Dynamic Multi-Server Updatable Encryption

Jodie Knapp[1,2][0000-0002-5929-2015] and Elizabeth A. Quaglia[2]
[0000-0002-4010-773X]

[1] University of Surrey
[2] Information Security Group, Royal Holloway, University of London,
j.knapp@surrey.ac.uk, elizabeth.quaglia@rhul.ac.uk

Abstract. In this paper, we propose the *Dynamic Multi-Server* Updatable Encryption (DMUE) primitive as an extension of standard public-key updatable encryption. Traditional UE aims to have efficient ciphertext updates performed by an untrusted server such that the compromise of several cryptographic keys and update tokens does not reduce the standard security of encryption. The update token supports outsourced ciphertext updates without requiring the server to decrypt and re-encrypt the ciphertext and it is typically derived from old and new keys. To mitigate the risk of a single point of failure in single-server UE and thus improve the resilience of the scheme, we formalise a multi-server variant of UE to treat the issue of token leakage. We can achieve a distributed update process by providing each server with an update token and requiring a *threshold* of servers to engage honestly. However, servers may act dishonestly or need to be replaced over time, so our primitive must cater to dynamic committee changes in the servers participating across epochs. Inspired by the work of Benhamouda et al. (TCC'20) on dynamic proactive secret sharing, we propose a *generic* DMUE scheme built from public-key UE and dynamic proactive secret sharing primitives and prove the ciphertext unlinkability of freshly encrypted versus updated ciphertexts.

Keywords: Public-Key Updatable Encryption, Dynamic Committees, Threshold Secret Sharing, Trust Management, and Security.

1 Introduction

Outsourcing encrypted data is a common practice for individuals and organisations wanting to store their information in a secure manner over long periods. Yet the server storing the information cannot always be trusted and there is greater opportunity for an adversary to corrupt the cryptographic key used for encryption. One solution to managing security in this setting is the updatable encryption (UE) primitive [6, 14, 22], which is utilised for privacy preservation in multiple applications such as cloud storage; online medical information and blockchain technology. Informally, UE allows a data owner to outsource the storage and key rotation of ciphertexts, from one epoch to the next, to an untrusted server. The server updates the ciphertext using an update token derived from

old and new cryptographic keys, which *evolve* with every *epoch*, such that they do not learn anything about the underlying information in the update process. Updatable encryption is traditionally viewed as a symmetric primitive, however, more recently it has been defined in the *public-key* setting (PKUE) [20] which has allowed research to explore outsourced ciphertext updates whereby the users can receive messages from multiple senders directly in the cloud environment [1, 16, 26]. We focus on the PKUE primitive in this work.[3]

The core purpose of (PK)UE is to reduce the impact of key exposure and, in turn, token exposure, preserving standard encryption security such as confidentiality and the updatable notion of *unlinkability* [19]. Despite efforts to increase security in any UE setting, there remain risks with respect to security and resilience. The most prevalent risk is a *single point of failure* if the server is corrupted by an external adversary. In this scenario, a data owner's encrypted data will remain encrypted under the same key, defeating the purpose of UE as an adversary has more time in which to corrupt the cryptographic key and learn the underlying message. A second possible scenario occurs when the server acts dishonestly in the sense of failure to update ciphertexts correctly, if at all. If the ciphertext is updated incorrectly then the data owner may be misled upon decrypting the ciphertext. To illustrate, if the encrypted data is regarding a personal financial account and the update is incorrect then the data owner may be misinformed about the amount of money in their account.

A natural solution to this issue is to *distribute* the token, used to update ciphertexts, across *multiple servers* such that some pre-defined *threshold* of servers can ensure the ciphertext is updated in each epoch. This solution works, but a *static* set of servers does not reflect the real world because servers often change over long periods or possibly need to be removed from a scheme due to dishonest behaviour. To illustrate, suppose we wish to store a secret on a public-key blockchain such that nodes of the blockchain structure are considered to be the servers in a multi-server UE scheme. The authors of [3] demonstrated that *node churning* needs to be taken into consideration when designing a scheme for this application. This led us to propose a multi-server PKUE primitive supporting a *dynamic committee* of servers from one epoch to the next. We call this primitive dynamic multi-server updatable encryption (DMUE) such that the ciphertext update process is designed to be *deterministic* and *ciphertext-independent*. We note that the approach of having an evolving committee of servers is similar to previous works such as [2, 21, 23, 28].

More concretely, DMUE captures servers in specific epochs each possessing an update token, whereby their tokens are utilised in the process of updating a ciphertext. Moreover, the committee of servers in consecutive epochs may differ and so a *redistribution protocol* is required to provide new servers with their

[3] The authors of [11, 12] established definitions for updatable *public-key* encryption (UPKE) using an alternative update procedure. One can view UPKE as a distinct primitive to PKUE [20] since the token mechanism used in a UPKE scheme only updates the public key. By contrast, PKUE updates public and secret key pairs as well as the ciphertext.

corresponding tokens. Defining the security of a DMUE primitive proves to be challenging and nuanced due to the bi-directional nature of key updates [18, 24] in the design of a PKUE scheme. Moreover, the inference of keys from tokens is further complicated in the multi-server setting since an adversary can only succeed in their attack if they corrupt a *threshold* of server tokens in any given time period, with the servers and threshold potentially evolving with each epoch. Note, the adversary modelled is assumed to be *mobile* [28], which means they can dynamically and actively corrupt servers at any given time in a DMUE scheme, provided their corruption capabilities are bounded.

Contributions Our contributions are threefold: we formalise a dynamic multi-server updatable primitive called DMUE in Section 2, used to mitigate the problem of a single point of failure in standard PKUE schemes. In Section 3 we present a new notion of security against update unlinkable chosen ciphertext attacks (MUE-IND-CCA), which captures a mobile adversary attempting to corrupt a threshold or more of secret update tokens. It is crucial to maintain confidentiality through the ciphertext update unlinkability notion as it guarantees a ciphertext generated by the update algorithm is unlinkable from a ciphertext generated by fresh encryption, even when the adversary sees many updated ciphertexts of chosen messages. We highlight that the focus of our paper is to capture a notion of *confidentiality* in the threshold multi-server PKUE setting. However, we also note that extending the security framework to capture ciphertext integrity is possible, thus preventing adversarial ciphertext forgeries, and is included in a full version of this work for completeness. In Section 4 we present a generic construction of DMUE built from a single-server public-key UE primitive and a dynamic threshold secret sharing scheme. The crux of our generic construction is that the data owner acts as the dealer and distributes a vector of n update tokens shares per epoch to the corresponding servers. Then at least a threshold of t servers can reconstruct the complete (*master*) token and proceed to update the ciphertext to encryption in epoch $(e + 1)$. This is achieved using standard PKUE and secret-sharing techniques. We then consider the practicalities of applying DMUE by providing an overview of a concrete scheme built from dynamic proactive secret sharing, in which an old server committee participates in a redistribution process to refresh and securely distribute update tokens to the new epoch server committee. We conclude our work by proving that our generic DMUE scheme satisfies the ciphertext unlinkability security notion we propose.

Related Work To the best of our knowledge, there has been no discussion within the UE literature that considers the insecurity of a UE scheme following a single-point-of-failure (SPOF) with respect to the server performing ciphertext updates. We believe it is a natural step to explore the resilience of a UE scheme to further support the strong security guarantees desired in this area of research. Not only is our solution of a multi-server UE primitive a novel design, but it also enables us to consider dynamic changes in servers over time which is essential if a server becomes corrupt or can no longer provide a service. The most closely aligned primitive to DMUE is threshold proxy re-encryption (PRE) [7, 31]

whereby schemes distribute the process of ciphertext re-encryption and decryption delegation using secret sharing and standard PRE as building blocks. More recently, the authors of [27] proposed the first *proactive* threshold PRE primitive, labelled PB-TPRE, which extends the work of [7, 31] by addressing the issue of long-term secret shares as well a change in the proxies possessing shares. Consequently, the authors of [27] propose similar techniques to our generic DMUE construction. However, it is notable that the work on PB-TPRE demonstrates provable security of a concrete scheme that achieves the weaker confidentiality notion of chosen plaintext security as opposed to our work which is proven to satisfy security against chosen ciphertext attacks. Furthermore, we highlight that the fundamental differences between DMUE and PB-TPRE primitives stem from the distinctions between the standard PKUE and PRE primitives. In particular, proxy re-encryption (PRE) was first introduced by [5] as a primitive in which a proxy server re-encrypts a ciphertext under a sender's secret key and *delegates* decryption under a recipient's secret key. In contrast, UE uses the technique of key rotation for *time* updates from one epoch to the next. Further differences between the two primitives have been explored extensively in [10, 19, 22]. Besides the PRE primitive, the recent work of [15] uses a similar approach to our own ideas. The authors propose the first policy-based single-sign-on (SSO) system to prevent service providers from tracking users' behaviour. To achieve this, their primitive distributes tokens, conditioned on users' attributes, to multiple service providers in order to shield attributes and access patterns from individual entities. Whilst access control is not a focus of UE research, we observe that the methods used in [15] to mitigate SPOF are akin to our own core ideas.

2 Dynamic Multi-Server Updatable Encryption

In this Section, we introduce the notation used in this paper, followed by a formal definition of a DMUE scheme and the corresponding definition of correctness.

Notation A traditional updatable encryption scheme is defined by epochs of time e_i from the range of time $i = \{0, \ldots, \mathsf{max}\}$. We denote the current epoch e or use the subscript notation e_i for $i \in \mathbb{N}$ if we define multiple epochs at once. Further, (e_i, e_{i+1}) are two consecutive epochs for any $i \in \mathbb{N}$, the token is denoted $\Delta_{e_{i+1}}$ to update a ciphertext to epoch e_{i+1}, and $\tilde{e}$ represents the challenge epoch in security games. In the *dynamic multi-server* setting, we define for epoch e_i a set of servers $S_{e_i} = \{S^j\}_{j \in [n]}$ where $S_{e_{i+1}}$ may not be the same set, and update token $\Delta^j_{e_{i+1}}$ pertains to the token server S^j possesses. We use $(\mathcal{MSP}, \mathcal{CSP})$ to respectively denote the message space and ciphertext space of our scheme.

Extending the PKUE primitive to the dynamic multi-server setting (DMUE), a data owner must distribute tokens to every qualified server in the committee for that epoch, who respectively work together to update the ciphertext. The dynamic aspect of this primitive enables different sets of servers, chosen by the data owner at the time of token creation, to perform the ciphertext update in successive epochs. Formally, a DMUE primitive is defined as $\Pi_{\mathsf{DMUE}} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{TG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Upd})$ whereby algorithms $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ are

formalised as in standard PKUE [20] and the data owner runs all algorithms asides from Upd, the latter of which is run by the servers in a given epoch.[4]

Definition 1 (DMUE). *Given a set of servers S of size $n \in \mathbb{N}$ and a threshold $t \leq n$, a dynamic multi-server updatable encryption scheme is defined by a tuple of six PPT algorithms $\Pi_{\mathsf{DMUE}} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{TG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Upd})$ as follows.*

1. $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} pp$: the setup algorithm is run by the data owner, who uses security parameter 1^λ as input and randomly outputs the public parameters pp.
2. $\mathsf{KG}(pp, e_i) \xrightarrow{\$} k_{e_i} := (pk_{e_i}, sk_{e_i})$: given public parameters, the data owner runs the probabilistic key generation algorithm and outputs the public and private key pair (pk_{e_i}, sk_{e_i}) for epoch $\{e_i\}_{i \in [0, \mathsf{max}]}$.
3. $\mathsf{TG}(pp, sk_{e_i}, k_{e_{i+1}}, S_{e_{i+1}}) \rightarrow \{\Delta^j_{e_{i+1}}\}_{j \in [n]}$: the token generation algorithm is run by the data owner, who uses the following inputs: public parameters, the old epoch secret key sk_{e_i}, the new epoch public and private key-pair $k_{e_{i+1}} := (pk_{e_{i+1}}, sk_{e_{i+1}})$ generated by the key generation algorithm, and the new set of servers $S_{e_{i+1}} = \{S^j\}_{j \in [n]}$. The deterministically computed output is n update tokens $\{\Delta^j_{e_{i+1}}\}_{j \in [n]}$, which are securely sent to the *chosen* servers $S^j \in S_{e_{i+1}}$.[5]
4. $\mathsf{Enc}(pp, pk_{e_i}, m) \xrightarrow{\$} C_{e_i}$: given public parameters and the epoch public key pk_{e_i}, the data owner runs the probabilistic encryption algorithm on message $m \in \mathcal{MSP}$ and outputs the ciphertext C_{e_i}.
5. $\mathsf{Dec}(pp, sk_{e_i}, C_{e_i}) \rightarrow \{m, \perp\}$: given public parameters and the epoch secret key, the data owner is able to run the deterministic decryption algorithm in order to output message m or abort ($\perp$).
6. $\mathsf{Upd}(pp, \{\Delta^k_{e_{i+1}}\}_{k \in \mathbb{N}}, C_{e_i}) \rightarrow C_{e_{i+1}}$: for some $k \geq t$, the subset $S' \in S_{e_{i+1}}$ of servers, such that $|S'| = k$, can deterministically update ciphertext C_{e_i} using their tokens $\Delta^k_{e_{i+1}}$ to output an updated ciphertext $C_{e_{i+1}}$.

Correctness Intuitively, defining the correctness of the DMUE primitive follows from the definition of correctness for the single server PKUE primitive. Specifically, the correctness property ensures that fresh encryptions and updated ciphertexts should decrypt to the underlying plaintext, given the appropriate epoch key. However, the multi-server setting additionally has to encapsulate the concept of ciphertext updates from a *threshold* number of tokens. The formal definition of correctness follows.

Definition 2 (Correctness). *Given security parameter λ and threshold $t \leq k \leq n$, dynamic multi-server updatable encryption scheme (Π_{DMUE}) for n servers,*

[4] We note that in the multi-server setting, the update process is interactive and is therefore a protocol. However, we chose to use the term algorithm to stay in keeping with the single-server PKUE terminology as Π_{DMUE} can reduce to the single-server setting when $n = t = 1$.

[5] Note in the definition of DMUE that the data owner chooses the committee of servers $\{S_{e_{i+1}}\}_{\forall i \in \mathbb{N}}$.

as formalised in Definition 1, is correct if for any message $m \in \mathcal{MSP}$, for any $l \in \{0, \ldots, \mathsf{max}-1\}$ such that max denotes the final epoch of the scheme, and $i = (l+1)$, there exists a negligible function negl such that the following holds with overwhelming probability.

$$\Pr\left[\begin{array}{l} pp \xleftarrow{\$} \mathsf{Setup}(1^\lambda); \\ k_{e_i} = (pk_{e_i}, sk_{e_i}) \xleftarrow{\$} \mathsf{KG}(pp, e_i); \\ \{\Delta^j_{e_i}\}_{j\in[n]} \leftarrow \mathsf{TG}(pp, sk_{e_{i-1}}, k_{e_i}, S_{e_i}); \\ C_{e_l} \xleftarrow{\$} \mathsf{Enc}(pp, pk_{e_l}, m); \\ \{C_{e_i} \leftarrow \mathsf{Upd}(pp, \{\Delta^k_{e_i}\}_{k\in\mathbb{N}}, C_{e_{i-1}}) : \\ i \in \{l+1, \cdots, \mathsf{max}\} \wedge |k| \geq t\}; \\ \mathsf{Dec}(pp, sk_{e_{\mathsf{max}}}, C_{e_{\mathsf{max}}}) = m \end{array}\right] \geq 1 - \mathsf{negl}(1^\lambda).$$

Remark 1. The *multi-server* aspect of Definition 1 affects the TG and Upd algorithm definitions compared to standard PKUE algorithms. Note, we have omitted the formal definition of traditional PKUE [20] due to lack of space, however, we emphasise that Definition 1 satisfies the PKUE definition when $t = n = 1$.

3 Security Modelling

In this Section, we define a notion of security satisfying the highest level of *confidentiality* achievable in *deterministic* PKUE schemes. More specifically, the experiment models dynamic multi-server PKUE security against *update unlinkable chosen ciphertext* attacks (MUE-IND-CCA security). Crucially, unlinkability needs modelling to ensure that a ciphertext generated by the update algorithm is indistinguishable from a ciphertext generated by fresh encryption. Note, capturing the full capabilities of an adversary attacking a DMUE scheme is inherently challenging due to the *bi-directional* nature of ciphertext updates [18, 24, 30]. Consequently, it is necessary to record inferable information obtained from corrupted keys, tokens, and ciphertexts.

We start by detailing the lists recorded and oracles necessary to model the security of a DMUE scheme. For clarity, we will separate descriptions of oracles specific to the DMUE setting versus standard PKUE oracles, that is, the remaining oracles required in security modelling that are unchanged from the security framework of single-server PKUE [20]. The lists and oracles play a vital role in preventing trivial wins and guaranteeing security by capturing, in the lists a challenger maintains, the information an adversary can infer. Further, lists are checked after oracle queries as well and they're incorporated into the winning conditions of the security experiment.

We observe that our security model defines an adversary $\mathcal{A} := \{\mathcal{A}_I, \mathcal{A}_{II}\}$, representing a malicious outside adversary ($\mathcal{A}_I$) and dishonest server ($\mathcal{A}_{II}$). Typically,

only adversary $\mathcal{A}_I$ is considered in single server PKUE, as the literature assumes the lone server is honest. Conversely, the main motivation in our work is tackling the issue of a single point of failure regarding server updates. Thus, we have to consider adversary $\mathcal{A}_{II}$ to capture the threat of dishonesty or collusion of a *threshold* or more servers. To be succinct, our security experiment and definition uses the notation $\mathcal{A}$, however, we capture both types simultaneously. That is, an outside adversary is modelled in the usual manner of UE, through lists recording corrupted and inferable information. Specific to a corrupt server, the token corruption oracle is crucial in recording any epoch in which a threshold or more tokens have been corrupted.

- $\mathcal{L} = \{(e', C_{e'})_{e' \in [e]}\}$: the list containing the epoch and corresponding ciphertext in which the adversary learns (through queries to the update oracle $\mathcal{O}_{\mathsf{Upd}}$) an updated version of an *honestly generated* ciphertext.
- $\mathcal{K} = \{e' \in [e]\}$: the list of epoch(s) in which the adversary has obtained an epoch secret key through calls to $\mathcal{O}_{\mathsf{Corrupt\text{-}Key}}(e')$.
- $\mathcal{T} = \{e' \in [e]\}$: the list of epoch(s) in which the adversary has obtained at least a *threshold* number of update tokens through calls to $\mathcal{O}_{\mathsf{Corrupt\text{-}Token}}(e')$.
- $\mathcal{C} = \{(e', C_{e'})_{e' \in [e]}\}$: the list containing the epoch and corresponding ciphertext in which the adversary learns (through queries to the update oracle $\mathcal{O}_{\mathsf{Upd}}$) an *updated version* of the challenge ciphertext.
- $\mathcal{C}^* \leftarrow \{e' \in \{0, \ldots, e_{\max}\} | \mathsf{challenge\text{-}equal}(e') = \mathsf{true}\}$: the list of challenge-equal ciphertexts, defined by a recursive predicate challenge-equal, such that $\mathsf{true} \leftarrow \mathsf{challenge\text{-}equal}(e')$ iff : $(e' \in \mathcal{C}) \vee (\mathsf{challenge\text{-}equal}(e'-1) \wedge e' \in \mathcal{T}) \vee (\mathsf{challenge\text{-}equal}(e'+1) \wedge (e'+1) \in \mathcal{T})$.

Fig. 1. The set of lists $\mathbf{L} := \{\mathcal{L}, \mathcal{K}, \mathcal{T}, \mathcal{C}^*\}$ the challenger maintains in the global state (GS) as a record of during security games.

Lists We provide Figure 1 as a descriptive summary of the lists maintained by the challenger (as part of the global state GS) in the ensuing security experiment. We note that the main deviation in the list description compared to the single server PKUE setting [20] is found in list $\mathcal{T}$. Here, the challenger maintains a count of how many server tokens have been corrupted per epoch (see Figure 2), and only records the epochs in which a *threshold* number token have been corrupted. List $\mathcal{C}^*$, which is an extension of list $\mathcal{C}$, is also modified to the multi-server setting. This list contains the epoch and corresponding ciphertext in which the adversary learns (through queries to the update oracle $\mathcal{O}_{\mathsf{Upd}}$) an *updated version* of the challenge ciphertext. In greater detail, a CCA-secure DMUE scheme with deterministic re-encryption requires a challenger to record all updates of honestly generated ciphertexts and maintain a list of challenge-equal epochs ($\mathcal{C}^*$) in which a version of the challenge ciphertext can be inferred. That is, list $\mathcal{C}^*$ incorporates

a *challenge-equal predicate* presented in Figure 1 which encapsulate all of the *challenge-equal epochs* in which the adversary knows a *version* of the challenge ciphertext, either from calls to the update oracle or through computation. To illustrate, if $\{e, e+1\} \in \mathcal{C}^*$, it is possible for an adversary to perform the update computation of the ciphertext since the adversary can infer information using corrupted ciphertexts and tokens from epochs $\{e, e+1\} \in (\mathcal{C}, \mathcal{T})$. Thus, if an adversary knows a ciphertext $\tilde{C}_e$ from challenge epoch e and the update token Δ_{e+1}, then the adversary can compute the updated ciphertext to the epoch $(e+1)$ and realise challenge ciphertext $\tilde{C}_{e+1}$.

3.1 Oracles

First, we present an important predicate in updatable encryption security modelling as it will be utilised in the running of decryption and update oracles in our security experiment.[6] Crucially, the $\mathsf{isChallenge}$ predicate defined by [19] is used to prevent the decryption of an updated challenge ciphertext, irrespective of whether the updatable encryption scheme is designed for probabilistic or deterministic ciphertext updates. Informally, the $\mathsf{isChallenge}(k_{e_i}, C)$ predicate detects any queries to the decryption and update oracles on challenge ciphertexts ($\tilde{C}$), or versions (i.e., updates) of the challenge ciphertext.

Definition 3 (isChallenge Predicate). *Given challenge epoch $\tilde{e}$ and challenge ciphertext $\tilde{C}$, the isChallenge predicate, on inputs of the current epoch key k_{e_i} and queried ciphertext C_{e_i}, responds in one of three ways:*

1. If $(e_i = \tilde{e}) \wedge (C_{e_i} = \tilde{C})$, return true;
2. If $(e_i > \tilde{e}) \wedge (\tilde{C} \neq \perp)$, return true if $\tilde{C_{e_i}} = C_{e_i}$ in which $\tilde{C_{e_i}}$ is computed iteratively by running $\mathsf{Upd}(pp, \Delta_{e_{l+1}}, \tilde{C_{e_l}})$ for $e_l = \{\tilde{e}, \ldots, e_i\}$;
3. Otherwise, return false.

Now we describe the five oracles $\mathcal{O} = \{\mathcal{O}_{\mathsf{Dec}}, \mathcal{O}_{\mathsf{Next}}, \mathcal{O}_{\mathsf{Upd}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Token}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Key}}\}$ at a high level before providing detail of how they run.

- $\mathcal{O}_{\mathsf{Dec}}$: to prevent an adversary from trivially winning by querying the decryption of a queried challenge ciphertext, the following condition must be satisfied. The predicate $\mathsf{isChallenge}$ (Definition 3) must return false. In this case, the decryption of a valid ciphertext under the current epoch secret key is returned. Else, the failure symbol $\perp$ is returned.
- $\mathcal{O}_{\mathsf{Upd}}$: the update oracle only accepts and responds to calls regarding *honestly generated* ciphertexts or derivations of the challenge ciphertext, by checking lists $\{\mathcal{L}, \mathcal{C}^*\}$ respectively. If this is the case, the output is an update of the queried ciphertext to the current epoch. Next, the updated ciphertext and current epoch are added to the list $\mathcal{L}$. Moreover, if the $\mathsf{isChallenge}$ predicate returns *true* on the input of the *queried* key and ciphertext, then the current epoch is added to the *challenge-equal* epoch list $\mathcal{C}^*$.

[6] A predicate is a statement or mathematical assertion that contains variables. The outcome of the predicate may be true or false depending on the input values.

– $\mathcal{O}_{\mathsf{Next}}$: queries to the next oracle in challenge epoch e result in an update of the global state to the epoch $(e+1)$. This is achieved by running key and token generation algorithms to output the epoch key pair $k_{e+1} = (pk_{e+1}, sk_{e+1})$ and tokens $\Delta^j_{e+1}, \forall j \in [n]$, respectively. If the query is in an epoch such that the adversary has corrupted the epoch key *or* the epoch belongs to list $\mathcal{L}$, then the current challenge ciphertext must be updated to the next epoch using a threshold or more of the generated update tokens and the new ciphertext is added to the list of honestly updated ciphertexts $(\mathcal{L})$.
– $\mathcal{O}_{\mathsf{Corrupt\text{-}Token}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Key}}$: queries to these oracles allow the corruption of a threshold number of tokens and epoch secret key respectively. The restriction for both oracles is that the adversary's query must be from an epoch preceding the challenge-epoch e. Additionally, if an adversary queries the corrupt-token oracle for server S^j, not in the queried epoch server committee $S_{e'}$ then the corrupt-token oracle returns a failure symbol $\perp$.

Security Experiment After the initialisation which outputs a global state (Figure 3) and with a challenge public key pk_e, the adversary proceeds to query the oracles in Figures 2 and 3. They output a challenge message m' and ciphertext C' in the queried epoch e. Before proceeding, the challenger must check that the given message and ciphertext are valid (belongs to $\mathcal{MSP}, \mathcal{CSP}$ respectively). Otherwise, the challenger aborts the game and returns $\perp$. Moving forward, the challenger randomly chooses bit $b \in \{0,1\}$ which dictates whether the DMUE encryption algorithm or a version of the update algorithm is run on the respective challenge inputs $\{m', C'\}$. The resulting output is a challenge ciphertext $C^{(b)}$ such that for $b = 0$ the ciphertext is from fresh encryption and for $b = 1$ the ciphertext is generated by the update algorithm $\mathsf{UpdateCh}$.[7] The global state must be updated by the challenger, especially the set of lists $\mathbf{L}$. Equipped with a challenge output $C^{(b)}$ and public parameters, the adversary can query the oracles again before outputting a guess bit $b' \in \{0,1\}$. The adversary succeeds in the security experiment if they satisfy certain winning conditions and successfully guess the correct bit $(b' = b)$.

Definition 4 (MUE-IND-CCA-Security). *Definition 1 of a dynamic multi-server updatable encryption scheme (Π_{DMUE}) is MUE-IND-CCA secure against update unlinkable chosen ciphertext attacks following Figure 4 if for any PPT adversary $\mathcal{A}$ the following advantage is negligible over security parameter λ:*

$$\mathrm{Adv}^{\mathsf{MUE\text{-}IND\text{-}CCA},b}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{MUE\text{-}IND\text{-}CCA},0}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{MUE\text{-}IND\text{-}CCA},1}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = 1]| \leq \mathsf{negl}(1^\lambda),$$

for some polynomial time function $\mathsf{negl}(\cdot)$.

Preventing Trivial Wins and Ciphertext Updates We demonstrate the importance of the challenger recording lists $\mathcal{T}$ in the corrupt-token oracle, and

[7] Algorithm $\mathsf{UpdateCh}$ is used as compact notation, following the notation of [8], to denote the process of repeated application of the update algorithm from epoch $\{e+1, \ldots, \tilde{e}\}$.

$\underline{\mathcal{O}_{\mathsf{Upd}}(C_{e_i})}$
if $((e_i, C_{e_i}) \notin \mathcal{L}) \vee (e_i \notin \mathcal{C}^*)$ **then**
  return $\perp$
else
  for $e_l = \{e_{i+1}, \ldots, e\}$ **do**
    $C_{e_l} \leftarrow \mathsf{Upd}(pp, \{\Delta^k_{e_l}\}_{t \le k \le n}, C_{e_i})$
$C_e \leftarrow C_{e_l}$
return C_e
$\mathcal{L} \leftarrow \mathcal{L} \cup \{(e, C_e)\}$
if $\mathsf{isChallenge}(k_{e_i}, C_{e_i}) = \mathsf{true}$ **then**
  $\mathcal{C}^* \leftarrow \mathcal{C}^* \cup \{e\}$

$\underline{\mathcal{O}_{\mathsf{Next}}(e)}$
$k_{e+1} := (pk_{e+1}, sk_{e+1}) \xleftarrow{\$} \mathsf{KG}(pp, e+1)$
$\{\Delta^j_{e+1}\}_{j \in [n]} \leftarrow \mathsf{TG}(pp, sk_e, k_{e+1}, S_{e+1})$
`Update GS`
$(pp, k_{e+1}, T_{e+1}, \mathbf{L}, e+1)$
if $(e \in \mathcal{K}) \vee (C, e) \in \mathcal{L}$ **then**
  $(C', e+1) \leftarrow \mathsf{Upd}(pp, \{\Delta^k_{e+1}\}_{|k| \ge t}, C)$
$\mathcal{L} \leftarrow \mathcal{L} \cup \{(C', e+1)\}$

$\underline{\mathcal{O}_{\mathsf{Corrupt\text{-}Token}}(e', j)}$
if $(e' \ge e) \vee (S^j \notin S_{e'})$ **then**
  return $\perp$
else
  return $\Delta^j_{e'}$ `some` $j \in [n]$
  `Store tokens in a list` $T_{e'}$
if $|T_{e'}| \ge t$ `tokens have been corrupted in epoch` e' **then**
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{e'\}$

Fig. 2. Details of oracles an adversary $\mathcal{A}$ has access to during the security experiment of Definition 4 that is specific to the multi-server setting.

list $\mathcal{C}^*$ in the update oracle. Without the restrictions imposed on the corrupt-token oracle, the following can occur. If an adversary $\mathcal{A}$ corrupts t or more tokens $\{\Delta^k_{e+1}\}_{k \ge t}$ from the corresponding server committee S_{e+1}, in an epoch proceeding the challenge epoch $\tilde{e}$, then $\mathcal{A}$ is capable of trivially updating the ciphertext into the next epoch $(e+1)$, using a computed token, following Definition 1. Consequently, we place restrictions on calls to $\mathcal{O}_{\mathsf{Corrupt\text{-}Token}}$ and impose the winning condition in Figure 4. This condition states that the intersection of lists $\mathcal{K}$ and $\mathcal{C}^*$ must be empty. Thus, the challenge epoch cannot belong to the set of epochs in which a threshold of update tokens have been obtained/inferred, and there doesn't exist a single epoch where the adversary knows both the epoch key (public and secret key components) and the (updated) challenge-ciphertext [22]. The distinction of DMUE security modelling from single-server PKUE is that

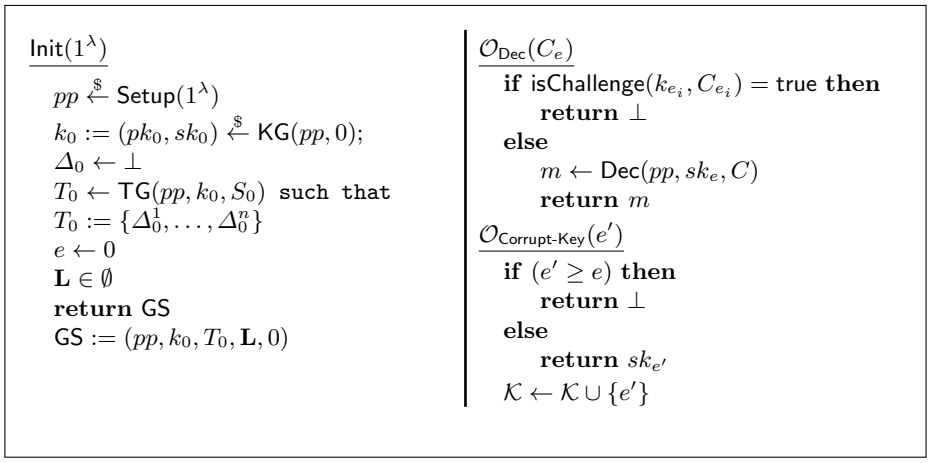

Fig. 3. The oracles an adversary has access to for the experiment capturing Definition 4 that remain unchanged from the single-server setting of a PKUE scheme.

list $\mathcal{T} \in \mathcal{C}^*$ does not record epochs in which token corruption occurred when the number of tokens corrupted is less than some threshold. That is, DMUE security modelling tolerates a certain level (below the threshold) of token corruption in any given epoch as less than the threshold of corrupted tokens does not provide the adversary with meaningful information.

4 Our Construction

In this Section, we use Definition 1 as a basis for formalising a *generic* DMUE construction. We achieve this using *dynamic proactive secret sharing* (DPSS) [2, 21, 23] and single server PKUE primitives as building blocks. Before going into detail about our construction, we present the formal definition of a DPSS protocol, as well as defining DPSS correctness and secrecy properties.

4.1 Construction Preliminaries

Dynamic proactive secret sharing (DPSS) [23] is an extension of traditional secret sharing [4, 29] such that shares belonging to a committee of parties are refreshed after some time has passed. A standard threshold secret sharing scheme (SS) [29, 4] has a dealer D distribute some secret s among a set of shareholders $\mathcal{P} = \{P^1, P^2, \ldots, P^n\}$ of n parties, according to an efficiently samplable distribution of the set of secrets labelled $\mathcal{S} = \{\mathcal{S}_\lambda\}_{\lambda \in \mathbb{N}}$, with security parameter λ. The aim of threshold SS is that no subset $t' < t$ of parties in P can learn the secret s, including an adversary controlling t' parties. Conversely, every subset $t' \geq t$ of parties in P is capable of reconstructing s. *Proactive* secret sharing schemes [17,

$\underline{\mathsf{Exp}^{\mathsf{MUE\text{-}IND\text{-}CCA,b}}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda)}$

$\mathsf{GS} \xleftarrow{\$} \mathsf{Init}(1^\lambda)$; $\mathsf{GS} := (pp, k_0, T_0, \mathbf{L}, 0)$ `such that` $\mathbf{L} := \{\mathcal{L}, \mathcal{K}, \mathcal{T}, \mathcal{C}^*\}$
$k_{e-1} \xleftarrow{\$} \mathsf{KG}(pp, e-1)$; $k_e \xleftarrow{\$} \mathsf{KG}(pp, e)$ `such that`
$k_{e-1} := (pk_{e-1}, sk_{e-1})$, $k_e := (pk_e, sk_e)$
$\{\Delta_e^j\}_{j\in[n]} \leftarrow \mathsf{TG}(pp, sk_{e-1}, k_e, S_e)$
$(m', C') \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(pp, pk_e)$
if $(m' \notin \mathcal{MSP}) \vee (C' \notin \mathcal{CSP})$ **then**
  return $\perp$
else
  $b \xleftarrow{\$} \{0,1\}$
  $C^{(0)} \xleftarrow{\$} \mathsf{Enc}(pp, pk_e, m')$ `and`
  $C^{(1)} \leftarrow \mathsf{UpdCh}(pp, \{\Delta_e^k\}_{|k|\geq t}, C')$
  $\mathcal{C}^* \leftarrow \mathcal{C}^* \cup \{e\}$; $\tilde{e} \leftarrow \{e\}$
$b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(pp, C^{(b)})$
if $(\mathcal{K} \cap \mathcal{C}^* = \emptyset)$ **then**
  return b'
`Else abort.`

Fig. 4. The security experiment for MUE-IND-CCA-security of a DMUE scheme. Let $\mathcal{O} = \{\mathcal{O}_{\mathsf{Dec}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Key}}, \mathcal{O}_{\mathsf{Next}}, \mathcal{O}_{\mathsf{Upd}}, \mathcal{O}_{\mathsf{Corrupt\text{-}Token}}\}$ denote the set of oracles that adversary $\mathcal{A}$ calls during the experiment, where the latter three oracles capture the multi-server aspect of a DMUE scheme.

25] (PSS) are designed for applications in which the long-term confidentiality of a secret matter, is achieved by a *refresh* of shares and consequently enable a reset of corrupted parties to uncorrupted. Observe that the secret itself remains constant, it is only the shares that are refreshed. *Dynamic* PSS (DPSS) [2, 21, 23] is a primitive with the same benefits as PSS plus an additional feature allowing the group of parties participating to change periodically. The following is a formal definition of DPSS protocol Π_{DPSS}=(Share, Redistribute, Recon) [2].

Definition 5 (DPSS Protocol). *Given a dealer $\mathcal{D}$, a secret $s \in \mathcal{S}_\lambda$ for security parameter λ, $L \in \mathbb{N}$ periods, and a set of $\{n^{(i)}\}_{i\in[L]}$ authorised parties $P^i = \{P_1^i, \ldots, P_n^i\}$, a (t, n) dynamic proactive secret sharing scheme is a tuple of four PPT algorithms $\Pi_{DPSS} =$ (Setup, Share, Redistribute, Recon) defined as follows:*

- **Share Phase**: $\mathcal{D}$ takes as input the secret s and performs the following steps non-interactively:
  1. $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} pp$: a probabilistic algorithm that takes as input security parameter 1^λ and outputs public parameters pp, which are broadcast to all parties in P.
  2. $\mathsf{Share}(pp, s, i) \xrightarrow{\$} \{s_1^i, \ldots, s_n^i\}$: a probabilistic algorithm that takes as input the secret $s \in \mathcal{S}_\lambda$ and period i, outputting n secret shares $\{s_j^i\}_{j\in[n]}$, one for each party in P.

3. Distribute s_j^i to party $P_j^i \in P^i$ for every $i \in [L]$ over a secret, authenticated channel.

- **Redistribution Phase**: the algorithm $\mathsf{Redistribute}$ takes as input consecutive periods $(i, i+1) \leq L$, the set of parties (P^i, P^{i+1}) and the vector of secrets $\{s_j^i\}_{j\in[n]}$ belonging to P^i, such that P^i need to refresh and communicate their vector of secret shares to the potentially different set of parties P^{i+1}. The output is a vector of secrets $\{s_{j'}^{i+1}\}_{j'\in[n]}$.
- **Reconstruction Phase**: In period i, any party in $P^i = \{P_1^i, \ldots, P_n^i\}_{i\in[L]}$ can participate in the following steps.
  1. Communication:
     (a) Each party $P_j^i, j \in [n]$ sends their share s_j^i over a secure broadcast channel to all other parties in P^i.
     (b) P^i parties independently check that they have received $(t-1)$ or more shares. If so, they proceed to the processing phase.
  2. Processing: Once P_j^i has a set of t' shares labelled S', they independently do the following:
     (a) $\mathsf{Recon}(pp, S', i) \rightarrow \{s, \perp\}$: a deterministic algorithm that takes as input the set S' of t' shares and outputs the secret s for period $i \in [L]$ if $t' \geq t$ or outputs abort $\perp$ otherwise.

The following two definitions are regarding the correctness and secrecy of a dynamic proactive secret sharing scheme Π_{DPSS}. We assume these properties hold when proving the correctness and security of a proposed construction presented in Chapter 5.

Definition 6 (DPSS Correctness). *Π_{DPSS} is* correct *if $\forall\lambda \in \mathbb{N}$ and for all possible sets of $\{n^{(i)}\}_{i\in[L]}$ authorised parties P^i, given $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} pp$; for all secrets $s \in \mathcal{S}_\lambda$ and any subset of $t' \geq t$ shares S' from $\mathsf{Share}(pp, s, i) \xrightarrow{\$} \{s_1^i, \ldots, s_n^i\}$ communicated by parties in P^i, there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\Pr[\mathsf{Recon}(pp, S', i) \neq s] \leq \mathsf{negl}(1^\lambda).$$

Definition 7 (DPSS Secrecy). *Π_{DPSS} is* secret *if $\forall\lambda \in \mathbb{N}$ and for all possible sets of $\{n^{(i)}\}_{i\in[L]}$ authorised parties P^i, given $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} pp$; for all secrets $s \in \mathcal{S}_\lambda$ and any subset of $t' < t$ shares S' from $\mathsf{Share}(pp, s, i) \xrightarrow{\$} \{s_1^i, \ldots, s_n^i\}$ communicated by parties in P, there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\Pr[\mathsf{Recon}(pp, S', i) \neq \perp] \leq \mathsf{negl}(1^\lambda).$$

Remark 2. In this work we focus on building our DMUE scheme from dynamic threshold secret sharing, however, observe that we can easily extend the construction of DMUE to be built from an alternative multi-party functionality, namely, a version of multi-party computation (MPC) [9, 13].

4.2 Building DMUE

Recall, a DMUE primitive is designed for the distribution of tokens, to multiple untrusted servers, which are used in the ciphertext update process. A threshold of servers can reconstruct the whole update token (Δ_e) for a given epoch (e), using the corresponding server tokens. By design, the threshold is necessary to correctly update the ciphertext into a new epoch. Moreover, the set of servers in any given epoch is fluid to allow for the removal of corrupted servers and support the realistic nature of long-term secret storage in which servers may need to change. Intuitively, DPSS is an ideal building block candidate since the techniques used cater to changes in the shareholders, achieved via a redistribution process from one epoch to the next. Additionally, it is required in a DPSS scheme (see Definition 5) that the secret is re-shared in every period in such a way that the shares from different windows of time cannot be combined to recover the secret. The only way to recover the secret is to obtain enough shares from the *same* period, a task which the literature [17] assumes is beyond the adversary's grasp and the redundancy of sharing allows robustness in the periods of the scheme. We incorporate the aforementioned techniques into the design of our DMUE construction.

High-Level Idea The key idea of our construction is that we leverage a single-server PKUE scheme and share the update token using a threshold secret sharing protocol. Intuitively, the update token in our construction will be formed from the current and preceding epoch keys, such that the data owner ($\mathcal{D}$), taking the position of the dealer in the DPSS scheme, distributes a vector of *token shares* $\{\Delta^j_{e_i}\}_{j\in[n]}$ to the set of n servers $S_{e_i} := \{S^1_{e_i}, \ldots, S^n_{e_i}\}$ for current epoch $e_i, \forall i \in \mathbb{N}$. Token share generation will take place after TG is run by $\mathcal{D}$ and this will occur for every epoch up to the final epoch (e_{max}). The algorithm Upd will also be adapted to the multi-server setting in line with Definition 1, such that a threshold of t or more servers in set $S_{e_{i+1}}$ are required to reconstruct the update token $\Delta_{e_{i+1}}$ and then independently perform the update process in the classical PKUE sense. Observe a key point clarified after the construction (Remark 3) is that the set of servers in consecutive epochs may overlap, and so they should not be able to learn the shares of the old or new epochs even though they participate in the redistribution process.

For ease of defining a generic construction, we design the scheme such that the *dynamic* feature is achieved in a trivial way, and does not use the DPSS techniques to evolve server committees. In other words, we do not trust the servers and assume the server committees for each epoch are selected by data owner $\mathcal{D}$ in some way. However, after presenting our construction in Definition 8 we will make practical considerations which allow for the servers in a given epoch to participate in the redistribution process of token shares in order to reduce the data owners' computational cost. More formally, we construct a DMUE scheme as follows.

Definition 8 (DMUE Generic Construction). *Given a (t,n) dynamic secret sharing scheme* $\Pi_{\mathsf{SS}} = (\mathsf{SS.Setup}, \mathsf{Share}, \mathsf{Redistribute}, \mathsf{Recon})$ *from Definition 5*

(Definition 5) and a standard public-key UE scheme $\Pi_{\mathsf{PKUE}} = (\mathsf{UE.Setup}, \mathsf{UE.KG}, \mathsf{UE.TG}, \mathsf{UE.Enc}, \mathsf{UE.Dec}, \mathsf{UE.Upd})$*, a DMUE scheme is defined by a tuple of six PPT algorithms* $\Pi_{\mathsf{DMUE}} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{TG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Upd})$ *as follows.*

1. $\mathsf{Setup}(1^\lambda) \xrightarrow{\$} pp$: run $\mathsf{SS.Setup}$ and $\mathsf{UE.Setup}$ on input security parameter 1^λ to randomly output the public parameters $pp := (pp_{\mathsf{SS}}, pp_{\mathsf{UE}})$ respectively.
2. $\mathsf{KG}(pp, e_i) \xrightarrow{\$} k_{e_i} := (pk_{e_i}, sk_{e_i})$: given public parameters pp, run the probabilistic key generation algorithm $\mathsf{UE.KG}$ to output the public and private key pair $k_{e_i} = (pk_{e_i}, sk_{e_i})$ for epoch $e_i, i \in \mathbb{N}, i \leq (\mathsf{max} - 1)$.
3. $\mathsf{TG}(pp, sk_{e_i}, k_{e_{i+1}}, S_{e_{i+1}}) \rightarrow \{\Delta^j_{e_{i+1}}\}_{j\in[n]}$: the data owner runs $\mathsf{UE.TG}$ to compute token $\Delta_{e_{i+1}}$, followed by $\mathsf{Share}(pp_{\mathsf{SS}}, S_{e_{i+1}}, \Delta_{e_{i+1}}) \rightarrow \{\Delta^j_{e_{i+1}}\}_{j\in[n]}$. Next, the data owner securely distributes $\Delta^j_{e_{i+1}}$ to server $S^j \in S_{e_{i+1}}$, where $S_{e_{i+1}}$ is the committee of servers for new epoch e_{i+1}.
4. $\mathsf{Enc}(pp, pk_{e_i}, m) \xrightarrow{\$} C_{e_i}$: given public parameters and the epoch public key pk_{e_i}, the data owner runs the probabilistic encryption algorithm $\mathsf{UE.Enc}$ on message $m \in \mathcal{MSP}$ and outputs the ciphertext C_{e_i}.
5. $\mathsf{Dec}(pp, sk_{e_i}, C_{e_i}) \rightarrow \{m, \perp\}$: given public parameters and the epoch secret key, the owner is able to run the deterministic decryption algorithm $\mathsf{UE.Dec}$ in order to output message m or abort ($\perp$).
6. $\mathsf{Upd}(pp, \{\Delta^k_{e_{i+1}}\}_{k\in\mathbb{N},|k|\geq t}, C_{e_i}) \rightarrow C_{e_{i+1}}$: given any valid subset $S' \subseteq S_{e_{i+1}}$ of the epoch e_{i+1} committee of servers, such that $|S'| \geq t$, shareholders in S' can reconstruct the update token by running $\mathsf{Recon}(pp_{\mathsf{SS}}, \{\Delta^k_{e_{i+1}}\}_{k\geq t}) \rightarrow \Delta_{e_{i+1}}$: Individually they can then update the ciphertext using the update algorithm $\mathsf{UE.Upd}(pp_{\mathsf{UE}}, \Delta_{e_{i+1}}, C_{e_i}) \rightarrow C_{e_{i+1}}$.

Correctness We show below our construction Π_{DMUE}, presented in Definition 8, satisfies correctness (Definition 2). Observe that by definition, the secret reconstruction algorithm Recon from Π_{DPSS}, used in step 6 of the update process, satisfies correctness following Definition 6 formalised at the start of this Section.

Theorem 1 (Correctness of Construction). *Π_{DMUE} is correct assuming the underlying public-key UE scheme Π_{PKUE} and the underlying secret sharing scheme Π_{SS} satisfy their respective definitions of correctness.*

Proof. Following Definition 2, Π_{DMUE} is correct if $\mathsf{Dec}(pp, sk_{e_{\mathsf{max}}}, C_{e_{\mathsf{max}}})$ outputs m with overwhelming probability, whereby $C_{e_{\mathsf{max}}}$ has been generated iteratively by the update algorithm Upd. In fact, this means the decryption algorithm $\mathsf{UE.Dec}$ is run and outputs m on the same honestly generated inputs. Note, one of the inputs is an update of the ciphertext to the final epoch ($C_{e_{\mathsf{max}}}$). Therefore, we assume this ciphertext has been generated correctly by entering a reconstruction phase of the SS scheme, that is, $\mathsf{Recon}(pp_{\mathsf{SS}}, \{\Delta^k_{e_i+1}\}_{k\geq t})$ is run to output token Δ_{e_i+1}. In turn, the resulting token is input into $\mathsf{UE.Upd}(pp_{\mathsf{UE}}, \Delta_{e_i+1}, C_{e_i})$ such that $C_{e_{\mathsf{max}}}$ is output. Let us assume instead that Recon and/or $\mathsf{UE.Upd}$ output $\perp$, contradicting both correctness assumptions, resulting in $\mathsf{UE.Dec}$ outputting $\perp$. In turn, the DMUE decryption algorithm Dec will also output $\perp$ instead of

m, which violates correctness in Definition 2. However, the assumptions that the failure symbol $\perp$ is output by the reconstruction phase or PKUE update algorithm contradict our assumption that the SS and PKUE schemes satisfy correctness. Thus, using proof by contradiction we can conclude that the DMUE scheme Π_{DMUE} also satisfies correctness. □

Practical Considerations: In Definition 8 of a generic DMUE scheme, the selection of epoch committees and generation of their respective update tokens arise from the data owner ($\mathcal{D}$). The advent of every epoch calls for $\mathcal{D}$ to generate token shares for the newly selected server committee. However, in Definition 8 we can also consider the involvement of the epoch server committee as a more elegant and *practical* solution to sharing the computational cost of token generation. That is, we can introduce a *redistribution phase* (following Definition 5) amongst the server committee during the running of token generation (TG) to exchange secret shares from one server committee to the next. Importantly, the redistribution techniques from DPSS literature support the refresh of the shares as an additional layer of security, such that these new shares still reconstruct the same secret. In the following, we redefine the running of token generation (Step 3 of Definition 8) to support server committee involvement in the redistribution phase.

$\underline{\mathsf{TG}(pp, sk_{e_i}, k_{e_{i+1}}, S_{e_{i+1}}) \rightarrow \{\Delta^j_{e_{i+1}}\}_{j\in[n]} :}$

1. $\mathsf{TG}(pp, sk_{e_0}, k_{e_1}, S_{e_1}) \rightarrow \{\Delta^j_{e_1}\}_{j\in[n]}$: the DMUE token generation algorithm is run in epoch e_0, as detailed in step 3 of Definition 8.
2. *Redistribution Phase:* To proactively redistribute token shares to a new epoch, the redistribution phase is run by data owner $\mathcal{D}$ and the committee of servers (S_{e_i}) in epoch $e_i, \forall i \in [1, \mathsf{max}-1]$. Using information provided by the data owner (this could be, for instance, a masking polynomial if Shamir's secret sharing [29] is being used), the servers in S_{e_i} proceed to refresh their individual secret token shares $\{\Delta^j_{e_i}\}_{j\in[n]}$. The new vector of token shares is labelled $\{\Delta^{j'}_{e_{i+1}}\}_{j'\in[n]}$, and they are securely distributed to the corresponding server $S^{j'} \in S_{e_{i+1}}$.

To explain the second step in more detail, the *refresh* of token shares can be achieved during the running of token share generation described above, using the underlying redistribution phase of the chosen concrete DPSS scheme (Π_{DPSS}). For instance, secret shares are refreshed in the Shamir-based [29] DPSS scheme of [2] in such a way that shareholders from the current committee *mask* their polynomial P with some polynomial Q, such that no party in this committee learns shares for new polynomial $P' := P + Q$ given to the next shareholder committee, and vice versa. Thus, care needs to be taken in the choice of DPSS scheme so as to preserve security, especially if there is a crossover between the old and new server committees. In line with the proposed DPSS scheme from the authors of [2], an overlap of *one* server possessing the same share in both committees is not a security issue, since the threshold of the scheme is not

violated. However, we must make the following stipulation if the crossover of servers is above the threshold to ensure the security (Definition 4, Section 3) holds in Definition 8.

Remark 3. If a threshold t or more servers, $S^j = S^{j'}$ for $S^j \in S_{e_i}$ and $S^{j'} \in S_{e_{i+1}}$ respectively, overlap in two consecutive server committees then we necessitate distinct token shares ($\Delta^j_{e_i} \neq \Delta^{j'}_{e_{i+1}}$).

5 Security Analysis

In this Section, we present and prove the formal statements of security for our DMUE generic construction Π_{DMUE} from Definition 8. The following statement of security is for our ciphertext unlinkability notion defined in Section 3. At a high level, we will separate our proof into two cases: when an adversary corrupts less than the threshold number of token shares, versus an adversary that corrupts a threshold or more token shares. In each case, we can rely on the security of the underlying building blocks. Specifically, we assume the secrecy of the DPSS scheme and the satisfaction of the corresponding single-server PKUE security notion.

Theorem 2. *Assume that Π_{DPSS} satisfies secrecy and suppose that Π_{PKUE} is a public-key updatable encryption scheme satisfying MUE-IND-CCA security for $t = n = 1$. Then Π_{DMUE} is a MUE-IND-CCA secure scheme.*

Proof. Following Definition 4, we want to show that there exists some negligible function negl under security parameter λ such that

$$\mathrm{Adv}^{\mathsf{MUE\text{-}IND\text{-}CCA},b}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) \leq \mathsf{negl}(1^\lambda). \tag{1}$$

given the security experiment detailed in Section 3, Figure 4. To prove Equation 1, we must focus on two separate cases: first when an adversary $\mathcal{A}$ has corrupted $l < t$ token shares in the corresponding security game epoch $\tilde{e}$ and second when $\mathcal{A}$ has corrupted $l \geq t$ token shares. Following the assumptions in Theorem 2, we note that for either scenario we also assume the adversary's challenge message and ciphertext (m', C') were created in some epoch $e < \tilde{e}$ before the current epoch $\tilde{e}$, otherwise, the security experiment will output $\perp$.

Case $(\mathbf{l} < \mathbf{t})$: Recall that *secrecy* is satisfied in the DPSS scheme Π_{DPSS}, the formal definition of which is detailed in Section 4. Consequently, an adversary has too few token shares from epoch $\tilde{e}$ to reconstruct the secret update token $\Delta_{\tilde{e}}$. In the case that the challenger randomly chose bit $b = 1$ (for $b = \{0, 1\}$) $\mathcal{A}$ cannot manually update their challenge ciphertext C' to ciphertext $C'_{\tilde{e}} := C^{(1)}$ due to the secrecy property. Moreover, if $\mathcal{A}$ queries oracle $\mathcal{O}_{\mathsf{Upd}}(C')$ to update the challenge ciphertext iteratively via epochs $\{e+1, \ldots, \tilde{e}\}$, as detailed in Figure 2, $\mathcal{A}$ is still incapable of winning the experiment as the update oracle will add $\tilde{e}$ to the list of challenge-equal epochs $\mathcal{C}^*$ and winning conditions $(\mathcal{K} \cap \mathcal{C}^*) = \emptyset$ mean that $\perp$ is output. Therefore, $\mathcal{A}$ is reduced to guessing bit b (in this case $b = 1$) which results in the advantage

$$\mathrm{Adv}^{\mathsf{MUE\text{-}IND\text{-}CCA},1}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{MUE\text{-}IND\text{-}CCA},1}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = 1] - \tfrac{1}{2}| \leq \mathsf{negl}(1^\lambda).$$

If the challenger randomly chose bit $b = 0$, $\mathcal{A}$ would either have to query the epoch secret key corruption oracle to obtain $sk_{\tilde{e}}$ to manually decrypt the ciphertext $C^{(0)}$, or make calls to the decryption oracle. The assumed security of Π_{PKUE} is essential in this instance to prevent trivial wins. We note that both of the named oracles are detailed in Figure 3. The former scenario requires $\mathcal{A}$ query oracle $\mathcal{O}_{\mathsf{Corrupt\text{-}Key}}(\tilde{e})$ which results in output $\perp$ to prevent trivial wins. The latter scenario means $\mathcal{A}$ calls oracle $\mathcal{O}_{\mathsf{Dec}}(C^{(0)})$ which will result in output $\perp$ due to the decryption oracle conditions. Specifically, the first condition of the $\mathsf{isChallenge}$ predicate (Definition 3, Section 3.1) is satisfied since $C^{(0)}$ is a challenge ciphertext and $\perp$ is output. Note that the output ($\perp$) does not inform the adversary whether or not the ciphertext is derived from fresh encryption in epoch $\tilde{e}$ or an update from a prior epoch. Therefore, $\mathcal{A}$ is reduced to guessing bit b (in this case $b = 0$) which results in the advantage

$$\mathrm{Adv}^{\mathsf{MUE\text{-}IND\text{-}CCA},0}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{MUE\text{-}IND\text{-}CCA},0}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = 1] - \tfrac{1}{2}| \leq \mathsf{negl}(1^\lambda).$$

Consequently, Equation 1 holds when $l < r$.

Case ($\mathbf{l} \geq \mathbf{t}$) : oracle $\mathcal{O}_{\mathsf{Corrupt\text{-}Token}}$ stipulates that the challenger needs to add the challenge epoch $\tilde{e}$ to list $\mathcal{T}$ (Figure 1). Crucially, epochs in $\mathcal{T}$ are incorporated into list $\mathcal{C}^*$ which captures all *challenge-equal* epochs. Thus, epoch $\tilde{e}$ belongs to $\mathcal{C}^*$ and winning conditions in our security experiment prevent trivial wins. That is, the intersection of sets ($\mathcal{K} \cap \mathcal{C}^*$) must be empty to prevent a trivial win from occurring. See the end of Section 3 for more depth on trivial wins.

If $t = n = 1$ we can rely on the assumed security of Π_{PKUE}, namely, Definition 4 is satisfied. Therefore, in the case of ($l \geq r$) and for either choice of $b = \{0, 1\}$, $\mathcal{A}$ is reduced to guessing bit b which results in the advantage

$$\mathrm{Adv}^{\mathsf{MUE\text{-}IND\text{-}CCA},b}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{MUE\text{-}IND\text{-}CCA},b}_{\Pi_{\mathsf{DMUE}},\mathcal{A}}(1^\lambda) = 1] - \tfrac{1}{2}| \leq \mathsf{negl}(1^\lambda).$$

Given the above, we can conclude that the Equation 1 is satisfied for any number (l) of corrupted tokens. □

Closing Discussion In this paper, we formalised a DMUE primitive and defined a generic construction, the latter of which was built from single-server PKUE and dynamic threshold secret sharing (DPSS) primitives. As such, the performance of our proposed DMUE scheme (from Definition 8) is directly reflected by the cost of adding a DPSS scheme to PKUE. In the future, we believe it is of interest to develop concrete DMUE schemes to formally analyse the efficiency, costs and security levels attained in the multi-server versus single-server setting of PKUE.

References

1. N. Alamati, H. Montgomery, and S. Patranabis. Symmetric primitives with structured secrets. In *Lecture Notes in Computer Science*, volume 11692, pages 650–679. Advances in Cryptology - CRYPTO 2019, Springer, 2019.

2. J. Baron, K. El Defrawy, J. Lampkins, and R. Ostrovsky. Communication-optimal proactive secret sharing for dynamic groups. In T. Malkin, V. Kolesnikov, A. Lewko, and M. Polychronakis, editors, *Lecture Notes in Computer Science*, volume 9092, pages 23–41. International Conference on Applied Cryptography and Network Security, ACNS 2015, Springer, 2015.
3. F. Benhamouda, C. Gentry, S. Gorbunov, S. Halevi, H. Krawczyk, C. Lin, T. Rabin, and L. Reyzin. Can a blockchain keep a secret? *IACR Cryptology ePrint Archive*, 2020:464, 2020.
4. G. R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the AFIPS National Computer Conference, NCC 1979*, volume 48, pages 313–318. International Workshop on Managing Requirements Knowledge (MARK), IEEE, 1979.
5. M. Blaze and M. Bleumer, G.and Strauss. Divertible protocols and atomic proxy cryptography. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 127–144. Springer, 1998.
6. K. Boneh, D.and Lewi, H. Montgomery, and A. Raghunathan. Key homomorphic prfs and their applications. In R. Canetti and Garay J.A., editors, *Lecture Notes in Computer Science*, volume 8042, pages 410–428. Advances in Cryptology, CRYPTO 2013, Springer, 2013.
7. X. Chen, Y. Liu, Y. Li, and C. Lin. Threshold proxy re-encryption and its application in blockchain. In X. Sun, Z. Pan, and E. Bertino, editors, *Lecture Notes in Computer Science*, volume 11066, pages 16–25. Cloud Computing and Security - ICCCS 2018, Springer, 2018.
8. V. Cini, S. Ramacher, D. Slamanig, C. Striecks, and E. Tairi. Updatable signatures and message authentication codes. In J.A. Garay, editor, *Lecture Notes in Computer Science*, volume 12710, pages 691–723. Public Key Cryptography, PKC 2021, Springer, 2021.
9. R. Cramer, I. Damgård, and U. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In B. Preneel, editor, *Lecture Notes in Computer Science*, volume 1807, pages 316–334. Advances in Cryptology- EUROCRYPT 2000, Springer, 2000.
10. A. Davidson, A. Deo, E. Lee, and K. Martin. Strong post-compromise secure proxy re-encryption. In *Australasian Conference on Information Security and Privacy-ACISP 2019*, volume 11547, pages 58–77. Lecture Notes in Computer Science, Springer, 2019.
11. Y. Dodis, H. Karthikeyan, and D. Wichs. Updatable public key encryption in the standard model. In *Lecture Notes in Computer Science*, volume 13044, pages 254–285. Theory of Cryptography Conference - TCC 2021, Springer, 2021.
12. E. Eaton, D. Jao, C. Komlo, and Y. Mokrani. Towards post-quantum key-updatable public-key encryption via supersingular isogenies. In *Lecture Notes in Computer Science*, volume 13203, pages 461–482. Selected Areas in Cryptography: 28th International Conference - SAC 2022, Springer, 2022.
13. D. Evans, V. Kolesnikov, M. Rosulek, et al. A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security*, 2(2-3):70–246, 2018.
14. A. Everspaugh, K. Paterson, T. Ristenpart, and S. Scott. Key rotation for authenticated encryption. In J. Katz and H. Shacham, editors, *Lecture Note in Computer Science*, volume 10403, pages 98–129. Advances in Cryptology- CRYPTO 2017, 2017.
15. T.K. Frederiksen, J. Hesse, B. Poettering, and P. Towa. Attribute-based single sign-on: Secure, private, and efficient. Cryptology ePrint Archive, Paper 2023/915, 2023. https://eprint.iacr.org/2023/915.

16. Y. J. Galteland and J. Pan. Backward-leak uni-directional updatable encryption from (homomorphic) public key encryption. In *Lecture Notes in Computer Science*, volume 13941, pages 399–428. Public-Key Cryptography - PKC 2023, Springer, 2023.
17. A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *Lecture Notes in Computer Science*, volume 963, pages 339–352. Annual International Cryptology Conference, Springer, 1995.
18. Y Jiang. The direction of updatable encryption does not matter much. In *Lecture Notes in Computer Science*, volume 12493, pages 529–558. Advances in Cryptology - ASIACRYPT 2020, Springer, 2020.
19. M. Klooß, A. Lehmann, and A. Rupp. (r) cca secure updatable encryption with integrity protection. In Y. Ishai and V. Rijmen, editors, *Lecture Notes in Computer Science*, volume 11476, pages 68–99. Advances in Cryptology- EUROCRYPT 2019, Springer, 2019.
20. J. Knapp and E. A. Quaglia. Epoch confidentiality in updatable encryption. In *Lecture Notes in Computer Science*, volume 13600, pages 60–67. International Conference on Provable Security - ProvSec 2022, Springer, 2022.
21. I. Komargodski and A. Paskin-Cherniavsky. Evolving secret sharing: dynamic thresholds and robustness. In Y. Kalai and L. Reyzin, editors, *Lecture Notes in Computer Science*, volume 10678, pages 379–393. Theory of Cryptography Conference- TCC 2017, Springer, 2017.
22. A. Lehmann and B. Tackmann. Updatable encryption with post-compromise security. In J. Nielsen and V. Rijmen, editors, *Lecture Notes in Computer Science*, volume 10822, pages 685–716. Advances in Cryptology, EUROCRYPT 2018, Springer, 2018.
23. S. K. D. Maram, F. Zhang, L. Wang, A. Low, Y. Zhang, A. Juels, and D. Song. Churp: dynamic-committee proactive secret sharing. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2369–2386. CCS 2019, Association for Computing Machinery, 2019.
24. R. Nishimaki. The direction of updatable encryption does matter. *Cryptology ePrint Archive*, 2021.
25. R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *Proceedings of the tenth annual ACM symposium on Principles of distributed computing*, pages 51–59. ACM, Association for Computing Machinery, 1991.
26. C. Qian, Y. J. Galteland, and G. T. Davies. Extending updatable encryption: Public key, tighter security and signed ciphertexts. *Cryptology ePrint Archive*, 2023.
27. Raghav, N. Andola, K. Verma, S. Venkatesan, and S. Verma. Proactive threshold-proxy re-encryption scheme for secure data sharing on cloud. *The Journal of Supercomputing*, pages 1–29, 2023.
28. D. Schultz, B. Liskov, and M. Liskov. Mpss: Mobile proactive secret sharing. *ACM Trans. Inf. Syst. Secur.*, 13, 2010.
29. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
30. D. Slamanig and C. Striecks. Puncture'em all: Updatable encryption with no-directional key updates and expiring ciphertexts. *Cryptology ePrint Archive*, 2021.
31. P. Yang, Z. Cao, and X. Dong. Threshold proxy re-signature. *Journal of Systems Science and Complexity*, 24(4):816–824, 2011.