authorized set and LSSS is assumed to be correct. As long as the S used for each call to $\mathsf{Takeover}$, i.e., each sh'_j is always the same, we can write

$$\begin{bmatrix} \mathsf{sh}'_1 \\ \vdots \\ \mathsf{sh}'_n \end{bmatrix} = \mathbf{M} \cdot \begin{bmatrix} s \\ \sum_{i \in S} \mathbf{r}(\mathbf{M}, S)_i \cdot v_{i,2} \\ \vdots \\ \sum_{i \in S} \mathbf{r}(\mathbf{M}, S)_i \cdot v_{i,d} \end{bmatrix}.$$

Since each of $v_{i,2}, \ldots, v_{i,d}$ is uniformly sampled from $\mathbb{Z}_q$, the entries $\sum_{i \in S} \mathbf{r}(\mathbf{M}, S)_i \cdot v_{i,k}$, for $k \in \{2, \ldots, d\}$, look uniform as well, implying that $(\mathsf{sh}'_1, \ldots, \mathsf{sh}'_m)$ is identically distributed to a fresh sharing of the secret s, as desired. □

2.6 Nested Shamir Secret Sharing

Let $q > m$ be a prime. The Shamir secret sharing scheme [Sha79] for $\mathbb{A}_{\mathsf{th}(t,m)}$ can be viewed as an LSSS over $\mathbb{Z}_q$ using sharing matrix $\mathbf{M}_{\mathsf{th}(t,m)} \in \mathbb{Z}_q^{n \times t}$ as shown in, e.g., [Bei96], Example 4.4.

We are mainly interested in a form of nested Shamir secret sharing for the previously defined access structure $\mathbb{A}_{\mathsf{Nstd}(t,m)}$, following [ACD+25] albeit using the standard Shamir secret sharing over the field $\mathbb{Z}_q$ instead of threshold schemes over the integers $\mathbb{Z}$ as they do. Let $\mathbf{M}_{\mathsf{th}(t,m)} \in \mathbb{Z}_q^{m \times t}$ be the Shamir share-generating matrix. Then the LSSS that realizes $\mathbb{A}_{\mathsf{Nstd}(t,m)}$ is given by $\mathbf{M}_{\mathsf{Nstd}(t,m)} := \mathbf{M}_{\mathsf{th}(t,m)} \otimes \mathbf{M}_{\mathsf{th}(t,m)} \in \mathbb{Z}_q^{m^2 \times t^2}$, as pointed out in [ACD+25] (here $\otimes$ denotes the tensor product). They argue that this satisfies correctness and privacy, the following Lemma shows that it also satisfies share simulatability. Its proof follows by privacy and linearity of LSSS.

Lemma 2 Let $\mathsf{LSSS} = \mathbf{M}$ be an LSSS that satisfies privacy. Then LSSS also satisfies share simulatability.

Proof. We first introduce some additional notation: For some subset of parties $S \subseteq [m]$, we write $\mathbf{M}_S$ to denote the matrix consisting of the rows $\{\mathbf{m}_i\}_{i \in S}$ of $\mathbf{M}$.

By definition of Share, the shares $\{\mathsf{sh}_i\}_{i \in S}$ are computed via $\mathbf{M}_S \mathbf{v}$ where $v_1 = s_1$ and the other entries of $\mathbf{v}$ are uniformly sampled. Then $\mathsf{ShareSim}(\{\mathsf{sh}_i\}_{i \in S}, s_2)$ works by computing the space of possible $\mathbf{v}'$, with $v'_1 = s_2$, such that $\mathbf{M}_S \mathbf{v} = \mathbf{M}_S \mathbf{v}'$, i.e., by Gaussian elimination (this space is nonempty due to privacy of the LSSS). Note that this does not require knowledge of $\mathbf{v}$. It then picks such a $\mathbf{v}'$ uniformly at random and outputs the remaining shares $\{\mathsf{sh}'_i\}_{i \in [n] \setminus S}$ via $\mathbf{M}_{[n] \setminus S} \mathbf{v}'$.

We now analyze the distribution of the output $\mathbf{M}_{[n] \setminus S} \mathbf{v}'$ of $\mathsf{ShareSim}$ combined with the existing shares $\mathbf{M}_S \mathbf{v}$, as required for share simulatability. We first analyze the distribution of $\mathbf{M}_{[n] \setminus S} \mathbf{v}'$ and then the distribution of $\mathbf{M}_S \mathbf{v}$ conditioned on said output. Using privacy of LSSS, we can equivalently replace the $\mathbf{M}_S \mathbf{v}$ given to $\mathsf{ShareSim}$ with $\mathbf{M}_S \mathbf{v}''$ where $v''_1 = s_2$ and the remaining entries are uniform. Then, clearly the entries of $\mathbf{v}'$ other than the first (which equals s_2) are also uniformly distributed, showing that the output of $\mathsf{ShareSim}$ is identically distributed as a subset of shares $[m] \setminus S$ of a fresh sharing of s_2. Now, conditioned on the output of $\mathsf{ShareSim}$, definition of $\mathsf{ShareSim}$ ensures that $\mathbf{M}_S \mathbf{v} = \mathbf{M}_S \mathbf{v}'$, so the overall distribution is consistent with the internal randomness $\mathbf{v}'$, as desired. □

Combining the facts from [ACD+25], Lemma 1, and Lemma 2, we obtain the following Lemma, showing that the LSSS given by $\mathbf{M}_{\mathsf{Nstd}(t,m)}$ satisfies all the desired properties.

Lemma 3 Let $\mathsf{LSSS} = \mathbf{M}_{\mathsf{Nstd}(t,m)}$. Then LSSS, for access structure $\mathbb{A}_{\mathsf{Nstd}(t,m)}$, augmented with the previously defined $\mathsf{Handover}$ and $\mathsf{Takeover}$ algorithms satisfies correctness, privacy, share simulatability, re-sharing correctness, and re-sharing privacy.

3 Dealer-Controlled EPSS

3.1 Modeling DC-ECPSS

We now introduce the formal model for $\mathsf{DC\text{-}ECPSS}$ in the form of an ideal functionality $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$, which conceptually models dealer-controlled maintenance of a long-term secret. We regard the model in Figure 2. The functionality $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ runs on an input security parameter λ with a universe of N parties $\mathcal{U} = \{P_1, \ldots, P_N\}$, a simulator $\mathcal{S}$ and an environment $\mathcal{Z}$. The functionality operates in a synchronous distributed model: honest parties are activated round by round by $\mathcal{Z}$ and, when instructed, consensually invoke `Ping` in the same round. All persistent states are maintained internally by $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$. The functionality is parameterized by a mode, which specifies the post-processing to be applied to the secret when a dealer fails to respond within an epoch.

The $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ functionality captures *four* main interfaces. A dealer D is modeled as a party P_j that invokes `StoreSecret`, and the functionality records the identity $D = P_j$ which it uses for subsequent checks. Upon receiving input $(\mathsf{sid}, \texttt{StoreSecret}, s, w, \mathcal{R}, \mathsf{mode})$ from P_j, the functionality first checks if a secret has already been stored for sid, where sid is a session identifier — if so, it aborts. Otherwise, it evaluates the well-formedness of the secret by checking if $\mathcal{R}(s, w) = 1$. If both checks pass, the functionality records $(\mathsf{sid}, s, \mathsf{mode}, D := P_j)$ internally, and notifies $\mathcal{S}$ and all parties that a secret has been stored.

A second interface called `Ping` implements an advancement of an epoch. Elaborating on this, the environment $\mathcal{Z}$ activates all honest parties at the beginning of each round. The consensual `Ping` input represents

a global agreement among all honest parties that an epoch has elapsed. Upon receiving $(\mathsf{sid}, \texttt{Ping})$ from all honest parties at the same round, it inspects two internal flags: Pinged, which records whether an alive signal was issued since the last response, and $\mathsf{DResponse}$, which records whether D has responded to that query. If so, i.e. $\mathsf{Pinged} = 1$ and $\mathsf{DResponse} = 0$, indicating that an alive signal was issued but went unanswered, the functionality concludes that D has failed to respond, and transitions to the terminal phase — it sets $\mathsf{Released} = 1$, delivers $(\mathsf{sid}, \mathsf{mode}(s))$ to every honest party and to $\mathcal{S}$, and aborts the current activation. Otherwise, it resets $\mathsf{Pinged} = 1$ and $\mathsf{DResponse} = 0$, and notifies D with $(\mathsf{sid}, \texttt{Pinged})$.

The third interface `DealerOK` is a dealer's response to an alive signal. Upon receiving $(\mathsf{sid}, \texttt{DealerOK})$ from D, the functionality checks if an unreleased secret exists for sid. It then ensures that $P_j = D$, ensuring that the only party who originally stored the secret — recorded as D at `StoreSecret` — may submit responses to alive signals. The functionality next checks if $\mathsf{Pinged} = 1$, ensuring that an alive signal has been issued and a response is expected. Finally, it verifies that $\mathsf{DResponse} = 0$, ensuring that a dealer has not already responded to the current query. If any of these conditions fail, the functionality sends $(\mathsf{sid}, \perp)$ and aborts the current activation. Else, it sets $\mathsf{DResponse} = 1$ internally and notifies $\mathcal{S}$, while no input is delivered to other parties.

The fourth interface `ReadSecret` allows any party to retrieve the post-processed secret after the terminal phase has been reached. Upon input $(\mathsf{sid}, \texttt{ReadSecret})$ from P_j, the functionality checks if $\mathsf{Released} = 1$; if not, it sends $(\mathsf{sid}, \perp)$ to P_j and aborts. If $\mathsf{Released} = 1$, then the functionality sends $(\mathsf{sid}, \mathsf{mode}(s))$ to P_j. In $\mathsf{release}$ mode, $\mathsf{mode}(s) = s$, while in delete mode, $\mathsf{mode}(s) = \perp$. We note that $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ imposes no restrictions on who may call `ReadSecret` — authorization to nominee is enforced at the DMS level, not at the $\mathsf{DC\text{-}ECPSS}$ level.

3.2 The protocol $\pi_{\mathsf{DC\text{-}ECPSS}}$

We now lay the details of our $\mathsf{DC\text{-}ECPSS}$ construction $\pi_{\mathsf{DC\text{-}ECPSS}}$ in the abstract world, where the protocol is run by roles, following the YOSO formalization [GHK+21]. We remind the readers that roles are simply an abstraction of entities, taking part in the protocol execution and communicating with other roles. Each role performs its action and erases its state. The mapping of roles to physical machines is handled by a role assignment mechanism (see discussion in Section 3.5).

Let S be the maximum possible number of sessions (secrets), i.e. $S = \max|\mathsf{SID}|$. Let T be the maximum lifespan of a secret. Let m^2 be the size of a holding committee, and let $N = m^2 \cdot T \cdot S$. Then, the protocol is run by a universe of roles $\{D_1, \ldots, D_S; R_1, \ldots R_n\}$, where the roles R_i are divided into committees, as follows: first m^2T roles $R_1, \ldots, R_{m^2T}$ form T holding committees for $\mathsf{sid} = 1$, next m^2T roles form T holding committees for $\mathsf{sid} = 2$, and so on.
We denote by C_i^{sid} the i-th holding committee for session sid, and by $C_i^{\mathsf{sid}}[j]$, the role of its j-th member. Each party maintains a local round counter $r_{\mathsf{sid}} \in \mathbb{N}$ initialized to 1, upon first observing sid. Protocol $\pi_{\mathsf{DC\text{-}ECPSS}}$ is built over a secret sharing scheme $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Recon})$ with re-sharing algorithms $\mathsf{Handover}$ and $\mathsf{Takeover}$, and a commitment scheme Com. We note that $\mathsf{Handover}$ and $\mathsf{Takeover}$ operate at a share level — no node ever reconstructs s.

We formally introduce the protocol $\Pi_{\mathsf{DC\text{-}ECPSS}}$ in Figures 3 and 4 in a $(\mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{SPP}}, \mathcal{F}_{\mathsf{NIZK}})$-hybrid world. Here $\mathcal{F}_{\mathsf{BC}}$ (Figure 8) is an authenticated broadcast, $\mathcal{F}_{\mathsf{SPP}}$ (Figure 9) a secure point-to-point channel, and $\mathcal{F}_{\mathsf{NIZK}}$ (Figure 10) a UC non-interactive zero-knowledge functionality for the relations $\mathcal{R}_{\mathcal{R}}^{\mathsf{Share}}$ and $\mathcal{R}^{\mathsf{Reshare}}$ defined below; Com (Appendix A.1) is a statistically hiding, computationally binding commitment scheme.

Functionality $\mathcal{F}_{\text{DC-ECPSS}}$

Notation. We use the following semantic instructions:

- **Break**: abort the current activation without output.
- **Parse**: extract well-formed messages matching a given pattern.
- **Retrieve**: look up a stored record by key; if absent, treat as $\perp$.

On input $(\mathsf{sid}, \texttt{Init})$ from a party P:

1. If P is already initialized for sid, **Break**.
2. Mark P as initialized for sid.
3. **Send** $(\mathsf{sid}, \texttt{Initialized}, P)$ to $\mathcal{S}$.

On input $(\mathsf{sid}, \texttt{StoreSecret}, s, w, \mathcal{R}, \mathsf{mode})$ from a party P_j:

Here $\mathsf{mode} \in \{\mathsf{release}, \mathsf{delete}\}$ *specifies the post-processing applied to the secret s before it is released.*

1. If there is an entry for sid, **Send** $(\mathsf{sid}, \perp)$ to P_j and **Break**
2. If $\mathcal{R}(s, w) \neq 1$, **Send** $(\mathsf{sid}, \perp)$ to P_j and **Break**
3. **Store** $(\mathsf{sid}, s, \mathsf{mode}, D = P_j, \mathsf{Pinged} = 0, \mathsf{DResponse} = 0, \mathsf{Released} = 0)$
4. **Send** $(\mathsf{sid}, P_j, \texttt{Stored}, \mathsf{mode}, \mathcal{R})$ to $\mathcal{S}$ and to all parties.

On input $(\mathsf{sid}, \texttt{Ping})$ from every honest party in the same round:

In the synchronous model, $\mathcal{Z}$ schedules a round and activates every honest party with input $(\mathsf{sid}, \texttt{Ping})$. $\mathcal{F}_{\text{DC-ECPSS}}$ *processes the ping only once all honest parties have consensually submitted it within that round; otherwise the activation is ignored.*

1. **Retrieve** $(\mathsf{sid}, s, \mathsf{mode}, D, \mathsf{Pinged}, \mathsf{DResponse}, \mathsf{Released})$
   No secret with such sid *or it was released*
2. If there is no entry for sid or $\mathsf{Released} = 1$, **Send** $(\mathsf{sid}, \perp)$ to every honest party and to $\mathcal{S}$, and **Break**
3. If $\mathsf{Pinged} = 1$ and $\mathsf{DResponse} = 0$:
   3.1 **Replace** entry for sid with $(\mathsf{sid}, s, \mathsf{mode}, \perp, \perp, \perp, 1)$
   3.2 **Send** $(\mathsf{sid}, \ \mathsf{mode}(s))$ to every honest party and to $\mathcal{S}$
   3.3 **Break**.
   Otherwise: first ping, or dealer responded to the previous one
4. Set $\mathsf{Pinged} \leftarrow 1$ and $\mathsf{DResponse} \leftarrow 0$ for sid
5. **Send** $(\mathsf{sid}, \ \texttt{Pinged})$ to D and to $\mathcal{S}$

On input $(\mathsf{sid}, \texttt{DealerOK})$ from P_j:

1. **Retrieve** $(\mathsf{sid}, s, \mathsf{mode}, D, \mathsf{Pinged}, \mathsf{DResponse}, \mathsf{Released})$
   Check that an unreleased secret with sid *exists, that OK comes from the correct dealer, and that no OK was sent for the current ping*
2. If there is no entry for sid or $\mathsf{Released} = 1$ or $D \neq P_j$ or $\mathsf{Pinged} = 0$ or $\mathsf{DResponse} = 1$, then **Send** $(\mathsf{sid}, \perp)$ to P_j and **Break**
   Otherwise this is a legitimate reply from the dealer
3. Set $\mathsf{DResponse} \leftarrow 1$ for sid
4. **Send** $(\mathsf{sid}, \ \texttt{DealerOK})$ to $\mathcal{S}$

On input $(\mathsf{sid}, \texttt{ReadSecret})$ from P_j:

1. **Retrieve** $(\mathsf{sid}, s, \mathsf{mode}, D, \mathsf{Pinged}, \mathsf{DResponse}, \mathsf{Released})$
2. If $\mathsf{Released} = 0$, then **Send** $(\mathsf{sid}, \perp)$ to P_j and **Break**
3. **Send** $(\mathsf{sid}, \ \mathsf{mode}(s))$ to P_j

Figure 2: Ideal Functionality for a DC-ECPSS scheme

NIZK Relations. For a sid, committee index r, and committee members $\{C_r^{\mathsf{sid}}[k]\}_k$:

$$\begin{aligned}
\mathcal{R}_{\mathcal{R}}^{\mathsf{Share}}\big(\, x,\, w \,\big) : x = (\{c_k\}_{k\in[m^2]}),\ w = \big(s,\, w',\, \{\mathsf{rsh}_k, \rho_k\}_k,\, \rho_{\mathsf{Share}}\big) \\
\text{s.t. } \mathcal{R}(s, w') = 1,\ \{\mathsf{rsh}_k\} = \mathsf{Share}(s, \mathsf{mode};\, \rho_{\mathsf{Share}}), \\
c_k = \mathsf{Com}(\mathsf{rsh}_k; \rho_k)\ \forall k. \\
\mathcal{R}^{\mathsf{Reshare}}\big(\, x,\, w \,\big) : x = \big(\{c_i^{\mathrm{in}}\}_{i\in\mathcal{I}}, \{c_k^{\mathrm{out}}\}_k\big), \\
w = \big(\{\mathsf{sh}_i, \rho_i^{\mathrm{in}}\}_{i\in\mathcal{I}},\, \mathsf{sh},\, \rho_{\mathsf{Takeover}},\, \{\mathsf{rsh}_k, \rho_k^{\mathrm{out}}\}_k,\, \rho_{\mathsf{Handover}}\big) \\
\text{s.t. } c_i^{\mathrm{in}} = \mathsf{Com}(\mathsf{sh}_i; \rho_i^{\mathrm{in}})\ \forall i \in \mathcal{I},\ c_k^{\mathrm{out}} = \mathsf{Com}(\mathsf{rsh}_k; \rho_k^{\mathrm{out}})\ \forall k, \\
\mathsf{sh} = \mathsf{Takeover}(\{\mathsf{sh}_i\}_{i\in\mathcal{I}};\, \rho_{\mathsf{Takeover}}), \\
\{\mathsf{rsh}_k\} = \mathsf{Handover}(\mathsf{sh};\, \rho_{\mathsf{Handover}}).
\end{aligned}$$

Design Rationale: The protocol uses $\mathcal{F}_{\mathsf{BC}}$ for all public state — commitments, NIZK proofs and dealer's alive signals — and $\mathcal{F}_{\mathsf{SPP}}$ for distribution of private shares. $\mathcal{F}_{\mathsf{SPP}}$ with commitments is used rather than PKE for modularity: the abstract protocol is kept independent of public key infrastructure, which is handled by the role assignment mechanism in the real world. Thus, the dealer control is lightweight — in the abstract world, the alive signal is a single broadcast $(\mathsf{sid}, \mathtt{DealerOK}, r)$ to the $\mathcal{F}_{\mathsf{BC}}$ per epoch, where authentication is implicit in the functionality.
We now discuss each interface of $\pi_{\mathsf{DC\text{-}ECPSS}}$.

Init: This interface is standard — it initializes a party's local state for session sid.

StoreSecret: The dealer role D splits a secret s into m^2 shares, one per role in the first holding committee. To allow verifiability, each share is committed using Com, and a NIZK proof is produced establishing that the committed values encode a well-formed secret satisfying $\mathcal{R}$, thus securing against a malicious dealer role who may register a fraudulent secret. The commitments and proofs are broadcast publicly via $\mathcal{F}_{\mathsf{BC}}$ so that any role can verify well-formedness, while the share openings are sent privately to the corresponding roles via $\mathcal{F}_{\mathsf{SPP}}$.

Ping: The `Ping` interface is activated for every role in the current holding committee C_r^{sid} at the start of epoch r. A role R that does not belong to the holding committee simply aborts without any output. For a role R that belongs C_r^{sid}, the activation proceeds in two phases: R first reads the broadcast and P2P messages from C_{r-1}^{sid} in round $r-1$ via $\mathcal{F}_{\mathsf{BC}}$ and $\mathcal{F}_{\mathsf{SPP}}$. In the first phase, R verifies the incoming share against public commitment — confirming that the share received privately is consistent with the commitment broadcast publicly, while the NIZK proof ensures that the committed shares constitute a valid re-sharing of the underlying secret. If a share does not pass the check, the corresponding roles are removed from the reconstruction set $\mathcal{I}$. If $\mathcal{I}$ falls below the reconstruction threshold, R breaks; otherwise, it computes its local share for the epoch.

DealerOK: This interface deals with the dealer's mechanism for sending alive signals across epochs. Concretely, the dealer role D retrieves the current round counter r_{sid}, broadcasts $(\mathsf{sid}, \mathtt{DealerOK}, r_{\mathsf{sid}})$ via $\mathcal{F}_{\mathsf{BC}}$ and increments the counter value — this constitutes the alive signal for epoch r. Upon observing this, the current holding committee proceeds with re-sharing. As one can see, the `DealerOK` signal is entirely decoupled from the secret — all it needs to do is issue a single authenticated broadcast, where authenticity is guaranteed by $\mathcal{F}_{\mathsf{BC}}$.

ReadSecret: This interface allows any role to recover the output from the terminal phase. Upon receiving $(\mathsf{sid}, \mathtt{ReadSecret})$ as input, the role receives all broadcasts for sid from $\mathcal{F}_{\mathsf{BC}}$, and looks for the latest round r^* in which at least τ distinct roles of $C_{r^\star}^{\mathsf{sid}}$ broadcast $(\mathsf{sid}, \mathtt{Finish}, r^\star, y_i)$ — indicating that the terminal phase is agreed upon by a threshold number of holding committee roles. If no such round exists, the terminal phase has not yet been reached, and the role breaks. Otherwise, it progresses with reconstruction and outputs (mode, s). In delete mode, reconstruction returns $\perp$, while in the $\mathsf{release}$ mode, the role is able to read the secret.

Protocol $\pi_{\mathsf{DC\text{-}ECPSS}}$ — Setup & Dealer Signals

Notation. Break: abort the current activation without output; **Parse**: extract well-formed messages matching a pattern; **P2P-send** m **to** $R \equiv$ input $(\mathsf{sid}, \mathtt{Send}_{\mathsf{SPP}}, R, m)$ to $\mathcal{F}_{\mathsf{SPP}}$; **Read-BC**$(R, r)$ / **Read-P2P**(r): read broadcast from R at round r, or incoming P2P messages addressed to self at round r.

Local state. For each sid, a party stores a round counter $r_{\mathsf{sid}} \in \mathbb{N}$, initialized to 1 upon first observing sid.

On input $(\mathsf{sid}, \mathtt{Init})$, a role R does:

1. Set $r_{\mathsf{sid}} \leftarrow 1$.

On input $(\mathsf{sid}, \mathtt{StoreSecret}, s, w, \mathcal{R}, \mathsf{mode})$, a role D does:

1. If D has already executed `StoreSecret` for sid, **Break**.
2. Sample fresh random tape ρ_{Share} and compute $(\mathsf{rsh}_1, \ldots, \mathsf{rsh}_{m^2}) \leftarrow \mathsf{Share}(s, \mathsf{mode}; \rho_{\mathsf{Share}})$.
3. For each $k \in [m^2]$, sample ρ_k and set $c_k \leftarrow \mathsf{Com}(\mathsf{rsh}_k; \rho_k)$.
4. Obtain $\pi_0 \leftarrow \mathcal{F}_{\mathsf{NIZK}}.\mathsf{Prove}\big(\mathcal{R}_{\mathcal{R}}^{\mathsf{Share}},\ \{c_k\}_k,\ (s,\ w,\ \{\mathsf{rsh}_k, \rho_k\}_k,\ \rho_{\mathsf{Share}})\big)$.
   Commitments and proof go public; openings go privately to first committee.
5. **Broadcast** $\big(\mathsf{sid}, \mathtt{Stored}, \mathsf{mode}, \mathcal{R}, \{c_k\}_{k\in[m^2]}, \pi_0\big)$.
6. For each $k \in [m^2]$: **P2P-send** $(\mathsf{sid}, \mathtt{Share}, r{=}1, \mathsf{rsh}_k, \rho_k)$ to $C_1^{\mathsf{sid}}[k]$.

On input $(\mathsf{sid}, \mathtt{DealerOK})$, a role D does:

1. Retrieve $r := r_{\mathsf{sid}}$.
2. **Broadcast** $(\mathsf{sid}, \mathtt{DealerOK}, r)$ via $\mathcal{F}_{\mathsf{BC}}$.
3. Set $r_{\mathsf{sid}} \leftarrow r{+}1$.

Figure 3: **Protocol** $\pi_{\mathsf{DC\text{-}ECPSS}}$: Setup interfaces (`Init`, `StoreSecret`) and the dealer's alive signal (`DealerOK`), in the $(\mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{SPP}}, \mathcal{F}_{\mathsf{NIZK}})$-hybrid model.

3.3 Security of $\pi_{\mathsf{DC\text{-}ECPSS}}$

Theorem 1 ($\pi_{\mathsf{DC\text{-}ECPSS}}$ UC-realizes $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$) *Let* SSS *be a secret-sharing scheme with re-sharing (Section 2.2) providing correctness, re-sharing correctness, privacy, re-sharing privacy, and share simulatability;* Com *a statistically hiding, computationally binding commitment scheme (Section A.1); and* $\mathcal{F}_{\mathsf{NIZK}}$ *a simulation-extractable NIZK (Figure 10) for* $\mathcal{R}_{\mathcal{R}}^{\mathsf{Share}}$ *and* $\mathcal{R}^{\mathsf{Reshare}}$*. Then* $\pi_{\mathsf{DC\text{-}ECPSS}}$ *UC-realizes* $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ *in the* $(\mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{SPP}}, \mathcal{F}_{\mathsf{NIZK}})$*-hybrid model against any static PPT adversary* $\mathcal{A}$*.*

Proof. (Proof sketch) Fix the static corruption set chosen by $\mathcal{A}$; in every committee the corrupt subset $\mathcal{B}_r$ stays below the reconstruction threshold. The simulator $\mathcal{S}$ runs $\mathcal{A}$ internally and moves from the real execution to the ideal one through a sequence of games. The first hops are syntactic or rest on the channel functionalities: $\mathcal{S}$ takes over the honest parties and the simulated $\mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{SPP}}$, replaces the $\mathcal{F}_{\mathsf{BC}}$-based dealer-control check by its own record of whether $\mathcal{Z}$ activated `DealerOK` (using authenticity of $\mathcal{F}_{\mathsf{BC}}$), and absorbs $\mathcal{F}_{\mathsf{NIZK}}$. The last step is what makes the rest of the argument go through: $\mathcal{S}$ can now simulate honest parties' proofs *without their witnesses* and extract a witness from every verifying proof a corrupt party produces.

With extraction in hand, $\mathcal{S}$ argues that an honest committee aggregates exactly the shares pinned down by the corrupt provers' witnesses: a mismatch between a privately delivered share and the extracted opening of the corresponding public commitment would break binding of Com, so it occurs only with negligible probability. $\mathcal{S}$ then peels the secret out of the adversary's view in three moves. Hiding of Com lets it replace every honest-to-honest commitment by a commitment to a dummy (the accompanying re-sharing proof, being simulated, stays valid). Share simulatability of SSS lets it decouple the release-time `Finish` broadcasts

Protocol $\pi_{\text{DC-ECPSS}}$ — Handover & Reconstruction

On input $(\mathsf{sid}, \texttt{Ping})$, a role R does:

1. If $R \notin \bigcup_i C_i^{\mathsf{sid}}$, **Break**.
2. Retrieve $r := r_{\mathsf{sid}}$. If $R \notin C_r^{\mathsf{sid}}$, set $r_{\mathsf{sid}} \leftarrow r{+}1$ and **Break**. Let j be such that $R = C_r^{\mathsf{sid}}[j]$.
   Phase 1: Acquire the local share from the previous handover.
3. **If** $r = 1$ *(border case: take share directly from the dealer's* `StoreSecret`*):*
   3.1 $(\mathsf{sid}, \texttt{Stored}, \mathsf{mode}, \mathcal{R}, \{c_k\}_{k\in[m^2]}, \pi_0) \leftarrow$ **Read-BC**$(D_{\mathsf{sid}}, 0)$.
   3.2 $(\mathsf{sid}, \texttt{Share}, 1, \mathsf{rsh}_j, \rho_j) \leftarrow$ **Read-P2P**(0) from D_{sid}.
   3.3 If $c_j \neq \mathsf{Com}(\mathsf{rsh}_j; \rho_j)$, **Break**.
   3.4 If $\mathcal{F}_{\mathsf{NIZK}}.\mathsf{Verify}\big(\mathcal{R}_{\mathcal{R}}^{\mathsf{Share}}, \{c_k\}_{k\in[m^2]}, \pi_0\big) = 0$, **Break**.
   3.5 Set $\mathcal{I} \leftarrow \{1\}$, $\mathsf{sh}_1 \leftarrow \mathsf{rsh}_j$, $\rho_{1,j}^{\text{in}} \leftarrow \rho_j$, $c_{1,j} \leftarrow c_j$, $C_{\text{base}} \leftarrow \{c_j\}$, $\mathsf{sh} \leftarrow \mathsf{rsh}_j$, $\rho_{\mathsf{Takeover}} \leftarrow \varepsilon$.
   The `StoreSecret` *broadcast is the dealer's implicit liveness attestation for round* 0*; the explicit* `DealerOK` *check is therefore skipped at* $r = 1$*, mirroring the ideal functionality's behavior on the first* `Ping`*. Setting* $\mathsf{sh} \leftarrow \mathsf{rsh}_j$ *is consistent with* $\mathsf{sh} = \mathsf{Takeover}(\{\mathsf{sh}_i\}_{i\in\mathcal{I}}; \rho_{\mathsf{Takeover}})$ *for the singleton* $\mathcal{I} = \{1\}$ *(no randomness needed).*
   3.6 Proceed to Phase 2 (reshare).
4. **Else** $(r \geq 2)$ *(take shares from the previous holding committee* C_{r-1}^{sid}*):*
   4.1 For each $i \in [m^2]$:
      i. $(c_{i,1}, \ldots, c_{i,m^2}, C_{\text{base},i}, \pi_i^{\text{in}}) \leftarrow$ **Read-BC**$(C_{r-1}^{\mathsf{sid}}[i], r{-}1)$.
      ii. $(\mathsf{sh}_i, \rho_{i,j}^{\text{in}}) \leftarrow$ **Read-P2P**$(r{-}1)$ from $C_{r-1}^{\mathsf{sid}}[i]$.
      iii. If $c_{i,j} \neq \mathsf{Com}(\mathsf{sh}_i; \rho_{i,j}^{\text{in}})$, mark i as invalid.
      iv. If $\mathcal{F}_{\mathsf{NIZK}}.\mathsf{Verify}(\mathcal{R}_{\mathsf{sid},r-1}^{\mathsf{Reshare}}, (C_{\text{base},i}, \{c_{i,k}\}_k), \pi_i^{\text{in}}) = 0$, mark i as invalid.
   4.2 If fewer than the reconstruction threshold i's remain valid, **Break**; otherwise set $\mathcal{I} = \{i, \text{s.t. } i \text{ is valid}\}$.
   4.3 Sample fresh random tape ρ_{Takeover} and compute $\mathsf{sh} \leftarrow \mathsf{Takeover}(\{\mathsf{sh}_i\}_{i\in\mathcal{I}}; \rho_{\mathsf{Takeover}})$; if $\mathsf{sh} = \perp$, **Break**.
   4.4 Set $C_{\text{base}} \leftarrow \{c_{i,j}\}_{i\in\mathcal{I}}$.
   Did the dealer post a round-$(r-1)$ *DealerOK on* $\mathcal{F}_{\mathsf{BC}}$*?*
   4.5 Let $m :=$ **Read-BC**$(D_{\mathsf{sid}}, r{-}1)$.
   4.6 If $m \neq (\mathsf{sid}, \texttt{DealerOK}, r{-}1)$: **Broadcast** $(\mathsf{sid}, \texttt{Finish}, r, \mathsf{mode}(\mathsf{sh}))$, and **Break**.
   Phase 2: Reshare to the next committee.
5. Sample fresh random tape ρ_{Handover} and compute $\{\mathsf{rsh}_k\}_{k\in[m^2]} \leftarrow \mathsf{Handover}(\mathsf{sh}; \rho_{\mathsf{Handover}})$.
6. For each $k \in [m^2]$, sample ρ_k^{out} and set $c_k^{\text{out}} \leftarrow \mathsf{Com}(\mathsf{rsh}_k; \rho_k^{\text{out}})$.
7. Obtain:

$$\pi_r \leftarrow \mathcal{F}_{\mathsf{NIZK}}.\mathsf{Prove}\left(\mathcal{R}_{\mathsf{sid},r}^{\mathsf{Reshare}},\ \big(C_{\text{base}}, \{c_k^{\text{out}}\}_k\big),\ \begin{pmatrix}\{\mathsf{sh}_i, \rho_{i,j}^{\text{in}}\}_{i\in\mathcal{I}},\ \mathsf{sh},\ \rho_{\mathsf{Takeover}},\\ \{\mathsf{rsh}_k, \rho_k^{\text{out}}\}_k,\ \rho_{\mathsf{Handover}}\end{pmatrix}\right).$$

8. **Broadcast** $\big(\mathsf{sid}, \texttt{Reshare}, r, \{c_k^{\text{out}}\}_k, C_{\text{base}}, \pi_r\big)$.
9. For each k: **P2P-send** $(\mathsf{sid}, \texttt{Share}, r, \mathsf{rsh}_k, \rho_k^{\text{out}})$ to $C_{r+1}^{\mathsf{sid}}[k]$.
10. Set $r_{\mathsf{sid}} \leftarrow r{+}1$.

On input $(\mathsf{sid}, \texttt{ReadSecret})$, any role does:

1. Read every broadcast for sid from $\mathcal{F}_{\mathsf{BC}}$.
2. **Parse** the latest round $r^\star$ in which at least the reconstruction threshold τ messages $(\mathsf{sid}, \texttt{Finish}, r^\star, y_i)$ were broadcast by distinct members of $C_{r^\star}^{\mathsf{sid}}$.
3. If no such $r^\star$ exists, **Break**.
4. Compute $s \leftarrow \mathsf{Reconstruct}(\{y_i\}_{i\in\mathcal{V}^\star})$.
5. Output (mode, s).

Figure 4: **Protocol** $\pi_{\text{DC-ECPSS}}$: Per-epoch handover (`Ping`) and post-release reconstruction (`ReadSecret`).

from the intermediate Takeover/Handover chain, so the only place the secret is read is the release step. Finally, processing committees in reverse order and invoking privacy (for the dealer's sharing) and re-sharing privacy (for each handover), it re-randomizes the corrupt-side shares committee by committee, leaving the adversary's entire lifetime view independent of the stored secret.

In the last hop $\mathcal{S}$ routes all secret handling through $\mathcal{F}_{\text{DC-ECPSS}}$: for an honest dealer it forwards `StoreSecret` and retrieves $\mathsf{mode}(s)$ only at release, which is undetectable since nothing before release depended on s; for a corrupt dealer it feeds $\mathcal{F}_{\text{DC-ECPSS}}$ the secret s^* it extracted from the $\mathcal{R}_{\mathcal{R}}^{\mathsf{Share}}$ proof, which by the binding argument is the unique value an honest reconstruction would recover. The full proof, with the explicit game sequence, is given in Appendix B. □

3.4 A Direct Realization of DMS-delete

We observe that DMS-delete is realized directly from $\pi_{\text{DC-ECPSS}}$.

Recall that the DMS-delete setting necessitates provable destruction of a long-term secret upon testator's death. As one can see, DC-ECPSS already realizes this requirement. It maintains a dealer's secret as a long-lived cryptographic state across epochs via Handover and Takeover algorithms, which is additionally conditioned on the dealer's ability to broadcast alive signal in every epoch. Finally, the YOSO anonymity ensures the secret shares exist hidden within the system — the current holding committee is unknown to a mobile adversary, till it speaks and erases it state.

To construct DMS-delete, we just instantiate $\pi_{\text{DC-ECPSS}}$ with $\mathsf{mode} = \mathsf{delete}$. Presumably, the dealer assumes the role of a testator: they register a secret via `StoreSecret`, which enforces a well-formedness check $\mathcal{R}(s, w) = 1$, securing against a malicious testator who might register a fraudulent secret. In every epoch, the testator broadcasts an alive signal via `DealerOK`, upon which the holding committees execute the Handover and Takeover algorithms via the `Ping` interface. The absence of a `DealerOK` signal within an epoch resonates the testator's death, upon which the current holding committee broadcasts $(\mathsf{sid}, \texttt{Finish}, r, \perp)$, and erases its local state. Quite evidently, the secret ceases to exist in the system after the terminal epoch.

We note that the deletion in DMS-delete is provable in the UC sense, i.e. the secret becomes computationally irrecoverable after the terminal phase under YOSO assumptions. Nevertheless, this erasure is not publicly verifiable in the sense of a proof of erasure. The `Finish` broadcast with $\perp$ publicly confirms that the terminal phase was reached, with the erasure of local state guaranteed by the underlying protocol specification and UC security.

Corollary 1 The protocol $\pi_{\text{DC-ECPSS}}$ parameterized with ($\mathsf{mode} = \mathsf{delete}$) UC-realizes $\mathcal{F}_{\mathsf{DMS}}$ (defined in Section) in the delete mode under the same assumptions as Theorem 1.

Proof Sketch. The proof follows immediately from Theorem 1 by observing that $\mathcal{F}_{\text{DC-ECPSS}}$ with $\mathsf{mode} = \mathsf{delete}$ delivers $\perp$ to all roles when $\mathsf{DealerOK} = 0$, which is precisely the delete mode of $\mathcal{F}_{\mathsf{DMS}}$. □

3.5 From Abstract to Real World

The abstract world of roles, which we opt throughout, provides a clean separation of modules in the sense that the protocol design can be studied independently of how roles are assigned to nodes, how shares are communicated privately to anonymous holding committees, and how authenticity is established. This separation is not merely aesthetic — it allows the security proof to go through via the UC framework, without conflating to the effects of instantiation choices. Concretely, three abstract components need real world instantiations.

Role Assignment: The YOSO model inherently assumes that the holding committees satisfy honest-majority, i.e. a randomly selected size-m party has at least half honest members with overwhelming probability. In practice, however, this assumption must be enforced without a trusted coordinator and with minimal assumptions on the overall pool of nodes. The ECPSS framework [BGG+20] addresses this (recall

prior discussion in Section) by introducing auxiliary nominating committees. At epoch r, a size-m nominating committee self-elects using a cryptographic sortition function with the sole responsibility of anonymously electing a size-m holding committee for epoch $r+1$ as follows: each member of a nominating committee selects a random party q, generates ephemeral key-pairs $(\mathsf{epk}, \mathsf{esk})$, encrypts esk under q's anonymous PKE public key pk_q, and publishes $(\mathsf{epk}, \mathsf{Enc}_{\mathsf{pk}_q}(\mathsf{esk}))$. The current holding committee (for epoch r) encrypts the fresh shares under epk such that only nominee q holding sk_q can decrypt esk to recover the share. Before broadcasting, all parties erase their share to preserve anonymity.

Concretely, an adversary controlling a minority of the pool, corrupts roughly the same fraction of the nominating committee — all of whom nominate corrupted holding committee members — plus an equal fraction of holding committees nominated by honest nominating committee members. Therefore, their solution only tolerates adversaries controlling less than 29% of the entire pool. This corresponds to the standard Q2 adversary structure in its threshold form: if an adversary corrupts at most $t < \frac{n}{2}$, then no two corruptible sets of size t can cover all N parties.

The nested YOSO model [ACD+25] considers a two-level hierarchical structure for a holding committee of size m^2. Specifically, a holding committee consists of m subcommittees of m parties each. Role assignment is due to nominating committee of size-m: each nominating committee selects a size-m subcommittee. Quite intuitively, a corrupt member may select an entirely corrupt subcommittee, whereas an honest member will choose a majorly honest subcommittee. Security holds under the Q2 assumption: as long as majority of the subcommittees in the holding committee are mostly honest, no two corruptible sets cover all parties. This relaxation significantly improves the corruption tolerance — whereas an m^2 size holding committee in [BGG+20] requires at least $\lfloor m^2/2 \rfloor + 1$ honest members, the [ACD+25] approach requires at least $\lfloor m/2 \rfloor + 1$ subcommittees each contain at least $\lfloor m/2 \rfloor + 1$ honest members (i.e. $\frac{3}{4}$ corrupt members compared to $\frac{1}{2}$).

Secure Channels: In the abstract world, $\mathcal{F}_{\mathsf{SPP}}$ provide P2P channels between roles for communicating the shares. In a real world, since the committee members are anonymous, a target anonymous channel needs to be established without knowing a priori the recipient's identity. Following [BGG+20], this is achieved via the role assignment mechanism — which precisely replaces $\mathcal{F}_{\mathsf{SPP}}$ with an encrypt-then-broadcast approach, preserving the anonymity and confidentiality guarantees of the abstract world.

Dealer's Authentication: In the abstract world, $\mathcal{F}_{\mathsf{BC}}$ implicitly guarantees the authenticity of dealer's `DealerOK` broadcast. However, in the real world, a dealer may sign their `DealerOK` message with a long-term signing key, and the holding committee members can verify the signature before proceeding to propagate the shares. To implement this, a dealer can publish the verification key on a blockchain at setup, binding the dealer's identity to a fixed key for the entire lifetime of the switch.

4 Dead Man's Switch - Release

4.1 Modeling DMS

We model DMS-release as an ideal functionality $\mathcal{F}_{\mathsf{DMS}}$ (Figure 5) that extends $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ to the release setting with authorized nominees. The dealer-control core — the flags Pinged, DResponse, Released and their transitions under `Ping` and `DealerOK` — is inherited from $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$; we describe only the departures.

`StoreSecret` additionally takes a nominee set $\mathcal{N}$, and fixes the post-processing to release. The functionality records $\mathcal{N}$ together with the secret and, upon storing, discloses the corrupt nominees $\mathcal{B}_{\mathcal{N}} = \mathcal{N} \cap \mathsf{Corrupt}$ to $\mathcal{S}$; this is the only nominee information it reveals.

`Ping` and `DealerOK` retain the $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ logic. The only change is at the terminal phase: rather than delivering the secret to every party, $\mathcal{F}_{\mathsf{DMS}}$ discloses $(\mathsf{sid}, s, \mathcal{N})$ to $\mathcal{S}$ alone and withholds s from the parties, deferring its release to `ReadSecret`.

`ReadSecret` adds the nominee authorization that $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ deliberately left to the DMS layer: after release it returns s to P_j only if $P_j \in \mathcal{N}$, and $\perp$ otherwise.

`CheckNominee` is a new interface: once a secret is stored, a party learns whether it is a nominee for sid, without learning s. This captures the ex-ante inclusion guarantee, letting a nominee confirm its designation during the testator's lifetime.

The functionality $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ runs on an input security parameter λ with a universe of N parties $\mathcal{U} = \{P_1, \ldots, P_N\}$, a simulator $\mathcal{S}$ and an environment $\mathcal{Z}$. The functionality operates in a synchronous distributed model: honest parties are activated round by round by $\mathcal{Z}$ and, when instructed, consensually invoke `Ping` in the same round. The functionality keeps a list of the currently corrupted parties in $\mathsf{Corrupt}$.

4.2 Protocol π_{DMS}

We construct π_{DMS} in the $(\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{DC\text{-}ECPSS}}, \mathcal{F}_{\mathsf{PKI}})$-hybrid model, treating $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ as a black box for dealer-controlled maintenance and $\mathcal{F}_{\mathsf{PKI}}$ (Figure 11) as a registry of the nominees' public keys. Besides these, the construction uses an $\mathsf{IND\text{-}Anon\text{-}CPA}$ multi-recipient PKE mPKE [KKPP20], an $\mathsf{IND\text{-}CPA}$ public-key encryption PKE, a key-committing symmetric encryption SKE, and the NIZK relations below.

The guiding principle is that $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ never holds the secret s: it maintains only a compact ciphertext c_0 from which s can be recovered, while s itself stays encrypted in a publicly broadcast ciphertext c_2. The testator samples a symmetric key K and sets $c_2 = \mathsf{SKE.Enc}(K, s)$; generates an ephemeral PKE pair $(\mathsf{esk}, \mathsf{epk})$ and encrypts K as $c_0 = \mathsf{PKE.Enc}(\mathsf{epk}, K)$; and encrypts the decryption key esk towards the nominees as $c_1 = \mathsf{mPKE.Enc}(\{\mathsf{pk}_i\}_{i \in \mathcal{N}}, \mathsf{esk})$. Only c_0 is entrusted to $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$, while (c_1, c_2) is published on $\mathcal{F}_{\mathsf{BC}}$.

NIZK Relations.

$$\begin{aligned}
\mathcal{R}_{\mathcal{R}}^{\mathsf{Enc}}\big(\, x,\, w \,\big) : {} & x = c,\ w = (K, s, w') \\
& \text{s.t. } \mathcal{R}(s, w') = 1,\ \ c = \mathsf{SKE.Enc}(K, s).
\end{aligned}$$

$$\begin{aligned}
\mathcal{R}_{c_1, c_2}^{\mathsf{Store}}\big(\, x,\, w \,\big) : {} & x = c_0,\ w = \big(\mathsf{esk}, \{\mathsf{pk}\}_{\mathcal{N}}, R\big) \\
& \text{s.t. } c_1 = \mathsf{mPKE.Enc}(\{\mathsf{pk}\}_{\mathcal{N}}, \mathsf{esk};\, R), \\
& \quad \mathsf{SKE.Dec}\big(\mathsf{PKE.Dec}(\mathsf{esk}, c_0),\, c_2\big) \neq \bot.
\end{aligned}$$

We now describe how each interface drives $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$.

Init: A nominee party samples an mPKE key pair, keeps the secret key locally, and registers the public key with $\mathcal{F}_{\mathsf{PKI}}$.

StoreSecret: The testator forms (c_0, c_1, c_2) as above together with a NIZK π that c_2 encrypts a well-formed secret (relation $\mathcal{R}_{\mathcal{R}}^{\mathsf{Enc}}$). It then invokes `StoreSecret` interface of $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$, *using the secret slot for* c_0, the relation slot for $\mathcal{R}_{c_1, c_2}^{\mathsf{Store}}$ (which binds c_0 to the published (c_1, c_2)), and mode $\mathsf{release}$. Well-formedness of s is thus carried by π on $\mathcal{F}_{\mathsf{BC}}$, while consistency of the chain is the relation $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ itself checks. The triple (c_1, c_2, π) is broadcast.

Ping: A party relays `Ping` to $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ unchanged; before its first relay it verifies the testator's broadcast, that (c_1, c_2) matches the relation handed to $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ and that π verifies, and aborts otherwise.

DealerOK: Relayed to $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ verbatim: the testator's alive signal is precisely the $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ one.

ReadSecret: A party obtains the released c_0 from $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$, decrypts c_1 with its mPKE secret key to recover esk (or $\bot$ if it is not a nominee), then recovers $K = \mathsf{PKE.Dec}(\mathsf{esk}, c_0)$ and outputs $s = \mathsf{SKE.Dec}(K, c_2)$.

CheckNominee: Without waiting for release, a party decrypts the published c_1 with its mPKE secret key; a non-$\bot$ result certifies nomination. Since $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ withholds c_0 until release, this reveals inclusion but not s.

Functionality $\mathcal{F}_{\mathsf{DMS}}$

On input $(\mathsf{sid}, \texttt{Init})$ from a party P:

1. If P is already initialized for sid, **Break**.
2. Mark P as initialized for sid.
3. **Send** $(\mathsf{sid}, \texttt{Initialized}, P)$ to $\mathcal{S}$.

On input $(\mathsf{sid}, \texttt{StoreSecret}, s, w, \mathcal{R}, \mathcal{N})$ from a party P_j:

1. If there is an entry for sid, **Send** $(\mathsf{sid}, \perp)$ to P_j and **Break**
2. If $\mathcal{R}(s, w) \neq 1$, **Send** $(\mathsf{sid}, \perp)$ to P_j and **Break**
3. **Store** $(\mathsf{sid}, s, D = P_j, \mathcal{N}, \mathsf{Pinged} = 0, \mathsf{DResponse} = 0, \mathsf{Released} = 0)$
4. Let $\mathcal{B}_{\mathcal{N}} \leftarrow \mathcal{N} \cap \mathsf{Corrupt}$.
5. **Send** $(\mathsf{sid},\ P_j,\ \texttt{Stored}, \mathcal{R},\ \mathcal{B}_{\mathcal{N}})$ to $\mathcal{S}$ and $(\mathsf{sid},\ P_j,\ \texttt{Stored}, \mathcal{R})$ to all parties.

On input $(\mathsf{sid}, \texttt{Ping})$ from every honest party in the same round:

In the synchronous model, $\mathcal{Z}$ schedules a round and activates every honest party with input $(\mathsf{sid}, \texttt{Ping})$. $\mathcal{F}_{\mathsf{DMS}}$ processes the ping only once all honest parties have consensually submitted it within that round; otherwise the activation is ignored.

1. **Retrieve** $(\mathsf{sid}, s, D, \mathcal{N}, \mathsf{Pinged}, \mathsf{DResponse}, \mathsf{Released})$
   No secret with such sid *or it was released*
2. If there is no entry for sid or $\mathsf{Released} = 1$, **Send** $(\mathsf{sid}, \perp)$ to every honest party and to $\mathcal{S}$, and **Break**
3. If $\mathsf{Pinged} = 1$ and $\mathsf{DResponse} = 0$:
   3.1 **Replace** entry for sid with $(\mathsf{sid}, s, D, \mathcal{N}, \perp, \perp, 1)$
   3.2 **Send** $(\mathsf{sid}, s, \mathcal{N})$ to $\mathcal{S}$ and **Break**.
   Otherwise: first ping, or dealer responded to the previous one
4. Set $\mathsf{Pinged} \leftarrow 1$ and $\mathsf{DResponse} \leftarrow 0$ for sid
5. **Send** $(\mathsf{sid},\ \texttt{Pinged})$ to D and to $\mathcal{S}$

On input $(\mathsf{sid}, \texttt{DealerOK})$ from P_j:

1. **Retrieve** $(\mathsf{sid}, s, D, \mathcal{N}, \mathsf{Pinged}, \mathsf{DResponse}, \mathsf{Released})$
   Check that an unreleased secret with sid *exists, that OK comes from the correct dealer, and that no OK was sent for the current ping*
2. If there is no entry for sid or $\mathsf{Released} = 1$ or $D \neq P_j$ or $\mathsf{Pinged} = 0$ or $\mathsf{DResponse} = 1$, then **Send** $(\mathsf{sid}, \perp)$ to P_j and **Break**
   Otherwise this is a legitimate reply from the dealer
3. Set $\mathsf{DResponse} \leftarrow 1$ for sid
4. **Send** $(\mathsf{sid},\ \texttt{DealerOK})$ to $\mathcal{S}$

On input $(\mathsf{sid}, \texttt{ReadSecret})$ from P_j:

1. **Retrieve** $(\mathsf{sid}, s, D, \mathcal{N}, \mathsf{Pinged}, \mathsf{DResponse}, \mathsf{Released})$
2. If $\mathsf{Released} = 0$, then **Send** $(\mathsf{sid}, \perp)$ to P_j and **Break**
3. If $P_j \notin \mathcal{N}$, then **Send** $(\mathsf{sid}, \perp)$ to P_j and **Break**
4. **Send** $(\mathsf{sid},\ s)$ to P_j

On input $(\mathsf{sid}, \texttt{CheckNominee})$ from P_j:

1. **Retrieve** $(\mathsf{sid}, s, D, \mathcal{N}, \mathsf{Pinged}, \mathsf{DResponse}, \mathsf{Released})$
   No secret stored yet
2. If $s = \perp$ then **Send** $(\mathsf{sid}, \perp)$ to P_j and **Break**
3. If $P_j \notin \mathcal{N}$, then **Send** $(\mathsf{sid}, \perp)$ to P_j and **Break**
4. **Send** $(\mathsf{sid},\ \texttt{Nominee})$ to P_j

Figure 5: Ideal Functionality for a DMS scheme

Protocol π_{DMS} — Store & Handover

On input $(\mathsf{sid}, \texttt{Init})$ from a party P:

1. Set $r_{\mathsf{sid}} \leftarrow 1$.
2. If $P \in \mathcal{U}_{\mathcal{N}}$ and wasn't registered in $\mathcal{F}_{\mathsf{PKI}}$:
   2.1 Generate $(\mathsf{sk}_P, \mathsf{pk}_P) \leftarrow \mathsf{mPKE.Gen}(1^\lambda)$ and store sk_P in local state for sid.
   2.2 Input $(\texttt{Register}, \mathsf{pk}_P)$ to $\mathcal{F}_{\mathsf{PKI}}$.

On input $(\mathsf{sid}, \texttt{StoreSecret}, s, w, \mathcal{R}, \mathcal{N})$, a party P_j does:

1. If someone has already executed `StoreSecret` for sid, **Break**.
2. Generate ephemeral PKE key pair $(\mathsf{esk}, \mathsf{epk}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$.
3. Sample a fresh symmetric key $K \leftarrow\!\$\ \{0,1\}^\lambda$ and compute $c_0 \leftarrow \mathsf{PKE.Enc}(\mathsf{epk}, K; R_0)$ with fresh randomness R_0.
4. For each $i \in \mathcal{N}$, retrieve $\mathsf{pk}_i \leftarrow \mathcal{F}_{\mathsf{PKI}}.\texttt{Query}(i)$; if any query returns $\perp$, **Break**.
5. Encrypt esk towards nominees $(c_1; R) \leftarrow \mathsf{mPKE.Enc}(\{\mathsf{pk}_i\}_{i\in\mathcal{N}}, \mathsf{esk})$.
6. Encrypt the secret $c_2 \leftarrow \mathsf{SKE.Enc}(K, s)$.
   Prove secret in c_2 satisfies $\mathcal{R}$.
7. Obtain $\pi \leftarrow \mathcal{F}_{\mathsf{NIZK}}.\mathsf{Prove}\big(\mathcal{R}^{\mathsf{Enc}}_{\mathcal{R}},\ c_2,\ (K, s, w)\big)$.
8. Input to $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$:

$$(\mathsf{sid}, \texttt{StoreSecret}, c_0, w = (\mathsf{esk}, \{\mathsf{pk}\}_{\mathcal{N}}, R), \mathcal{R}^{\mathsf{Store}}_{c_1,c_2}, \mathsf{release}).$$

9. **Broadcast** $\big(\mathsf{sid}, \texttt{Stored}, \mathcal{R}, (c_1, c_2), \pi\big)$.

On input $(\mathsf{sid}, \texttt{Ping})$, a party P_j does:

1. If $r_{\mathsf{sid}} = 1$:
   1.1 Check for input $(\mathsf{sid}, P_i, \texttt{Stored}, \mathsf{release}, \mathcal{R}^{\mathsf{Store}}_{c_1',c_2'})$ from $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$; if none, **Break**.
   Read `Stored` message from P_i for sid from broadcast.
   1.2 $(c_1, c_2) \leftarrow$ **Read-BC**$(P_i, 1)$ and set P_i as D_{sid}.
   1.3 Check $\mathcal{R}^{\mathsf{Store}}_{c_1',c_2'} = \mathcal{R}^{\mathsf{Store}}_{c_1,c_2}$, if not **Break**.
   1.4 If $\mathcal{F}_{\mathsf{NIZK}}.\mathsf{Verify}(\mathcal{R}^{\mathsf{Enc}}_{\mathcal{R}}, c_2, \pi_i^{\mathrm{in}}) = 0$, then **Break**.
2. Input to $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$: $(\mathsf{sid}, \texttt{Ping})$.
3. Set $r_{\mathsf{sid}} \leftarrow r_{\mathsf{sid}} + 1$.

On input $(\mathsf{sid}, \texttt{DealerOK})$, a party P_j does:

1. Input to $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$: $(\mathsf{sid}, \texttt{DealerOK})$.

Figure 6: **Protocol π_{DMS}**: Setup, Store and Ping in $(\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{DC\text{-}ECPSS}}, \mathcal{F}_{\mathsf{PKI}})$.

Protocol π_{DMS} — Dealer Control & Reconstruction

On input $(\mathsf{sid}, \texttt{ReadSecret})$, any party does:

1. Check for input $(\mathsf{sid}, P_i, \texttt{Stored}, \mathsf{release}, \mathcal{R}^{\mathsf{Store}}_{c_1',c_2'})$ from $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$; if none, **Break**.
   Read `Stored` *message from* P_i *for* sid *from broadcast.*
2. $(c_1, c_2) \leftarrow$ **Read-BC**$(P_i, 1)$.
3. Check $\mathcal{R}^{\mathsf{Store}}_{c_1',c_2'} = \mathcal{R}^{\mathsf{Store}}_{c_1,c_2}$, if not **Break**.
4. If $\mathcal{F}_{\mathsf{NIZK}}.\mathsf{Verify}(\mathcal{R}^{\mathsf{Enc}}_{\mathcal{R}}, c_2, \pi_i^{\mathrm{in}}) = 0$, then **Break**.
5. Input $(\mathsf{sid}, \texttt{ReadSecret})$ to $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ to get c_0; if $\perp$ is returned, output $(\mathsf{sid}, \perp)$.
6. Retrieve the local mPKE secret key sk_P stored for sid by a prior $(\mathsf{sid}, \texttt{Init}, \mathsf{nominee})$ call; if none, output $(\mathsf{sid}, \perp)$.
7. $\mathsf{esk} \leftarrow \mathsf{mPKE.Dec}(\mathsf{sk}_P, c_1)$.
   If esk *is* $\perp$*, then not a nominee.*
8. If $\mathsf{esk} = \perp$, output $(\mathsf{sid}, \perp)$.
9. $K \leftarrow \mathsf{PKE.Dec}(\mathsf{esk}, c_0)$.
10. $s \leftarrow \mathsf{SKE.Dec}(K, c_2)$.
11. Output (sid, s).

On input $(\mathsf{sid}, \texttt{CheckNominee})$, a party P_j does:

1. If $P_j \notin \mathcal{U}_{\mathcal{N}}$, **Break**.
2. Check for input $(\mathsf{sid}, P_i, \texttt{Stored}, \mathsf{release}, \mathcal{R}^{\mathsf{Store}}_{c_1',c_2'})$ from $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$; if none, **Break**.
   Read `Stored` *message from* P_i *for* sid *from broadcast.*
3. $(c_1, c_2) \leftarrow$ **Read-BC**$(P_i, 1)$.
4. Check $\mathcal{R}^{\mathsf{Store}}_{c_1',c_2'} = \mathcal{R}^{\mathsf{Store}}_{c_1,c_2}$, if not **Break**.
5. If $\mathcal{F}_{\mathsf{NIZK}}.\mathsf{Verify}(\mathcal{R}^{\mathsf{Enc}}_{\mathcal{R}}, c_2, \pi_i^{\mathrm{in}}) = 0$, then **Break**.
6. Retrieve the local mPKE secret key sk_P stored for sid by a prior $(\mathsf{sid}, \texttt{Init}, \mathsf{nominee})$ call; if none, output $(\mathsf{sid}, \perp)$.
7. $\mathsf{esk} \leftarrow \mathsf{mPKE.Dec}(\mathsf{sk}_P, c_1)$.
   If esk *is* $\perp$*, then not a nominee.*
8. If $\mathsf{esk} = \perp$, output $(\mathsf{sid}, \perp)$.
9. Output $(\mathsf{sid}, \texttt{Nominee})$.

Figure 7: **Protocol** π_{DMS}: Dealer Control and Reconstruction.

4.3 Security of π_{DMS}

Corruption model. We use a partially-static / partially-adaptive corruption model. The subset of $\mathcal{U}$ of parties eligible to act as nominees is fixed beforehand, denoted $\mathcal{U}_{\mathcal{N}}$. The environment $\mathcal{Z}$ commits, before the execution for sid starts, to (i) the dealer's corruption status and (ii) the static set of corrupt nominee parties, yielding for each sid a subset $\mathcal{B}_{\mathcal{N}} \subseteq \mathcal{N} \subseteq \mathcal{U}_{\mathcal{N}}$ of corrupt nominees of that session. All other parties may be corrupted adaptively at any time. We additionally assume *secure erasure*: an honest dealer erases its `StoreSecret`-local ephemeral state $(\mathsf{esk}, K, s, \rho_*)$ at the end of `StoreSecret`, so adaptive dealer corruption thereafter reveals no secret-dependent state.

Theorem 2 (π_{DMS} UC-realizes $\mathcal{F}_{\mathsf{DMS}}$) *Let* mPKE *be an* IND-Anon-CPA *multi-recipient PKE (Appendix A.3);* PKE *an* IND-CPA *public-key encryption (Appendix A.2);* SKE *a key-committing symmetric encryption with equivocation extractor (Appendix A.4); and* $\mathcal{F}_{\mathsf{NIZK}}$ *a simulation-extractable NIZK (Figure 10) for* $\mathcal{R}^{\mathsf{Enc}}_{\mathcal{R}}$*. Then* π_{DMS} *UC-realizes* $\mathcal{F}_{\mathsf{DMS}}$ *in the* $(\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{DC\text{-}ECPSS}}, \mathcal{F}_{\mathsf{PKI}})$*-hybrid model against any PPT adversary* $\mathcal{A}$ *in*

the partial-adaptive corruption model described above.

Proof. (Proof sketch) We construct a simulator $\mathcal{S}$ that treats $\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{DC\text{-}ECPSS}}, \mathcal{F}_{\mathsf{PKI}}$ as internal sub-functionalities (absorbing their UC simulators) and argue by cases on the corruption pattern. The main observation is that $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ stores only the compact ciphertext c_0, never the secret s; the secret lives encrypted in the public c_2, recoverable through the chain $c_1 \to \mathsf{esk} \to K \to s$. Recall $\mathcal{F}_{\mathsf{DMS}}$ leaks the corrupt nominee subset $\mathcal{B}_{\mathcal{N}}$ to $\mathcal{S}$ at `StoreSecret`, so $\mathcal{S}$ knows from the outset which recipients the adversary controls.

If the dealer is corrupt, $\mathcal{S}$ is essentially passive. From the dealer's broadcast $\mathcal{R}_{\mathcal{R}}^{\mathsf{Enc}}$ proof, simulation-extractability of $\mathcal{F}_{\mathsf{NIZK}}$ yields a witness exhibiting a Rel-valid secret s with $c_2 = \mathsf{SKE.Enc}(K, s)$; meanwhile $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$'s $\mathcal{R}^{\mathsf{Store}}$ check (performed perfectly by the ideal functionality) guarantees that the chain reconstructs to $\mathsf{SKE.Dec}(\mathsf{PKE.Dec}(\mathsf{esk}, c_0), c_2)$. Since c_2 decrypts successfully under at most one key (key-commitment of SKE), the two agree, so the value honest nominees would recover is exactly the extracted s, which $\mathcal{S}$ forwards to $\mathcal{F}_{\mathsf{DMS}}$. Dealer-control events are relayed verbatim, the $\mathcal{F}_{\mathsf{DMS}}$ and $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ state machines being identical.

If the dealer is honest, $\mathcal{S}$ proceeds through a short sequence of games that strips s from the adversary's view. It first simulates the dealer's NIZK, then makes c_1 independent of the honest recipients — towards dummy keys when all nominees are honest, or towards only the corrupt subset $\mathcal{B}_{\mathcal{N}}$ otherwise — by anonymity of mPKE. It then severs the symmetric key K from the adversary's view ($\mathsf{IND\text{-}CPA}$ of PKE, since $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ withholds c_0 during the lifetime and $\mathcal{A}$ never holds esk before release) and replaces c_2 by a uniform string (one-time SKE security). At this point nothing before release depends on s, so $\mathcal{S}$ routes storage through $\mathcal{F}_{\mathsf{DMS}}$ and obtains s only at the release-triggering Ping; it then extracts, using $\mathsf{Ext}_{\mathsf{SKE}}$ to pick the key $K^\star$ consistent with (c_2, s) and encrypting it honestly into c_0, so that a corrupt nominee's recovery yields s. The full proof, with the explicit games for all cases, is given in Appendix C. □

References

[ACD+25] Ittai Abraham, Eli Chouatt, Ivan Damgård, Yossi Gilad, Gilad Stern, and Sophia Yakoubov. Nested yoso mpc: Near optimal resilience without an mpc setup. *Cryptology ePrint Archive*, 2025.

[AFP25] Amit Agarwal, Rex Fernando, and Benny Pinkas. Efficiently-thresholdizable batched identity based encryption, with applications. In *Annual International Cryptology Conference*, pages 69–100. Springer, 2025.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018.

[Bei96] Amos Beimel. *Secure schemes for secret sharing and key distribution.* PhD thesis, Technion - Israel Institute of Technology, Israel, 1996.

[BGG+20] Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? In *Theory of Cryptography Conference*, pages 260–290. Springer, 2020.

[BN00] Dan Boneh and Moni Naor. Timed commitments. In *Annual international cryptology conference*, pages 236–254. Springer, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145. IEEE Computer Society Press, October 2001. doi:10.1109/SFCS.2001.959888.

[CCN+23] Andrea Cerulli, Aisling Connolly, Gregory Neven, Franz-Stefan Preiss, and Victor Shoup. vetkeys: How a blockchain can keep many secrets. *Cryptology ePrint Archive*, 2023.

[DJ97] Yvo Desmedt and Sushil Jajodia. Redistributing secret shares to new access structures and its applications. Technical report, Technical Report ISSE TR-97-01, George Mason University, 1997.

[DT06] Ivan Damgård and Rune Thorbek. Linear integer secret sharing and distributed exponentiation. In *Proceedings of the 9th International Conference on Theory and Practice of Public-Key Cryptography*, PKC'06, page 75–90, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11745853_6.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 467–476, 2013.

[GHK+21] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. Yoso: You only speak once: Secure mpc with stateless ephemeral roles. In *Annual International Cryptology Conference*, pages 64–93. Springer, 2021.

[GKM+22] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. In *IACR International Conference on Public-Key Cryptography*, pages 252–282. Springer, 2022.

[HJKY95] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *annual international cryptology conference*, pages 339–352. Springer, 1995.

[KKPP20] Shuichi Katsumata, Kris Kwiatkowski, Federico Pintore, and Thomas Prest. Scalable ciphertext compression techniques for post-quantum kems and their applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 289–320. Springer, 2020.

[KL12] Alptekin Küpçü and Anna Lysyanskaya. Usable optimistic fair exchange. *Computer Networks*, 56(1):50–63, 2012.

[KRY22] Sebastian Kolby, Divya Ravi, and Sophia Yakoubov. Constant-round YOSO MPC without setup. Cryptology ePrint Archive, Paper 2022/187, 2022. URL https://eprint.iacr.org/2022/187.

[MM22] Owen McCadden and Houlton McGuinn. A cryptographic dead person's switch. 2022.

[RSW96] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979. doi:10.1145/359168.359176.

[SST20] István András Seres, Omer Shlomovits, and Pratyush Ranjan Tiwari. Cryptowills: how to bequeath cryptoassets. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 417–426. IEEE, 2020.

[Tim93] C Timothy. May. *Time-release crypto*, 1993.

[Wik22] Wikipedia. Dead man's switch: A crypto mystery. https://en.wikipedia.org/wiki/Dead_Man%27s_Switch:_A_Crypto_Mystery, 2022.

[Wik25] Wikipedia. Vanish (computer science). https://en.wikipedia.org/wiki/Vanish_(computer_science), 2025.

[ZSVR05] Lidong Zhou, Fred B Schneider, and Robbert Van Renesse. Apss: Proactive secret sharing in asynchronous systems. *ACM transactions on information and system security (TISSEC)*, 8(3):259–286, 2005.

SUPPLEMENTAL MATERIALS

A Additional Preliminaries

A.1 Commitment Schemes

Definition 9 *A* non-interactive commitment scheme Com *is a polynomial algorithm that maps a message* m *and randomness* ρ *to a commitment* $c = \mathsf{Com}(m;\rho)$*; a commitment is opened by revealing* (m,ρ)*, and a verifier accepts iff* $c = \mathsf{Com}(m;\rho)$*.*

We require the following two properties.

Definition 10 (Hiding) Com *is* (statistically) hiding *if for every pair of messages* m_0, m_1*, the distributions of* $\mathsf{Com}(m_0;\rho)$ *and* $\mathsf{Com}(m_1;\rho)$ *over uniform* ρ *are statistically indistinguishable.*

Definition 11 (Binding) Com *is* (computationally) binding *if for every PPT adversary* $\mathcal{A}$*,*

$$\Pr\left[\,\mathsf{Com}(m_0;\rho_0) = \mathsf{Com}(m_1;\rho_1) \ \wedge\ m_0 \neq m_1 : (m_0,\rho_0,m_1,\rho_1) \leftarrow \mathcal{A}(1^\lambda)\,\right] \leq \mathsf{negl}(\lambda).$$

A.2 Public Key Encryption

Definition 12 *A* public-key encryption *scheme* $\mathsf{PKE} = (\mathsf{Gen},\mathsf{Enc},\mathsf{Dec})$ *with message space* $\mathcal{M}$ *is a tuple of PPT algorithms:*

$(\mathsf{epk},\mathsf{esk}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$**:** *outputs a public key* epk *and a secret key* esk*.*

$c \leftarrow \mathsf{PKE.Enc}(\mathsf{epk}, m;\rho)$**:** *encrypts* $m \in \mathcal{M}$ *under* epk *with randomness* ρ*.*

$m' \leftarrow \mathsf{PKE.Dec}(\mathsf{esk}, c)$**:** *outputs a message* $m' \in \mathcal{M}$ *or* $\perp$*.*

Correctness *requires* $\mathsf{PKE.Dec}(\mathsf{esk},\mathsf{PKE.Enc}(\mathsf{epk},m)) = m$ *for all* $m \in \mathcal{M}$ *and all* $(\mathsf{epk},\mathsf{esk})$ *in the support of* Gen*.*

Definition 13 (IND-CPA security) PKE *is* IND-CPA secure *if for every PPT adversary* $\mathcal{A} = (\mathcal{A}_1,\mathcal{A}_2)$ *outputting* $|m_0| = |m_1|$*,*

$$\left|\Pr\left[b' = b \ :\ \begin{array}{l}(\mathsf{epk},\mathsf{esk}) \leftarrow \mathsf{Gen}(1^\lambda),\ (m_0,m_1,\mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{epk}),\\ b \leftarrow\!\$\ \{0,1\},\ c \leftarrow \mathsf{Enc}(\mathsf{epk},m_b),\ b' \leftarrow \mathcal{A}_2(c,\mathsf{st})\end{array}\right] - \tfrac{1}{2}\right| \leq \mathsf{negl}(\lambda).$$

A.3 Multi-Recipient Public Key Encryption

Definition 14 *A* multi-recipient PKE $\mathsf{mPKE} = (\mathsf{mGen},\mathsf{mEnc},\mathsf{mDec})$ *encrypts a single message to a set of recipients at once:*

$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{mGen}(1^\lambda)$**:** *per-recipient key generation.*

$c \leftarrow \mathsf{mEnc}(\{\mathsf{pk}_i\}_{i\in\mathcal{N}}, m;\rho)$**:** *encrypts* m *towards the recipient set* $\{\mathsf{pk}_i\}_{i\in\mathcal{N}}$*.*

$m' \leftarrow \mathsf{mDec}(\mathsf{sk}_i, c)$**:** *recipient* i *recovers* m*, or* $\perp$ *if* i *is not a recipient of* c*.*

Correctness *requires* $\mathsf{mDec}(\mathsf{sk}_i,c) = m$ *for every* $i \in \mathcal{N}$ *when* $c \leftarrow \mathsf{mEnc}(\{\mathsf{pk}_i\}_{i\in\mathcal{N}}, m)$*, and* $\mathsf{mDec}(\mathsf{sk}_j,c) = \perp$ *for* $j \notin \mathcal{N}$*.*

We use the *anonymous* variant [KKPP20], in which a ciphertext hides both the message and the recipient set.

Definition 15 (IND-Anon-CPA security) mPKE *is* IND-Anon-CPA secure *if for every PPT adversary* $\mathcal{A} = (\mathcal{A}_1,\mathcal{A}_2)$ *that, given all public keys, outputs two recipient sets* $\mathcal{N}^0,\mathcal{N}^1$ *with* $|\mathcal{N}^0| = |\mathcal{N}^1|$ *and two messages* m_0, m_1 *with* $|m_0| = |m_1|$*, while holding no secret key* sk_i *for* $i \in \mathcal{N}^0 \cup \mathcal{N}^1$*,*

$$\left|\Pr[b' = b \ :\ b \leftarrow\!\$\ \{0,1\},\ c \leftarrow \mathsf{mEnc}(\{\mathsf{pk}_i\}_{i\in\mathcal{N}^b}, m_b),\ b' \leftarrow \mathcal{A}_2(c,\mathsf{st})] - \tfrac{1}{2}\right| \leq \mathsf{negl}(\lambda).$$

A.4 Symmetric-Key Encryption

Definition 16 *A* symmetric-key encryption *scheme* $\mathsf{SKE} = (\mathsf{Enc}, \mathsf{Dec})$ *with key space* $\{0,1\}^\lambda$ *and message space* $\mathcal{M}$ *consists of two PPT algorithms:*

$c \leftarrow \mathsf{SKE.Enc}(K, m)$**:** *encrypts* $m \in \mathcal{M}$ *under key* K*.*

$m' \leftarrow \mathsf{SKE.Dec}(K, c)$**:** *outputs a message* $m' \in \mathcal{M}$ *or* $\perp$*.*

We absorb the encryption randomness into the ciphertext, so that $\mathsf{SKE.Dec}(K, \cdot)$ *is the deterministic inverse of* $\mathsf{SKE.Enc}(K, \cdot)$*.* Correctness *requires* $\mathsf{SKE.Dec}(K, \mathsf{SKE.Enc}(K, m)) = m$ *for all* $K \in \{0,1\}^\lambda$ *and* $m \in \mathcal{M}$*.*

We require three further properties.

Definition 17 (One-time indistinguishability) *For every* $m_0, m_1 \in \mathcal{M}$ *with* $|m_0| = |m_1|$ *and uniform* $K \leftarrow\!\!\$\ \{0,1\}^\lambda$*, the ciphertexts* $\mathsf{SKE.Enc}(K, m_0)$ *and* $\mathsf{SKE.Enc}(K, m_1)$ *are indistinguishable to any PPT adversary given no other information about* K*. (For the one-time pad the two distributions are identical.)*

Definition 18 (Key-commitment) SKE *is* key-committing *if no PPT adversary finds* (c, K_0, K_1) *with* $K_0 \neq K_1$ *and* $\mathsf{SKE.Dec}(K_0, c) \neq \perp \neq \mathsf{SKE.Dec}(K_1, c)$ *except with negligible probability; that is, every ciphertext decrypts successfully under at most one key.*

Definition 19 (Equivocation extractor) SKE *admits an* equivocation extractor $\mathsf{Ext}_{\mathsf{SKE}}$ *if, given any pair* (c, m) *for which some key* K *satisfies* $\mathsf{SKE.Enc}(K, m) = c$*, the algorithm* $\mathsf{Ext}_{\mathsf{SKE}}(c, m)$ *returns that key (unique by key-commitment).*

Remark 1 A bare one-time pad is not key-committing, since every key decrypts every ciphertext. Appending a key-binding tag fixes this: writing $\mathsf{SKE.Enc}(K, m) = \big(\mathsf{PRG}(K) \oplus m,\ H(K)\big)$ with PRG and H modeled as random oracles, the tag $H(K)$ enforces key-commitment by preimage resistance, and $\mathsf{Ext}_{\mathsf{SKE}}$ is realized in the programmable random-oracle model (sample K, then program PRG and H so that the chosen (c, m) is consistent). For $|K| = |\mathcal{M}|$ this degenerates to a tagged one-time pad.

A.5 Universal Composability Framework

We analyze security in the Universal Composability (UC) framework [Can01]. A protocol π *UC-realizes* an ideal functionality $\mathcal{F}$ if for every PPT adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ such that no PPT environment $\mathcal{Z}$ can distinguish a real execution of π with $\mathcal{A}$ from an ideal execution involving $\mathcal{F}$ and $\mathcal{S}$. We work in the $\mathcal{G}$-hybrid model, in which the protocol invokes one or more ideal sub-functionalities $\mathcal{G}$; by the composition theorem, replacing each $\mathcal{G}$ with a protocol that UC-realizes it preserves security. Our constructions use the channel functionalities $\mathcal{F}_{\mathsf{BC}}$ (authenticated broadcast [GHK+21], Figure 8) and $\mathcal{F}_{\mathsf{SPP}}$ (secure point-to-point [GHK+21], Figure 9), the NIZK functionality $\mathcal{F}_{\mathsf{NIZK}}$ (**??**, Figure 10), and the public-key registry $\mathcal{F}_{\mathsf{PKI}}$ (Figure 11).

Functionality $\mathcal{F}_{\mathsf{BC}}$ (authenticated broadcast)

$\mathcal{F}_{\mathsf{BC}}$ stores broadcast messages indexed by (sender, round), initialized to $\emptyset$.

On input $(\mathsf{sid}, \texttt{Broadcast}, m)$ from a party R in round r:

1. Record (R, r, m) and send $(\mathsf{sid}, \texttt{Broadcast}, R, r, m)$ to $\mathcal{S}$.
   The message is public, and authenticated: it is bound to sender R and round r.

On input $(\mathsf{sid}, \texttt{Read}, R, r)$ from any party Q:

1. Return the message recorded for (R, r), or $\perp$ if none exists.

Figure 8: Ideal functionality $\mathcal{F}_{\mathsf{BC}}$ for authenticated broadcast. A message broadcast by R in round r is publicly readable by any party (and known to $\mathcal{S}$), and is bound to its sender and round.

Functionality $\mathcal{F}_{\mathsf{SPP}}$ (secure point-to-point)

$\mathcal{F}_{\mathsf{SPP}}$ stores messages addressed to a recipient for the next round, initialized to $\emptyset$.

On input $(\mathsf{sid}, \texttt{Send}_{\mathsf{SPP}}, R, m)$ from a party R' in round r:

1. Record (R', R, r, m) for delivery to R at round $r{+}1$, and send the metadata $(\mathsf{sid}, \texttt{Send}_{\mathsf{SPP}}, R', R, r, |m|)$ to $\mathcal{S}$.
   Only the length $|m|$ leaks; the content is delivered privately to R alone.

On input $(\mathsf{sid}, \texttt{Read}, r)$ from a party R:

1. Return every message recorded for R at round $r{-}1$.

Figure 9: Ideal functionality $\mathcal{F}_{\mathsf{SPP}}$ for a secure point-to-point channel. A message sent in round r is delivered privately to its recipient at round $r{+}1$; the adversary learns only the sender, recipient, and message length.

Functionality $\mathcal{F}_{\mathsf{NIZK}}$ (UC NIZK with simulation-extractability)

$\mathcal{F}_{\mathsf{NIZK}}$ is parameterised by a family $\{\mathcal{R}\}$ of NP relations. It maintains an internal table Π of *honest* (statement, proof) pairs, initialized to $\emptyset$. $\mathcal{F}_{\mathsf{NIZK}}$ interacts with parties (or roles) and with the simulator $\mathcal{S}$; some interfaces are accessible only to $\mathcal{S}$.

On input $(\mathsf{sid}, \mathsf{Prove}, \mathcal{R}, x, w)$ from a role R:

1. If $(x, w) \notin \mathcal{R}$, return $\perp$.
2. Send $(\mathsf{sid}, \mathsf{Prove}, \mathcal{R}, x)$ to $\mathcal{S}$ and wait for a proof π from $\mathcal{S}$.
   (In particular, $\mathcal{S}$ returns a simulated proof without learning w; this is the zero-knowledge interface.)
3. Insert $(\mathcal{R}, x, \pi)$ into Π and return π to R.

On input $(\mathsf{sid}, \mathsf{Verify}, \mathcal{R}, x, \pi)$ from a role R:

1. If $(\mathcal{R}, x, \pi) \in \Pi$, return 1. *(Completeness for honest proofs.)*
2. Send $(\mathsf{sid}, \mathsf{Verify}, \mathcal{R}, x, \pi)$ to $\mathcal{S}$ and wait for a witness w or the symbol $\perp$.
3. If $w \neq \perp$ and $(x, w) \in \mathcal{R}$, insert $(\mathcal{R}, x, \pi)$ into Π and return 1. *(Simulation-extractability: any verifying proof not produced by an honest party comes with an extractable witness.)*
4. Otherwise return 0. *(Soundness: no proof verifies without a witness in $\mathcal{R}$.)*

Figure 10: Ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$ for UC non-interactive zero-knowledge with simulation-extractability. The interface is the standard Camenisch–Krenn–Shoup style: Prove requests embed zero-knowledge (the witness is never seen by $\mathcal{S}$), and Verify requests embed both soundness and extractability (any verifying (x, π) either was honestly produced or $\mathcal{S}$ supplies a valid witness).

Functionality $\mathcal{F}_{\mathsf{PKI}}$ (public-key registry)

$\mathcal{F}_{\mathsf{PKI}}$ maintains a table T mapping party identities to public keys, initialized to $\emptyset$. It interacts with parties and with the simulator $\mathcal{S}$.

On input $(\texttt{Register}, \mathsf{pk})$ from a party P:

1. If $\mathsf{T}[P] \neq \perp$, send $\perp$ to P and **Break**. *(Each party may register at most once.)*
2. Set $\mathsf{T}[P] \leftarrow \mathsf{pk}$.
3. Send $(\texttt{Registered}, P, \mathsf{pk})$ to $\mathcal{S}$ and to all parties.

On input $(\texttt{Query}, P)$ from any party Q:

1. Send $\mathsf{T}[P]$ to Q (which equals $\perp$ when P has not registered).

Figure 11: Ideal functionality $\mathcal{F}_{\mathsf{PKI}}$ for a public-key registry.

B Proof of Theorem 1

Let $\mathcal{A}$ be an adversary that interacts with parties running the $\pi_{\mathsf{DC\text{-}ECPSS}}$ protocol in the $(\mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{SPP}}, \mathcal{F}_{\mathsf{NIZK}})$-hybrid model. Following the YOSO convention, in the abstract world we consider *static* corruptions: the adversary chooses arbitrary corruptions among the input/output roles (here, the dealer D_{sid}) and uniformly random corruptions among the computation roles (here, the committees $\{C_r^{\mathsf{sid}}\}_r$), all fixed before the execution starts.

We construct a simulator $\mathcal{S}$ that interacts with $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ and simulates $\mathcal{A}$ towards $\mathcal{Z}$. $\mathcal{S}$ runs $\mathcal{A}$ and the other parties internally and forwards all messages from $\mathcal{Z}$ to $\mathcal{A}$ and back. The environment can trigger functions of $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ by sending inputs to the dummy parties; $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ may then send messages to $\mathcal{S}$ which the latter uses for its simulation.

Proof. (Proof of Theorem 1) Fix the (static) corruption set chosen by $\mathcal{A}$: this determines, for every committee r, the corrupt set $\mathcal{B}_r \subseteq C_r^{\mathsf{sid}}$ and whether the dealer is honest or corrupt; $|\mathcal{B}_r|$ is below the threshold in every committee. Let T denote the lifetime of sid (number of committees).

We define a sequence of games H_0, H_1, H_2, H_3, H_4, H_5, H_6, $\mathsf{H}_7^{(0)}, \ldots, \mathsf{H}_7^{(T)}$, H_8, with $\mathsf{H}_0 = \mathrm{REAL}_{\pi_{\mathsf{DC\text{-}ECPSS}},\mathcal{A}}(1^\lambda)$ and $\mathsf{H}_8 = \mathrm{IDEAL}_{\mathcal{F}_{\mathsf{DC\text{-}ECPSS}},\mathcal{S}}(1^\lambda)$.

Game H_0 (Real execution). $\mathcal{A}$ interacts with honest parties running $\pi_{\mathsf{DC\text{-}ECPSS}}$ in the $(\mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{SPP}}, \mathcal{F}_{\mathsf{NIZK}})$-hybrid model. Calls to $\mathcal{F}_{\mathsf{NIZK}}$ are answered by the external ideal functionality; proof generation and witness extraction are handled internally by $\mathcal{F}_{\mathsf{NIZK}}$ via $\mathcal{S}_{\mathsf{NIZK}}$, invisibly to the protocol.

Game H_1 (Introduce the simulator). We introduce $\mathcal{S}$, which internally runs $\mathcal{A}$, executes the protocol on behalf of honest parties, and simulates $\mathcal{F}_{\mathsf{BC}}$ and $\mathcal{F}_{\mathsf{SPP}}$ internally; $\mathcal{F}_{\mathsf{NIZK}}$ is still treated by $\mathcal{S}$ as a black box: $\mathcal{S}$ relays honest parties' Prove/Verify inputs to the external $\mathcal{F}_{\mathsf{NIZK}}$ and relays the outputs back. Honest parties are replaced with dummy parties forwarding messages between $\mathcal{Z}$ and $\mathcal{S}$. This change is purely syntactic: $\mathsf{H}_0 \equiv \mathsf{H}_1$.

Game H_2 (Dealer-control via $\mathcal{F}_{\mathsf{BC}}$). In H_1, an honest party executing $(\mathsf{sid}, \mathsf{Ping})$ checks $\mathcal{F}_{\mathsf{BC}}$ for $(\mathsf{sid}, \mathtt{DealerOK}, r-1)$ from D_{sid}. In H_2, $\mathcal{S}$ instead consults its internal record of whether $(\mathsf{sid}, \mathtt{DealerOK})$ was input by $\mathcal{Z}$ to the (simulated) dealer. This is indistinguishable because $\mathcal{F}_{\mathsf{BC}}$ is authenticated: only D_{sid} can produce a $\mathtt{DealerOK}$ message under its identity, so $\mathcal{S}$'s internal record matches what $\mathcal{F}_{\mathsf{BC}}$ would deliver: $\mathsf{H}_1 \equiv \mathsf{H}_2$.

Game H_3 (Absorb $\mathcal{F}_{\mathsf{NIZK}}$ into $\mathcal{S}$). Until now, $\mathcal{F}_{\mathsf{NIZK}}$ has been external to $\mathcal{S}$: honest parties' Prove/Verify inputs went to the ideal functionality, which in turn communicated with its UC simulator $\mathcal{S}_{\mathsf{NIZK}}$ to produce proofs and extract witnesses. In H_3, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{NIZK}}$ itself as a subroutine, including the dialogue that $\mathcal{F}_{\mathsf{NIZK}}$ normally has with $\mathcal{S}_{\mathsf{NIZK}}$. Concretely, $\mathcal{S}$ now exercises the two $\mathcal{S}_{\mathsf{NIZK}}$-side interfaces of $\mathcal{F}_{\mathsf{NIZK}}$ directly:

- On every $(\mathsf{sid}, \mathsf{Prove}, \mathcal{R}, x, w)$ submitted by an honest party, $\mathcal{S}$ invokes $\mathcal{S}_{\mathsf{NIZK}}$ on input $(\mathcal{R}, x)$ to obtain a simulated proof π—*without using the witness* w—and returns π to the honest party while recording $(\mathcal{R}, x, \pi)$ as honest in the internal table Π.
- On every $(\mathsf{sid}, \mathsf{Verify}, \mathcal{R}, x, \pi)$ where π is not recorded as honest in Π (i.e., π was produced by a corrupt prover), $\mathcal{S}$ invokes $\mathcal{S}_{\mathsf{NIZK}}$ to obtain a witness $w \in \mathcal{R}$ and stores it.

The change is syntactic from the environment's perspective: the joint distribution of messages on the $(\mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{SPP}}, \mathcal{F}_{\mathsf{NIZK}})$ interface and of $\mathcal{A}$'s view is exactly what external $\mathcal{F}_{\mathsf{NIZK}}$ would have produced in H_2, since the only difference is *who* runs $\mathcal{S}_{\mathsf{NIZK}}$ (the external $\mathcal{F}_{\mathsf{NIZK}}$ vs. our $\mathcal{S}$). Hence $\mathsf{H}_2 \equiv \mathsf{H}_3$.

Game H_4 (Commitment binding). By soundness of $\mathcal{F}_{\mathsf{NIZK}}$ (Figure 10), every verifying $\mathcal{R}^{\mathsf{Reshare}}/\mathcal{R}^{\mathsf{Share}}$ proof is accompanied by a witness in the corresponding relation. In particular, from a corrupt predecessor $C_{r-1}^{\mathsf{sid}}[i]$'s verifying $\mathcal{R}^{\mathsf{Reshare}}$ proof, $\mathcal{S}$ has extracted (in H_3) a witness $\big(\{\mathsf{sh}'_l, \rho'_l\}_{l\in\mathcal{I}'}, \mathsf{sh}', \{\mathsf{rsh}'_k, \rho'^{\mathrm{out}}_k\}_k\big)$ such that the broadcast commitments $\{c_{i,k}\}_k$ are openings to $\{\mathsf{rsh}'_k\}_k$, $\mathsf{sh}' = \mathsf{Takeover}(\{\mathsf{sh}'_l\}_{l\in\mathcal{I}'})$, and $\{\mathsf{rsh}'_k\} = \mathsf{Handover}(\mathsf{sh}')$. Binding of Com is *not* guaranteed by $\mathcal{F}_{\mathsf{NIZK}}$, however, and must be argued separately: nothing in $\mathcal{F}_{\mathsf{NIZK}}$ prevents a corrupt prover from privately delivering, via $\mathcal{F}_{\mathsf{SPP}}$, a value sh_i different from the extracted rsh'_j while still passing an honest recipient's commitment-equality check $c_{i,j} = \mathsf{Com}(\mathsf{sh}_i; \rho^{\mathrm{in}}_{i,j})$.

In H_4, $\mathcal{S}$ explicitly aborts and outputs a distinguished symbol $\perp_{\mathsf{bad}}$ if the following bad event occurs during the simulation:

E_{bind}: some sh_i delivered via $\mathcal{F}_{\mathsf{SPP}}$ to an honest recipient passes the equality check $c_{i,j} = \mathsf{Com}(\mathsf{sh}_i; \rho^{\text{in}}_{i,j})$, but the witness extracted in H_3 from π^{in}_i contains a value $\mathsf{rsh}'_j \neq \mathsf{sh}_i$ at position j.

Occurrence of E_{bind} yields two distinct openings of $c_{i,j}$, contradicting computational binding of Com. Hence $\Pr[E_{\text{bind}}] \in \mathsf{negl}(\lambda)$, and $\mathsf{H}_3 \approx_c \mathsf{H}_4$.
After this game, on the indices the honest committee retained as valid, the shares fed into $\mathsf{Takeover}$ coincide with the values *extracted from* the corrupt provers' $\mathcal{R}^{\mathsf{Reshare}}/\mathcal{R}^{\mathsf{Share}}$ witnesses, which by definition of those relations are valid $\mathsf{Handover}$ outputs of a well-defined local share sh' (resp., a valid Share output of a well-defined s^*). Inductively over $r = 1, \ldots, T$, the local share that an honest committee member at r obtains via $\mathsf{Takeover}$ is a valid share, under SSS, of the same secret as in committee $r-1$, anchored at $r = 1$ either to the honest dealer's input s or to the unique s^* extracted from a corrupt dealer's $\mathcal{R}^{\mathsf{Share}}$ proof.

Game H_5 (Hide honest-to-honest openings sent over $\mathcal{F}_{\mathsf{SPP}}$). In H_4, every honest sender (the dealer if honest, every honest re-sharing committee member) commits on $\mathcal{F}_{\mathsf{BC}}$ to its outgoing shares and delivers the openings via $\mathcal{F}_{\mathsf{SPP}}$ to the next committee. By privacy of $\mathcal{F}_{\mathsf{SPP}}$, openings whose recipient is honest are not seen by $\mathcal{A}$. In H_5, for every $(\mathsf{sid}, \mathtt{Share}, r, \mathsf{rsh}_k, \rho_k)$ destined to an honest recipient $C^{\mathsf{sid}}_r[k]$, $\mathcal{S}$ replaces the broadcast commitment by $\mathsf{Com}(\widetilde{\mathsf{rsh}}_k; \rho_k)$ for a fresh dummy $\widetilde{\mathsf{rsh}}_k$, while internally bookkeeping the real rsh_k. Commitments destined to corrupt recipients in $\mathcal{B}_r$ remain honest commitments to the real shares, and the real opening is delivered unchanged via $\mathcal{F}_{\mathsf{SPP}}$.

This change is indistinguishable by hiding of Com, applied independently to each honest-to-honest commitment. The step is sound precisely because in H_3 we absorbed $\mathcal{F}_{\mathsf{NIZK}}$ into $\mathcal{S}$: the accompanying $\mathcal{R}^{\mathsf{Reshare}}$ proof is produced by $\mathcal{S}_{\mathsf{NIZK}}$ without reference to the committed values, so it remains valid even though the committed value has changed. Hence $\mathsf{H}_4 \approx_c \mathsf{H}_5$.

Game H_6 (Share-simulatable release). After H_5, the honest-side shares $\{\mathsf{sh}_{r,k}\}_{k \notin \mathcal{B}_r}$ are still computed by the real $\mathsf{Takeover}/\mathsf{Handover}$ chain and surface in the adversary's view only at release: at the terminal committee $r^\star$, each honest member $C^{\mathsf{sid}}_{r^\star}[k]$ broadcasts $(\mathsf{sid}, \mathtt{Finish}, r^\star, \mathsf{mode}(\mathsf{sh}_{r^\star,k}))$.

In H_6, $\mathcal{S}$ *stops* computing those honest-side shares via the real chain and instead, at the moment of release, invokes share simulatability:

$$\{\mathsf{sh}'_{r^\star,k}\}_{k \notin \mathcal{B}_{r^\star}} \leftarrow \mathsf{SSS.ShareSim}\big(\{\mathsf{sh}_{r^\star,k}\}_{k \in \mathcal{B}_{r^\star}},\, s\big),$$

where s is the honest dealer's stored secret (or s^* extracted in H_3 from a corrupt dealer's $\mathcal{R}^{\mathsf{Share}}$ proof). $\mathcal{S}$ then broadcasts $(\mathsf{sid}, \mathtt{Finish}, r^\star, \mathsf{mode}(\mathsf{sh}'_{r^\star,k}))$ on behalf of each honest member instead.

By share simulatability of SSS (Definition 5) applied to the unauthorized set $\mathcal{B}_{r^\star}$ with target secret s, the joint distribution $\big(\{\mathsf{sh}_{r^\star,k}\}_{k \in \mathcal{B}_{r^\star}}, \{\mathsf{sh}'_{r^\star,k}\}_{k \notin \mathcal{B}_{r^\star}}\big)$ is identical to a fresh $\mathsf{Share}(s)$, which is exactly the distribution of the real honest-side shares in H_5 by re-sharing correctness. Hence $\mathsf{H}_5 \equiv \mathsf{H}_6$.

The point of this hop is to decouple the release broadcasts from the intermediate $\mathsf{Takeover}/\mathsf{Handover}$ chain: from now on the only place s is read by $\mathcal{S}$ is the $\mathsf{ShareSim}$ call at release, freeing the chain itself to be re-randomized in the next hops.

Games $\mathsf{H}^{(0)}_7, \ldots, \mathsf{H}^{(T)}_7$ (Replace the secret sharing chain with fresh independent randomness). At this point, the only remaining dependence on s in the adversary's view is in the corrupt-side shares $\{\mathsf{sh}_{r,k}\}_{r,\, k \in \mathcal{B}_r}$ and their commitments delivered via $\mathcal{F}_{\mathsf{SPP}}$. We strip this dependence one committee at a time, processing committees in *reverse* order $r = T, T-1, \ldots, 1$. At each step, the input share fed to the honest sender's $\mathsf{Handover}$ is replaced by a *fresh independent* share, so that across consecutive committees the adversary observes shares drawn from unrelated randomness.

Set $\mathsf{H}^{(0)}_7 := \mathsf{H}_6$. For $k = 1, \ldots, T$, let $r := T - k + 1$. $\mathsf{H}^{(k)}_7$ differs from $\mathsf{H}^{(k-1)}_7$ only at committee r:

- If $r \geq 2$, instead of producing the corrupt-side shares $\{\mathsf{sh}_{r,k'}\}_{k' \in \mathcal{B}_r}$ as the $\mathcal{B}_r$-marginal of $\mathsf{Handover}(\mathsf{sh}_{r-1}; \rho_{\mathsf{Handover}})$, where sh_{r-1} was the honest sender's real local share at committee $r-1$, $\mathcal{S}$ samples a fresh

share $\widetilde{\mathsf{sh}}_{r-1}$, independent of all other randomness and uniformly drawn from the share space of SSS, and uses $\mathsf{Handover}(\widetilde{\mathsf{sh}}_{r-1}; \widetilde{\rho}_{\mathsf{Handover}})$ with a fresh tape $\widetilde{\rho}_{\mathsf{Handover}}$.

- If $r = 1$ and the dealer is honest, $\mathcal{S}$ samples a fresh secret $s' \leftarrow\!\!\$\ K$ and replaces $\mathsf{Share}(s; \rho_{\mathsf{Share}})$ with $\mathsf{Share}(s'; \widetilde{\rho}_{\mathsf{Share}})$, exposing $\{\mathsf{sh}_{1,k'}\}_{k' \in \mathcal{B}_1}$ as the corresponding corrupt-side marginal. (If the dealer is corrupt, the $r = 1$ shares are the adversary's own choice and the hop is not applied.)

The corresponding commitments destined to corrupt recipients are recomputed honestly on the new values; honest-to-honest commitments remain dummies (from H_5); NIZK proofs remain simulated (from H_3). The $\mathsf{ShareSim}$ call at release introduced in H_6 is untouched.

When $\mathcal{S}$ reaches committee r at hop $k = T - r + 1$, committees $r+1, \ldots, T$ have already been replaced with fresh independent randomness and no longer reference sh_{r-1} or its descendants. Replacing the input share to committee r's $\mathsf{Handover}$ therefore affects only committee r's own corrupt-side view, with no cascade into later committees.

Indistinguishability $\mathsf{H}_7^{(k-1)} \equiv \mathsf{H}_7^{(k)}$ at the hop that touches committee r is information-theoretic:

- For $r \geq 2$, by re-sharing privacy of SSS (Section 2.2) applied to $\mathcal{B}_r$, the marginal of $\mathsf{Handover}(\mathsf{sh})$ on $\mathcal{B}_r$ is identical for every input share sh; replacing sh_{r-1} with the fresh independent $\widetilde{\mathsf{sh}}_{r-1}$ therefore preserves the corrupt-side distribution exactly.
- For $r = 1$, by privacy of SSS (Definition 4) applied to $\mathcal{B}_1$, the marginals of $\mathsf{Share}(s)$ and $\mathsf{Share}(s')$ on $\mathcal{B}_1$ are identical.

By a hybrid argument over the T committees, $\mathsf{H}_6 = \mathsf{H}_7^{(0)} \equiv \mathsf{H}_7^{(T)}$.

After $\mathsf{H}_7^{(T)}$, every corrupt-side share and every corrupt-side commitment is a function of fresh randomness independent of s, and the shares the adversary observes at any two consecutive committees are independent. The only place s enters $\mathcal{S}$'s computation is the $\mathsf{ShareSim}$ call at release introduced in H_6.

Game H_8 (Ideal world). We now route all secret handling through $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$.

- When $\mathcal{Z}$ sends $(\mathsf{sid}, \texttt{StoreSecret}, s, \mathsf{mode})$ to an honest dealer, $\mathcal{S}$ forwards it to $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ and removes s from its local state until the release step. By $\mathsf{H}_7^{(T)}$, no operation before release depends on s, so this removal is undetectable.
- If the dealer is corrupt, $\mathcal{S}$ acts as the dealer's envelope: when the corrupt dealer broadcasts its $\texttt{Stored}$ message together with a verifying $\mathcal{R}^{\mathsf{Share}}$ proof, $\mathcal{S}$ uses the secret s^* extracted in H_3 and inputs $(\mathsf{sid}, \texttt{StoreSecret}, s^*, \mathsf{mode})$ to $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ on the corrupt dealer's behalf. If no verifying proof is produced (or the dealer aborts), $\mathcal{S}$ forwards nothing, and both worlds stay in their initial state.
- Each round, when $\mathcal{Z}$ activates the honest parties with $(\mathsf{sid}, \mathsf{Ping})$, $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ sends $(\mathsf{sid}, \texttt{Pinged})$ to the dealer and $\mathcal{S}$ drives $\mathcal{A}$'s view accordingly. Whether release occurs is decided by $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ from $\mathcal{Z}$'s $(\mathsf{sid}, \texttt{DealerOK})$ inputs, exactly mirroring $\mathcal{S}$'s internal check from H_2.
- Upon release, $\mathcal{S}$ obtains $\mathsf{mode}(s)$ from $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ and uses it as the target of the $\mathsf{ShareSim}$ call introduced in H_6: $\{\mathsf{sh}'_{r^\star,k}\}_{k \notin \mathcal{B}_{r^\star}} \leftarrow \mathsf{SSS.ShareSim}(\{\mathsf{sh}_{r^\star,k}\}_{k \in \mathcal{B}_{r^\star}}, s)$, and broadcasts the corresponding $\texttt{Finish}$ messages for honest members on the simulated $\mathcal{F}_{\mathsf{BC}}$. By H_4 and re-sharing correctness, any honest party invoking $(\mathsf{sid}, \texttt{ReadSecret})$ in the simulated execution reconstructs $\mathsf{mode}(s)$, exactly what $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ delivers.

The transition $\mathsf{H}_7^{(T)} \to \mathsf{H}_8$ is syntactic for an honest dealer: s now lives inside $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ rather than in $\mathcal{S}$'s local state, and $\mathcal{S}$ retrieves $\mathsf{mode}(s)$ from $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ at exactly the moment $\mathsf{H}_7^{(T)}$ would have read it locally. For a corrupt dealer, indistinguishability follows from the extraction guarantee of $\mathcal{F}_{\mathsf{NIZK}}$ in H_3 together with H_4: the extracted s^* is, with overwhelming probability, the unique value that an honest chain would reconstruct.

Combining the above:

$$\mathsf{H}_0 \equiv \mathsf{H}_1 \equiv \mathsf{H}_2 \equiv \mathsf{H}_3 \approx_c \mathsf{H}_4 \approx_c \mathsf{H}_5 \equiv \mathsf{H}_6 \equiv \mathsf{H}_7^{(1)} \equiv \cdots \equiv \mathsf{H}_7^{(T)} \approx_c \mathsf{H}_8,$$

from where the claim follows.

□

C Proof of Theorem 2

We construct a simulator $\mathcal{S}$ that interacts with $\mathcal{F}_{\mathsf{DMS}}$ and simulates $\mathcal{A}$ towards $\mathcal{Z}$, in the corruption model of Section 4.3. $\mathcal{S}$ runs $\mathcal{A}$ and the (simulated) honest parties internally and forwards all messages between $\mathcal{Z}$ and $\mathcal{A}$. $\mathcal{S}$ also simulates the sub-functionalities $\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{DC\text{-}ECPSS}}, \mathcal{F}_{\mathsf{PKI}}$ internally, absorbing their UC simulators $\mathcal{S}_{\mathsf{NIZK}}$ and $\mathcal{S}_{\mathsf{DC\text{-}ECPSS}}$. In particular: on a Prove call from an honest party $\mathcal{S}$ invokes $\mathcal{S}_{\mathsf{NIZK}}$ to obtain a simulated proof without using the witness, and on a Verify call of an unrecorded proof $\mathcal{S}$ invokes $\mathcal{S}_{\mathsf{NIZK}}$ to extract a witness. For each honest party $P \in \mathcal{U}_{\mathcal{N}}$, $\mathcal{S}$ simulates the $(\mathsf{sid}, \texttt{Init})$ behavior by sampling $(\mathsf{sk}_P, \mathsf{pk}_P) \leftarrow \mathsf{mPKE.Gen}$ and registering pk_P in $\mathcal{S}$'s internal $\mathcal{F}_{\mathsf{PKI}}$ table; corrupt nominees' registrations are driven by $\mathcal{A}$.

Proof. (Proof of Theorem 2) Fix the dealer's corruption status and the static set of corrupt nominee parties; for each sid, $\mathcal{F}_{\mathsf{DMS}}$ will deliver $\mathcal{B}_{\mathcal{N}} := \mathcal{N} \cap \mathsf{Corrupt}$ to $\mathcal{S}$ on the corresponding $\texttt{StoreSecret}$ event. We argue by cases on whether the dealer is honest or corrupt, and (if honest) whether $\mathcal{B}_{\mathcal{N}} = \emptyset$.

Case A (Corrupt dealer). When the corrupt dealer performs $\texttt{StoreSecret}$, $\mathcal{S}$ observes the broadcast $(\mathsf{sid}, \texttt{Stored}, \mathcal{R}, (c_1, c_2), \pi)$ on $\mathcal{F}_{\mathsf{BC}}$ and the input $(\mathsf{sid}, \texttt{StoreSecret}, c_0, w_{\mathcal{R}^{\mathsf{Store}}}, \mathcal{R}^{\mathsf{Store}}_{c_1,c_2}, \mathsf{release})$ to (simulated) $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$, with $w_{\mathcal{R}^{\mathsf{Store}}} = (\mathsf{esk}^*, \{\mathsf{pk}\}^*_{\mathcal{N}}, R^*)$. $\mathcal{S}$ verifies π on $\mathcal{F}_{\mathsf{NIZK}}$. If verification fails or $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ rejects $\mathcal{R}^{\mathsf{Store}}_{c_1,c_2}(c_0, w_{\mathcal{R}^{\mathsf{Store}}})$, no honest party will accept the dealer's broadcast at the first Ping and the real execution stalls; $\mathcal{S}$ does not forward $\texttt{StoreSecret}$ to $\mathcal{F}_{\mathsf{DMS}}$, matching the ideal world.

Otherwise, by $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$'s perfect $\mathcal{R}^{\mathsf{Store}}$ check,

$$c_1 = \mathsf{mPKE.Enc}(\{\mathsf{pk}\}^*_{\mathcal{N}}, \mathsf{esk}^*; R^*), \quad s^{\mathcal{R}^{\mathsf{Store}}} = \mathsf{SKE.Dec}\big(\mathsf{PKE.Dec}(\mathsf{esk}^*, c_0),\, c_2\big) \neq \bot,$$

and by simulation-extractability of $\mathcal{F}_{\mathsf{NIZK}}$, $\mathcal{S}$ obtains a witness $(K^{\mathcal{R}^{\mathsf{Enc}}}, s^{\mathcal{R}^{\mathsf{Enc}}}, w'^{\mathcal{R}^{\mathsf{Enc}}})$ for $\mathcal{R}^{\mathsf{Enc}}_{\mathcal{R}}$ on c_2 with $\mathcal{R}(s^{\mathcal{R}^{\mathsf{Enc}}}, w'^{\mathcal{R}^{\mathsf{Enc}}}) = 1$ and $c_2 = \mathsf{SKE.Enc}(K^{\mathcal{R}^{\mathsf{Enc}}}, s^{\mathcal{R}^{\mathsf{Enc}}})$, hence $\mathsf{SKE.Dec}(K^{\mathcal{R}^{\mathsf{Enc}}}, c_2) = s^{\mathcal{R}^{\mathsf{Enc}}} \neq \bot$ by SKE correctness. Both $K^{\mathcal{R}^{\mathsf{Enc}}}$ and $\mathsf{PKE.Dec}(\mathsf{esk}^*, c_0)$ are therefore keys under which c_2 decrypts to a non-$\bot$ value; by key-commitment of SKE,

$$K^{\mathcal{R}^{\mathsf{Enc}}} = \mathsf{PKE.Dec}(\mathsf{esk}^*, c_0), \quad \text{and hence} \quad s^{\mathcal{R}^{\mathsf{Enc}}} = s^{\mathcal{R}^{\mathsf{Store}}}.$$

Let $\mathcal{N}^* \subseteq \mathcal{U}_{\mathcal{N}}$ be the set of parties whose $\mathcal{F}_{\mathsf{PKI}}$-registered public keys appear in $\{\mathsf{pk}\}^*_{\mathcal{N}}$ (any entries of $\{\mathsf{pk}\}^*_{\mathcal{N}}$ that do not correspond to a registered nominee are simply ignored, since the corresponding $\mathsf{mPKE.Dec}$ on c_1 yields $\bot$ in the real protocol and therefore does not yield any honest-recipient output). $\mathcal{S}$ inputs $(\mathsf{sid}, \texttt{StoreSecret}, s^{\mathcal{R}^{\mathsf{Enc}}}, w'^{\mathcal{R}^{\mathsf{Enc}}}, \mathcal{R}, \mathcal{N}^*)$ to $\mathcal{F}_{\mathsf{DMS}}$ on the corrupt dealer's behalf; $\mathcal{F}_{\mathsf{DMS}}$ accepts since $\mathcal{R}(s^{\mathcal{R}^{\mathsf{Enc}}}, w'^{\mathcal{R}^{\mathsf{Enc}}}) = 1$.

For subsequent epochs, $\mathcal{S}$ relays Ping / $\texttt{DealerOK}$ between $\mathcal{F}_{\mathsf{DMS}}$ and $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ verbatim: the dealer-control state machines $\{\mathsf{Pinged}, \mathsf{DResponse}, \mathsf{Released}\}$ are identical in the two functionalities (Figures 2 and 5). At release, an honest party calling $\texttt{ReadSecret}$ on $\mathcal{F}_{\mathsf{DMS}}$ receives $s^{\mathcal{R}^{\mathsf{Enc}}}$ iff it lies in $\mathcal{N}^*$; in the real protocol the same party recovers esk^* from c_1 (via its $\mathcal{F}_{\mathsf{PKI}}$-registered key pair stored at $\texttt{Init}$), decrypts c_0 with esk^* to obtain $K^{\mathcal{R}^{\mathsf{Enc}}}$, and decrypts c_2 to obtain $s^{\mathcal{R}^{\mathsf{Enc}}} = s^{\mathcal{R}^{\mathsf{Store}}}$.

Case B (Honest dealer, all nominees honest). $\mathcal{A}$ knows neither esk nor any $\mathsf{sk}_{\mathsf{nom}}$. Let $\mathsf{H}_0 = \mathrm{REAL}_{\pi_{\mathsf{DMS}},\mathcal{A}}(1^\lambda)$.

Game H_1 (Simulate the dealer's NIZK). $\mathcal{S}$ simulates the dealer's $\mathcal{R}^{\mathsf{Enc}}_{\mathcal{R}}$ proof on $\mathcal{F}_{\mathsf{BC}}$ via $\mathcal{S}_{\mathsf{NIZK}}$ without using the witness. Indistinguishable by zero-knowledge of $\mathcal{F}_{\mathsf{NIZK}}$: $\mathsf{H}_0 \equiv \mathsf{H}_1$.

Game H_2 (Anonymize c_1). Replace $c_1 = \mathsf{mPKE.Enc}(\{\mathsf{pk}\}_{\mathcal{N}}, \mathsf{esk}; R)$ by $c_1' = \mathsf{mPKE.Enc}(\{\widetilde{\mathsf{pk}_i}\}_{i\in[|\mathcal{N}|]}, \perp; \widetilde{R})$ for arbitrary dummy public keys $\widetilde{\mathsf{pk}_i}$ and dummy plaintext $\perp$. Indistinguishable by IND-Anon-CPA of mPKE (no $\mathsf{sk}_{\mathsf{nom}}$ is in $\mathcal{A}$'s hands): $\mathsf{H}_1 \approx_c \mathsf{H}_2$.

Game H_3 (Decouple c_0 from K). Replace $c_0 = \mathsf{PKE.Enc}(\mathsf{epk}, K; R_0)$ by $c_0' = \mathsf{PKE.Enc}(\mathsf{epk}, \widetilde{K}; \widetilde{R}_0)$ for a fresh uniformly random $\widetilde{K} \leftarrow\!\$\ \{0,1\}^{|K|}$, while keeping the same K inside c_2. Indistinguishable by IND-CPA of PKE: $\mathcal{A}$ eventually sees c_0 (at release) but never holds esk, since c_1' from H_2 no longer carries esk to anyone $\mathcal{A}$ can decrypt for. Hence $\mathsf{H}_2 \approx_c \mathsf{H}_3$, and K (the key used inside c_2) is now information-theoretically independent of $\mathcal{A}$'s view.

Game H_4 (Randomize c_2). Replace $c_2 = \mathsf{SKE.Enc}(\widetilde{K}, s)$ by a uniformly random c_2' of the appropriate length. Indistinguishable by one-time SKE security: $\mathsf{H}_3 \approx_c \mathsf{H}_4$.

Game H_5 (Ideal world). $\mathcal{S}$ no longer uses s to construct $\mathcal{A}$'s view: π is simulated, c_1' is independent of s, and c_2' is uniform. $\mathcal{S}$ forwards $\mathtt{StoreSecret}(s, w, \mathcal{R}, \mathcal{N})$ from $\mathcal{Z}$ to $\mathcal{F}_{\mathsf{DMS}}$ exactly as the protocol does to $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ in the hybrid model. At release, honest nominees calling `ReadSecret` on $\mathcal{F}_{\mathsf{DMS}}$ obtain s directly; their internal state is not visible to $\mathcal{A}$, so bypassing the protocol's local recovery (which on c_1', c_2' would not yield s) is undetectable. The Ping / `DealerOK` traffic is relayed verbatim through $\mathcal{S}_{\mathsf{DC\text{-}ECPSS}}$; $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ hides c_0 until release, and the release-time broadcast of c_0 is harmless because $\mathcal{A}$ holds no information on esk. Hence $\mathsf{H}_4 \equiv \mathsf{H}_5$, and $\mathsf{H}_5 = \mathrm{IDEAL}_{\mathcal{F}_{\mathsf{DMS}},\mathcal{S}}(1^\lambda)$.

Combining the above, in Case B:

$$\mathsf{H}_0 \equiv \mathsf{H}_1 \approx_c \mathsf{H}_2 \approx_c \mathsf{H}_3 \approx_c \mathsf{H}_4 \equiv \mathsf{H}_5.$$

Case C (Honest dealer, at least one corrupt nominee). $\mathcal{S}$ learns $\mathcal{B}_{\mathcal{N}}$ from the $(\mathsf{sid}, P_j, \mathtt{Stored}, \mathcal{R}, \mathcal{B}_{\mathcal{N}})$ notification $\mathcal{F}_{\mathsf{DMS}}$ delivers at the honest dealer's `StoreSecret`, before any Ping has fired. Let $\mathsf{H}_0 = \mathrm{REAL}_{\pi_{\mathsf{DMS}},\mathcal{A}}(1^\lambda)$.

Game H_1 (Simulate the dealer's NIZK). As in Case B: $\mathsf{H}_0 \equiv \mathsf{H}_1$.

Game H_2 (Restrict c_1 to corrupt nominees). Replace $c_1 = \mathsf{mPKE.Enc}(\{\mathsf{pk}\}_{\mathcal{N}}, \mathsf{esk}; R)$ by

$$c_1' = \mathsf{mPKE.Enc}(\{\mathsf{pk}_i\}_{i\in\mathcal{B}_{\mathcal{N}}}, \mathsf{esk}; \widetilde{R})$$

— encryption of esk *only* to the corrupt subset. Indistinguishable by IND-Anon-CPA of mPKE applied to swapping out the honest portion of the recipient set; corrupt nominees still decrypt c_1' to the same esk by mPKE correctness. $\mathsf{H}_1 \approx_c \mathsf{H}_2$.

Game H_3 (Randomize c_2, defer c_0). $\mathcal{S}$ samples $c_2' \leftarrow\!\$\ \{0,1\}^{|c_2|}$ uniformly at `StoreSecret` time and uses c_2' on the broadcast in place of $\mathsf{SKE.Enc}(K, s)$. $\mathcal{S}$ does *not* compute the PKE-ciphertext c_0 yet: it simply lets the (simulated) $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ record the `StoreSecret` event with a placeholder for the stored value. Throughout the lifetime of the secret, $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ does not reveal c_0 to $\mathcal{A}$, so the deferral is invisible. On the Ping that triggers release, $\mathcal{S}$ computes $K^\star \leftarrow \mathsf{Ext}_{\mathsf{SKE}}(c_2', s)$, the unique key such that $\mathsf{SKE.Enc}(K^\star, s) = c_2'$, and then computes $c_0 \leftarrow \mathsf{PKE.Enc}(\mathsf{epk}, K^\star; R_0)$ with fresh randomness R_0 (using the $(\mathsf{esk}, \mathsf{epk})$ pair $\mathcal{S}$ generated on behalf of the honest dealer at `StoreSecret` time). $\mathcal{F}_{\mathsf{DC\text{-}ECPSS}}$ broadcasts this c_0 on $\mathcal{F}_{\mathsf{BC}}$ via the holding committee's `Finish` messages; a corrupt nominee running real-world recovery obtains $\mathsf{mPKE.Dec}(\mathsf{sk}_{\mathsf{nom}}, c_1') \to \mathsf{esk}$, $\mathsf{PKE.Dec}(\mathsf{esk}, c_0) \to K^\star$ by PKE correctness, and $\mathsf{SKE.Dec}(K^\star, c_2') = s$ by choice of $K^\star$.

Indistinguishability $\mathsf{H}_2 \equiv \mathsf{H}_3$ is statistical. In both games, $\mathcal{A}$'s joint view consists of a ciphertext c_2 and, at release, the value $K = \mathsf{PKE.Dec}(\mathsf{esk}, c_0)$, subject to the consistency constraint $\mathsf{SKE.Dec}(K, c_2) = s$. In H_2 the dealer samples $K \leftarrow\!\$\ \{0,1\}^\lambda$ uniformly and sets $c_2 = \mathsf{SKE.Enc}(K, s)$, $c_0 = \mathsf{PKE.Enc}(\mathsf{epk}, K; R_0)$; in H_3, $\mathcal{S}$ samples c_2' uniformly, sets $K^\star = \mathsf{Ext}_{\mathsf{SKE}}(c_2', s)$ (uniformly distributed by key-commitment of SKE), and sets $c_0 = \mathsf{PKE.Enc}(\mathsf{epk}, K^\star; R_0)$. In both games the marginal on (c_2, K) is uniform over $\{(c, K) \mid \mathsf{SKE.Dec}(K, c) = s\}$, and c_0 is in both cases a fresh PKE encryption of the (uniform) K.

Game H_4 (Ideal world). $\mathcal{S}$ now routes secret handling through $\mathcal{F}_{\mathsf{DMS}}$: when $\mathcal{Z}$ hands $\mathtt{StoreSecret}(s, w, \mathcal{R}, \mathcal{N})$ to the (dummy) honest dealer, the input is forwarded to $\mathcal{F}_{\mathsf{DMS}}$, and $\mathcal{S}$ does not learn s at this point. By H_3,

the only place $\mathcal{S}$ uses s is the $\mathsf{Ext}_{\mathsf{SKE}}$ call at release together with the subsequent c_0-construction; both of these happen *after* $\mathcal{F}_{\mathsf{DMS}}$ delivers $(\mathsf{sid}, s, \mathcal{N})$ to $\mathcal{S}$ on the release-triggering Ping, so $\mathcal{S}$ has s at exactly the moment it needs it. Honest nominees' `ReadSecret` outputs are returned directly by $\mathcal{F}_{\mathsf{DMS}}$. The change is syntactic: $\mathsf{H}_3 \equiv \mathsf{H}_4$, and $\mathsf{H}_4 = \mathrm{IDEAL}_{\mathcal{F}_{\mathsf{DMS}},\mathcal{S}}(1^\lambda)$.

Combining the above, in Case C:

$$\mathsf{H}_0 \equiv \mathsf{H}_1 \approx_c \mathsf{H}_2 \equiv \mathsf{H}_3 \equiv \mathsf{H}_4.$$

The claim follows. □