Cross-User Leakage in Searchable Encryption: A Unified Approach to Symbols, Terminology, and Security Definitions

Victor Kampen

26th August 2024

Abstract—This thesis addresses the challenge of comparing security definitions for searchable symmetric encryption (SSE) which is when a party outsources the storage of his data to another party in an encrypted manner whilst still being able to search over it. It focuses on cross-user leakage, where the actions of one user leak information about the actions of or the data available to another user. We systematically review and clarify the implementations from four influential papers, introducing unified symbols and terminology to standardise protocols and security definitions. Key contributions include the development of a unified framework, the reformulation of security definitions, a new simulation-based security definition for the multi-key setting, and the proof that semantic security is equivalent to indistinguishability in the multi-key setting. The result of our work is a comprehensive unification of the SSE framework in the context of cross-user leakage, providing a clearer and more consistent foundation for future research.

Keywords: Cross-user leakage · Unified security framework

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Florian Hahn, for his unwavering support, insightful feedback, and approachable guidance throughout the course of this research. His assistance has elevated the quality of my work. Additionally, I am deeply thankful for all who prayed for me during these demanding times, providing me with unasked-for support and confidence. I also wish to thank those who candidly shared their own thesis-related challenges, as their honesty reassured me. Above all, I would like to thank God for bestowing upon me the talents I have yet did not deserve. HE has led me (through life) and without HIM I would be lost.

Contents

1 Introduction **5**
1.1 Searchable encryption and cross-user leakage . . . 5
1.2 Limitations to Curtmola et al. . . . 5
1.3 A difficult comparison . . . 7
1.3.1 Different security settings: MKSE and MUSE . . . 7
1.3.1.1 The setting: a different number of corpus owners . . . 7
1.3.1.2 Resulting complexity . . . 7
1.3.1.3 Challenges in comparing security in MKSE and MUSE . . . 9
1.3.2 Different language used . . . 10
1.4 Our contribution . . . 10
1.4.1 Covering contextual implementations . . . 11

2 Background information **11**
2.1 Index-based SSE . . . 11
2.2 SSE: Documents and local/auxiliary information . . . 12
2.2.1 Documents . . . 12
2.2.2 Local and auxiliary information . . . 12
2.3 Cryptography: PRF, DDH, oblivious maps . . . 13
2.4 Notation . . . 14

3 Security implementations across the papers **15**
3.1 Hamlin et al. (2018) . . . 15
3.1.1 Performance . . . 16
3.1.2 Final Notes . . . 16
3.2 Patel et al. (2018) . . . 17
3.2.1 Performance . . . 18
3.2.2 Final notes . . . 18
3.3 Wang et al. (2021) . . . 19
3.3.1 NFNU . . . 19
3.3.2 FU . . . 20
3.3.3 FNU . . . 20
3.3.4 Final notes . . . 21
3.4 Chamani et al. (2023) . . . 21
3.4.1 MITRA . . . 22
3.4.2 Adapting MITRA . . . 23
3.4.3 O-μSE . . . 23
3.4.4 Q-μSE . . . 24
3.4.5 Final notes . . . 24
3.5 Auxiliary information and the database . . . 25

4 Unification of notation and terminology **26**
4.1 Consistency across symbols . . . 26
4.2 Unification tables . . . 27
4.3 Synonyms . . . 30

5 Unification of protocols 32
5.1 Static SSE . . . 32
5.1.1 Algorithms and protocols . . . 32
5.1.2 Defining static SSE . . . 33
5.2 Dynamic SSE . . . 36
5.3 Fixed-size and resizable DSSE . . . 38
5.4 Application of our definition . . . 38

6 Current security definitions 43
6.1 Hamlin et al. . . . 44
6.2 Patel et al. . . . 46
6.3 Wang et al. . . . 48
6.3.1 Share forward-private . . . 48
6.3.2 Wang et al.'s definition for security . . . 49
6.4 Chamani et al. . . . 50

7 Defining security (MKSE) 52
7.1 Redefining multi-user auxiliary notions . . . 53
7.2 Indistinguishability and semantic security (MKSE) . . . 58
7.3 Proving equivalence . . . 60

8 Unified security definition 62
8.1 Unifying semantic security (MUSE) . . . 62

9 Literature overview 67

10 Discussion & conclusion 69

11 Appendix 74
11.1 Verifiability extension to μSE . . . 74
11.2 Forward and backward privacy . . . 74
11.3 Verifiability . . . 76

1 Introduction

1.1 Searchable encryption and cross-user leakage

Searchable encryption (SE) is a technique that allows a server to search over encrypted documents without decrypting them. It allows a corpus owner to upload a set of encrypted documents called a corpus to the server and to grant a set of authorised users access to it. Authorised users can then issue encrypted search queries over these documents by sending the search query to the server. The server can only view the encrypted documents but should still be able to retrieve only the documents that contain the searched keyword. The server should be able to do this searching and retrieving with minimum information leakage. Ideally, the server would learn nothing about either the user's queries or the documents. Although this can be achieved used homomorphic encryption and oblivious maps, in practice some leakage is allowed for better performance and the different implementations have varying security-efficiency trade-offs.

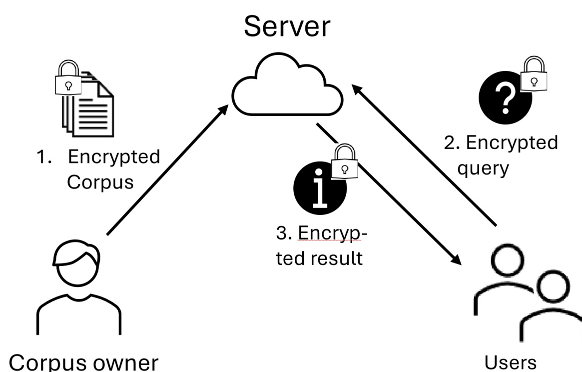

Figure 1: General idea of searchable encryption

There are multiple approaches to implementing SE but a popular one is searchable *symmetric* encryption (SSE), which uses symmetric key encryption. Symmetric key encryption has a significant performance advantage over asymmetric key encryption, and as such SSE is more popular than SE using asymmetric encryption. This work focuses on the security provided by recent works on SSE that focus on cross-user leakage. Cross-user leakage is when the actions of one user leak information about actions of and documents shared with a different user, and as such it only occurs in settings when there are multiple users to share documents with and who can issue search queries.

1.2 Limitations to Curtmola et al.

The de facto security standard for SE is given by Curtmola et al. [6]. Ideally, the server would learn nothing about either the user's queries or the documents and although this can be achieved using homomorphic encryption and oblivious maps, in practice some leakage is allowed for better performance. As such, Curtmola et al. come up with security definitions for two different adversarial models, defining the standard "minimal leakage" in the process. There are several limitations to their paper. Three significant limitations that are addressed in this work are:

1. The corpus is static as opposed to dynamic. This means that no documents can be added to or removed from the corpus after initial deployment. It also means that documents cannot be updated.
2. Users are granted access to the entire corpus; there is no sharing on a per-document basis.
3. Both adversarial models by Curtmola et al. are non-colluding adversarial models; there are no users that collude with the adversarial server. If the server tries to obtain as much information about the encrypted documents and queries as possible, then colluding users giving the adversarial server their decryption keys provides the server with a huge advantage. In particular, colluding users raise the new security concern of cross-user leakage: if one user colludes with or is corrupted by an adversarial server, what information is leaked about the queries of other users? Can this new information be used to learn more about the documents that the colluding user does not have (been granted) access to? It is crucial to examine this as such cross-user leakage could for example be detrimental in the context of patient data stored in the cloud.

It are these limitations that the works discussed in this paper address. The works by [13], [20], [33], and [4] all consider sharing on a per-document basis and all of them assume colluding adversarial models. The work by Chamani et al. also addresses the first limitation and considers a dynamic corpus. The primary concern of the papers however is to address the security concern caused by colluding users. The papers vary in the achieved efficiency, functionality and leakage, but comparison between the different works is difficult for multiple reasons.

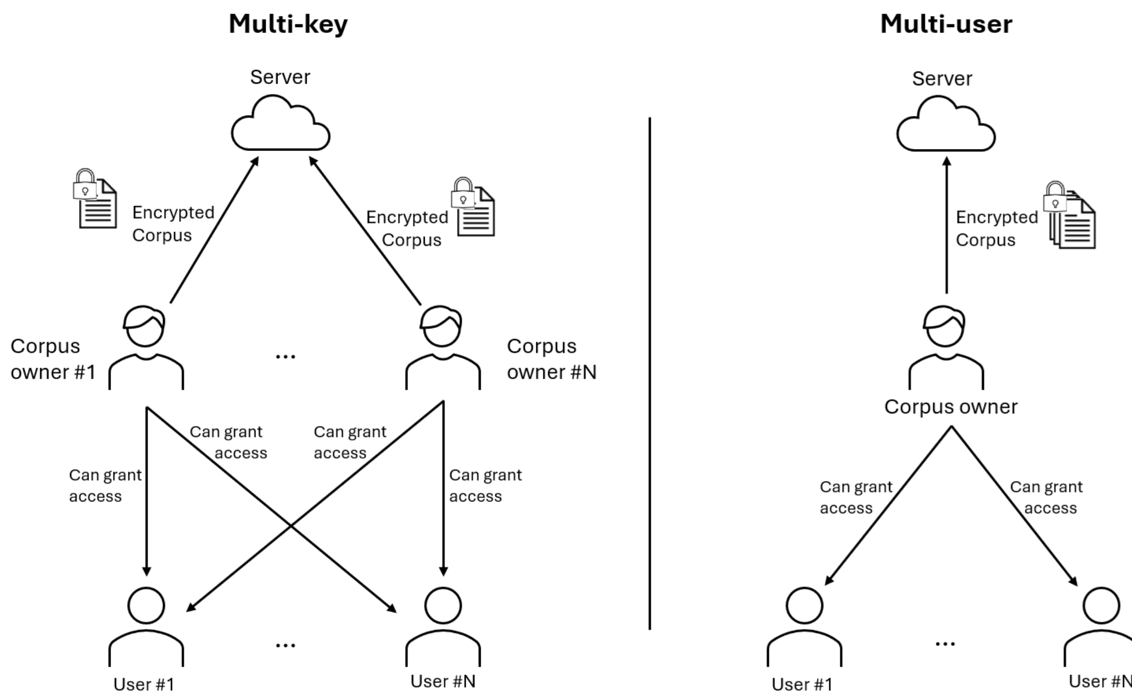

Figure 2: Difference between MKSE and MUSE

1.3 A difficult comparison

1.3.1 Different security settings: MKSE and MUSE

One factor complicating comparison between Hamlin et al. and the other works is that Hamlin et al. assume a multi-key setting instead of a multi-user setting like the other papers. The multi-key setting differs from the multi-user setting in only one way, namely the number of corpus owners. MKSE assumes a polynomial number of corpus owners and the multi-user setting assumes only one corpus owner. However, the multi-key setting assumed by Hamlin et al. also addresses the latter two limitations to Curtmola et al., namely the binary sharing of the corpus (by which we mean that it is shared completely or not at all) and the non-colluding adversarial model. It turns out that these two limitations give rise to a much more complex setting in the multi-key setting than in the multi-user setting. This is because colluding corpus owners are much stronger than colluding users and because different attacks and leakage become possible.

1.3.1.1 The setting: a different number of corpus owners

The work by Hamlin et al. concerns itself with multi-key searchable encryption (MKSE) whereas the other works concern themselves with multi-user searchable encryption (MUSE). The multi-key setting was first defined by [21] as a SE scheme where each user can provide a single (i.e. one) search token to the server to search over a set of documents encrypted with different keys. In simpler words, despite each document being encrypted with its own unique key, the search token is constructed in a way that is independent to this. I.e. the search token is independent of the number of shared documents/unique keys; the search token is of constant size/length.

Hamlin et al. assume in this multi-key setting that each document belongs to a different corpus owner. Note that this means that each of the corpus owners has a corpus of size '1 document'. Of course, corpus owners do not share keys with each other and thus the keys used are assumed to be unique. In this paper we will use Hamlin et al.'s assumption regarding MKSE, because if a real-life corpus owner owns multiple documents, he can be modeled by a set of MKSE-corpus owners. Thus, without loss of generality the only difference between the multi-key setting and the multi-user setting is the number of corpus owners. We have illustrated this in figure 2, where the adversary and colluding parties are in red. Note that any number of the respective parties can collude.

In MUSE there is one corpus owner that uploads his corpus to a server and grants users the ability to search over documents, and download and decrypt them. In MKSE there are multiple corpus owners that upload their corpus to a server and grant users the ability to search over documents, and download and decrypt them. As stated, initial works such as [6] were binary in the sharing of the corpus: either the entire corpus was shared with a user or none of it was shared. All the works we cover however allow documents to be shared on a per-document basis. Note that in MKSE each corpus consists of only one document and sharing is automatically on a per-document basis.

1.3.1.2 Resulting complexity

Colluding corpus owners: As stated: the primary concern of the different papers is to address the security concern caused by colluding users. It is clear that in MKSE a corpus owner has more information than a user: corpus owners have access to the plaintexts (i.e. the unencrypted keywords) and the encryption keys. Therefore, a server colluding with corpus owners is a more powerful adversary than a server colluding with users. In all implementations of MUSE covered however the corpus owner

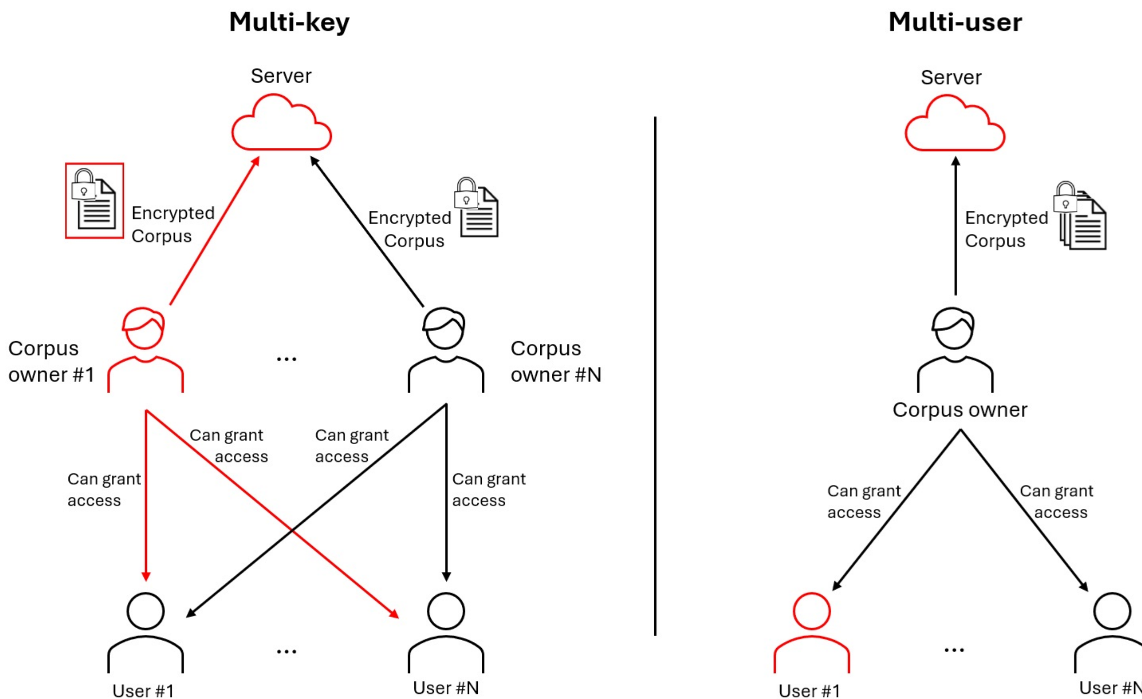

Figure 3: Adversary and colluding parties(red) in MKSE and MUSE

is assumed to be honest. This is simply because there is only one corpus owner and thus a malicious/colluding corpus owner would leak everything: there would be no privacy/security at all. Thus, the adversarial model used in MKSE where corpus owners collude with the server is a very strong one, and it is much stronger than the adversary in MUSE indeed.

Dictionary attacks and inherent leakage: It is thus that MKSE is the only setting that assumes colluding corpus owners. Significantly, assuming colluding corpus owners instead of colluding users allow for a new attack: a corpus owner can share a dictionary with a user. A dictionary is a document or a set of documents that allow the adversary to learn the queried keyword if the keyword is in the dictionary. For example, if a document contains only one keyword then upon a search hit it is certain that the query was for the only keyword. Note that the server must be able to determine search hits in SE. As stated, the corpus owner has access to the plaintexts and he is thus able to learn the queried keyword. As such, by sharing documents consisting of one keyword only and by covering all possible keywords with a document, the corpus owner can always learn the queried keyword. Therefore, corpus owners colluding with the adversary provide a big risk.

To remedy this risk "users can perform sanity checks on the documents shared with them to test how much leakage the server will get on their queries, and refuse to accept shared documents if they lead to too much leakage. Understanding when access pattern leakage is acceptable and when the leakage is not acceptable, is a fascinating and important direction for future study" [13]. The idea here is that queries always leak information. For example, one user has been granted access (by the same

corpus owner) to document 1 containing the keywords (apple, butter), to document 2 containing (butter, syrup), and document 3 (syrup, pear), then if document 1 and 2 are returned, the queried keyword was butter, if 2 and 3 were returned then it was syrup, and if 1 or 3 was returned then it was apple or pear respectively. This is an example of how queries inherently leak information even when the document contains multiple keywords. A way to think about this leakage is in terms of Venn diagrams. To illustrate our example we have made a Venn diagram illustration in figure 4. It should be noted though that the sanity check, when to reject or accept a document, was outside the scope of [13] and is thus also outside of the scope of this paper (cf. the related work section 9). Note that the entire issue of dictionary attacks via the sharing of documents does not exist in the MUSE where the corpus owner is honest. Our discussion of MKSE clearly points to MKSE being a more complex setting than MUSE.

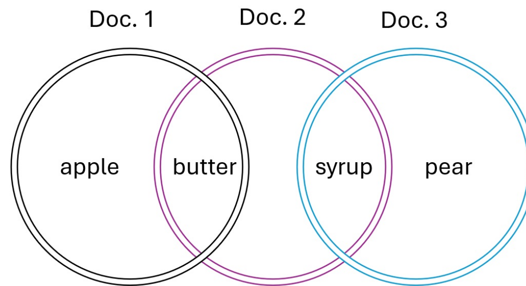

Figure 4: Queries inherently leak information

1.3.1.3 Challenges in comparing security in MKSE and MUSE

In essence, MKSE is the generalisation of MUSE and because of this it is more difficult to model and to prove theorems in as seen above. We stress that in addition to this generalisation described above, the multi-key setting as discussed by Hamlin et al. addresses the latter two of the limitations to Curtmola et al. [6] as described in section 1.2. The question then is: if the assumed setting by Hamlin et al. is much more complex, to what degree can it be compared to the security provided by Curtmola et al. [6]?

Specifically, Curtmola et al. have two different security definitions per adversarial model. The security definitions are a simulation-based definition called “semantic security” and a definition based on the comparing of encryptions called “indistinguishability”. The two adversarial models are a “non-adaptive” (NA) or a “selective” adversary who issues all queries to be executed (or simulated) at the same time, and an “adaptive” adversary who issues a set of queries to be executed (or simulated) with each query being determined after the execution of the previous one.[1] Curtmola et al. proved for MUSE that semantic security is equivalent to indistinguishability in the presence of a non-adaptive adversary. However, in the case of an adaptive adversary semantic security implies indistinguishability and is assumed to be a stronger requirement. Note that this proof is under the limitations as described in section 1.2.

Hamlin et al.’s assume a non-adaptive adversary. Their security requirement is one of indistinguishability and “a natural question that arises in this context is whether indistinguishability-based

[1] These concepts are expounded in section 6

security ... implies simulation-based security" [13]. It should be noted that there is no formal definition for simulation-based security in this context either and as such another natural question to ask is what simulation-based security is in MKSE. All in all the question can broadly be formulated as the following: What security is exactly provided by Hamlin et al. [13]? As described in section 1.3.1, the multi-key setting is more complex than the multi-user setting, legitimising the question.

1.3.2 Different language used

Another reason why comparing the different works is difficult is because the language used can differ significantly from work to work. The work by Curtmola et al. [6] was defined with respect to MUSE. Hamlin et al.'s MKSE brings its own security definitions and lexicon – i.e. terminology and symbols – with it. There is some overlap between the lexica but the two also share terms which differ in meaning. For example, a "data owner" in Hamlin et al. owns exactly 1 document which is called a set whereas a "data owner" in Wang et al. [33] owns exactly 1 dataset which is a set of documents. As such, using the term "encrypted set" can also have different meanings if we are not careful. An encrypted set refers in [13] to an encrypted set of keywords, but an encrypted set in [6] would or could logically speaking refer to an encrypted set of documents.

Despite the lexica of [20], [33], and [4] being more similar to each other, there is no consensus on certain aspects such as if a document contains metadata or even a list of authorisation tokens allowing a user to query on the document in question. Other examples of aspects without agreement are protocol names, names for security players (manager, data owner, corpus owner), and part of the symbols. Besides this, some terminology contributes to the confusion: We have previously mentioned the processed set, but as another example: what is the difference between metadata and auxiliary data?

1.4 Our contribution

Our aim is to facilitate comparison between the different papers by introducing a unified framework for SE with a focus on multiple users. It is necessary to have a consistent and coherent narrative and to get rid of all ambiguity by addressing all terms, symbols, algorithms and protocols that are not used consistently across the papers. The terms and symbols are an essential part of the protocols, and the protocols are an essential part of the security games. It is of absolute importance to remove any ambiguity from all of these components before we can unify the security definitions.

Thus, we will start the introduction of our unified framework by unifying the symbols and terminology. We do this in section 4, using tables to link our unified symbols back to the different papers and demonstrating our coverage of them. After this, we expand the framework by including the algorithms and protocols, and definitions for static and dynamic SSE in section 5. Similar to the previous section, we conclude section 5 by showing that our unified protocols and definitions can be applied to the different papers.

Having finished the basics of the framework we will then unify the security definitions. First we will examine the different security definitions as introduced by the papers in section 6. We will then unify these definitions in the sections that follow. In section 7 we will provide a simulation-based security definition for the multi-key setting that is equivalent to the indistinguishability based definition by Hamlin et al. [13]. Both definitions assume a non-adaptive adversary. This simulation-based security definition is novel and requires the reformulation of the security definition by [13]. We are going to do all this in section 7 in the following way:

1. We will redefine the auxiliary notions as introduced by Curtmola et al. [6], which were defined with respect to the multi-user setting whereas the security definition by Hamlin et al. [13] is with respect to the multi-key setting. The auxiliary notions that we define will address the following limitations to Curtmola et al.: documents are not shared on a per-document basis, and the adversary is assumed non-colluding.
2. We will reformulate the security game by Hamlin et al. into a indistinguishability-based definition similar to Curtmola et al. (but then for the multi-key setting as described by [13]).
3. We will introduce a novel simulation-based definition for the setting as described by Hamlin et al.
4. We will prove equivalence between our novel simulation-based definition and our indistinguishability-based definition.

Having done so we have addressed an open question in the field of MKSE: is semantic security in MKSE stronger than indistinguishability or are they equivalent assuming a non-adaptive adversary? Now that we have addressed part of the unclarity in the field, we will move on to address the remaining unclarity: we are going to finish the unification framework by unifying the remaining MUSE-security definitions in section 8.

At the end of section 7 our framework is complete and consists of the following aspects of SE with a focus on multiple users:

1. A unified lexicon covering the terms and symbols.
2. Unified algorithms/protocols.
3. A unified definition of SSE including a definition for static SSE and dynamic SSE (DSSE), and fixed-size and resizable DSSE.
4. A unified security definition.

The conclusion of our unified framework is followed by information on related work in section 9.

1.4.1 Covering contextual implementations

It is the aim of this paper to facilitate the security comparison of the 4 papers concerning cross-user leakage mentioned in the introduction, addressing each of the difficulties that obscure a comparison between them. The first step towards this facilitation is to introduce the implementations as put forth by the different papers using our unified framework. In this first step we explain the different design choices by the different authors and we address any unclarity concerning the implementations in the process. By providing the implementations we provide a contextual understanding of SE and the current state of research into SE with cross-user leakage. The contextual understanding will make it easier to follow our unification and as such its crucial to the introduction of our unified framework.

2 Background information

2.1 Index-based SSE

A common technique to boost performance is using a server-stored index to increase search performance. An index is a data structure that links keywords to document pointers whilst supporting efficient

keyword searching. Given a keyword, the index returns the pointers to the documents containing it. If the index does not contain the given keyword, no further information should be leaked about its contents. To achieve this, the search operation for a keyword can only be performed by users that possess a (valid) "trapdoor" for the given keyword. Valid trapdoors can only be created using a secret key. Without knowledge of the trapdoors, the index leaks no information about its contents.

Half of the recent works are indexless and half of them are (secure) index-based SSE. The indexless approaches are by Hamlin et al. [13] and Patel et al. [20] and do indeed suffer significantly in performance compared to the works with an index by Wang et al. [33] and Chamani et al. [4]. Despite their lower performance, both papers are included in our work for analysis and unification due to its influence on other papers – especially the paper by Hamlin et al. who is consistently mentioned for having an exceptionally strong security definition.

2.2 SSE: Documents and local/auxiliary information

2.2.1 Documents

General document definition: SE is all about the storage of documents on a server by a corpus owner, granting access/search rights to a set of users. The question "what is a document" turns out to be a nontrivial question and is thus a good starting point for our discussion. In the days of [6] SE-documents consisted of an identifier, a list of associated keywords that could be searched over, and some content. If a search query resulted in a document hit, then the document identifier would be returned to the user. The user could then request (a subset of) the contents corresponding to the document identifiers returned as the search result. Since this is shared across almost all implementations, modern approaches[2] assume that this is how it is done without making mention of this. In this paper we will use the same assumption. Thus, throughout our work a document generally consists of a document identifier and a set of associated keywords: $d = (id(d), W(d))$. We would like to point out that when modern approaches mention the "contents" of a document they subsequently mean the set of keywords $W(d)$ (and not the actual content of a document)! Now, we would like to point out that Hamlin et al. had a slightly different approach, not returning the document identifiers but returning the document content directly instead. Throughout this work we assume however that their implementation too returns document identifiers.

Patel et al., metadata: Now, a document *generally* consists of a document identifier and a set of associated keywords because a document by Patel et al. [20] contains some additional information. They assume that a document also contains some metadata which is returned along with the document identifier to help a user decide if he wants to retrieve the document's actual content (not referring to the keywords in this case). The metadata can for example be an upload date or title snippet. For [20] a document is thus of the form $d = (id(d), W(d), meta_d))$. We stress that the metadata does not change the document content retrieval mechanism; that this mechanism is still assumed; and that the content of a document in modern approaches and in particular by Patel et al. thus still refers to the set of associated keywords $W(d)$.

2.2.2 Local and auxiliary information

To store the documents on a server the papers often make use of local information ι and/or auxiliary information aux. The local information and auxiliary information are similar to each other but subtly

[2] I.e. the papers after Hamlin et al. [13] that we discuss: Patel et al. [20], Wang et al. [33], and Chamani et al. [4].

different. Local information is everything that is stored locally by the corpus owner (then it is the corpus owner's local information which we denote by ι) or user (then it is the user's local information which we denote by ι_u). Local information includes the auxiliary information. Auxiliary information is everything that is implementation specific. A problem with this terminology is that when a paper describes the auxiliary information, it is specific to that paper. When the PRF-based implementation by [13] is given, the auxiliary information is empty since it is the only scheme that is considered. Is the implementation however compared against those given by [33], none of which use a share key, then the share key in [13] is auxiliary information all of a sudden. As such, we are always careful to explain what the auxiliary information is, often referring back to section 3.5. Because of the uncertainty in auxiliary information, we will never use it in our definitions. It does however find use when we link our algorithm/protocol definitions back to the different implementations. There we have written out all commonalities and used aux to denote the differences.

2.3 Cryptography: PRF, DDH, oblivious maps

Several of the papers make use of a pseudo-random function (PRF). We will always denote PRFs by F. In cryptography, PRF are keyed function whose output is indistinguishable from that of a truly random function for any probabilistic polynomial time (PTT) adversary. This assumes that the adversary does not know the secret key. PRFs are deterministic and rely on the secret key to seem random. We will now formally define a pseudo-random function using a "negligible function" $negl : \mathbb{N} \to \mathbb{R}$. Formally, a negligible function $negl(\lambda)$ is such that for any positive polynomial $poly(.)$ and sufficiently large λ it holds that $|negl(\lambda)| < 1/poly(\lambda)$.

Definition 1 (Pseudo-random function)**.** *A function $F : \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^n$ is a pseudo-random if it is computable in polynomial time in k and if for all polynomial-size $\mathcal{A}$*

$$\Big| p\big(\mathcal{A}^{f_K(.)} = 1 : K \leftarrow \{0,1\}^k\big) - p\big(\mathcal{A}^{g(.)} = 1 : g \leftarrow \text{Func}[n,m]\big)\Big| \leq negl(k)$$

where the probabilities are taken over the choice of K and g. If f is bijective then it is a pseudo-random permutation.

When covering the implementation by [20] we note that the security of their PRF relies on the Decisional Diffie-Hellman (DDH) assumption. DDH is an assumption concerning computational hardness and the security provided by Patel et al. is dependent on this assumption.

Definition 2 (Decisional Diffie-Hellman assumption)**.** *Given a group generator g or order n then given $a, b, c \in \mathbb{Z}_n$ the tuples (g^a, g^b, g^{ab}) and (g^a, g^b, g^c) are computationally indistinguishable.*

In other words, if an adversary with limited resources is given g^a and g^b and g^c, he cannot tell if $g^c = g^{ab}$. Note that we can estimate what an adversary can reasonably have as "limited resources" by looking at the current technology, and based upon this we can scale n accordingly.

The implementations by [33] and [4] make use of oblivious maps (OMAPs). An OMAP is a data-structure of keys and values that completely hides the access pattern, meaning that data retrieval happens without revealing which data is being accessed. This ensures that the access pattern does not leak information about the nature of the data being retrieved. Oblivious maps are used to enhance privacy and security in various applications, specifically by [33] and [4] to store keyword counters in. We refer to [32] for a formal definition.

2.4 Notation

Throughout this work we will use the following notation conventions:

- If a subscript is shared across symbols then we often use a subscripted tuple instead. E.g., $(a, (b, c)_i) = (a, b_i, c_i)$.
- We can replace subscripted instance symbols with subscripted set symbol to denote the set. E.g., $K_U = \{K_u : u \in U\}$. The instance and set symbols are part of the unification of the symbolism section 4.2.
- We use $[n] = \{1, \ldots, n\}$.
- When an algorithm/protocol is keyed, we often subscript the key. E.g. $\mathsf{Setup}_K(.)$. Similarly, a keyed PRF F is often written as F_K. If F does not have a subscripted key, then the first input-parameter is the key for F. Note that throughout this document F's input parameters never contain more than one key and thus there is no confusion possible.
- When an algorithm/protocol is defined for an instance, we can replace the instance symbols in the in-/output to denote sequential invoking of the algorithm/protocol. E.g., $K_U \leftarrow \mathsf{UserKeyGen}(1^\lambda)$ is the same as $(K_{u_1} \leftarrow \mathsf{UserKeyGen}(1^\lambda)), (\ldots, K_{u_n} \leftarrow \mathsf{UserKeyGen}(1^\lambda))$ if $|U| = n$.[3]
- When defining protocols we can use $[.]$ to denote that the in-/output is optional, meaning it depends on the implementation.
- We denote a corpus by D per convention. In MKSE there are however multiple corpus owners – denoted by $\odot$ – and thus we might need to clarify whose corpus we mean: $D_\odot$. Note that $D_\odot$ can be interpreted to mean "all documents that $\odot$ has access to". In a similar fashion we can denote all documents that have been shared with user u by D_u. Strictly-speaking there is a subtle: the corpus owner owns the corpus whereas a user has been granted access to the document. We think however that these notations will not cause confusion due to their similarity.
- To restrict a set of documents D to the set of documents that contain keyword w we write $D(w)$.
- An adversary has to make a guess in security games indicated by a bit b. In the case of a real-ideal experiment we assume without loss of generality that 0 refers to the real world and 1 refers to the ideal world.
- We use 1^λ to denote a string of ones of length λ.

Most works explicitly require or assume that keywords in a document or keyword list are distinct. Throughout this work we will assume that all keywords in a set are distinct except for queries in a security game. If the queries in the security game are required to be distinct – such as for [13] – then we will mention this explicitly.

[3] We will later see that UserKeyGen is a probabilistic algorithm, meaning that $K_{u_i} \neq K_{u_j}$ for $i \neq j$.

3 Security implementations across the papers

In this section, we provide a high-level overview of the schemes proposed by various papers. Understanding these implementations is crucial not only for discussing the security guarantees but also for comprehending the chosen lexicon.[4] We review the papers [13], [20], [33], and [4] in order, which is in order of (first) publication date. Given that [13] is the only paper on MKSE while the subsequent papers address MUSE, this section is simultaneously organized by security setting. Throughout this section we will use $AccList(d)$ to denote the set of users that have access to document d, and $Access(u)$ to denote the set of documents that u has access to. $Access$ is the list of accesslists $Access = \{Access(u) : u \in U\}$.

All implementations have the following commonalities: there is at least one corpus owner. He encrypts his corpus and uploads it to the server. He shares documents with some users, granting them access and search rights. These users are able to send search queries to the sever, which retrieves the corresponding document (identifier) without decrypting either the queries or documents. We will now explain how the different papers implement this.

3.1 Hamlin et al. (2018)

As explained in section 1.3.1, Hamlin et al. [13] consider themselves with the multi-key setting. In this setting there are multiple corpus owners with corpora of size "1 document", each uploading his corpus to the server and sharing it with a set of users. The adversary by [13] is assumed to be the server colluding with corrupt corpus owners. The malicious corpus owners can explicitly share documents and in particular dictionaries with honest users, requiring users to perform sanity checks.

Corpus uploading: A corpus owners encrypts his document d using a document key K_d and uploads it to the server. This process is not restricted to the initial setup of the database but can be repeated for subsequent uploads.

Document sharing: To share a document d, the corpus owner sends the associated document key K_d to user u via a secure channel. The user downloads the document from the server and decrypts it using K_d. Hamlin et al. assume that a keyed pseudo-random function F is known to the server, corpus owner, and users. For each keyword w in d, the user uses his user key K_u to calculate the pseudo-random key $k^w = F_{K_u}(w)$. Another layer of encryption is used to prevent information leakage. Specifically, user u generates a random value r_d for the document, ensuring a unique identifier for each document. The user then computes the token for each keyword in d as $t^w = F_{k^w}(r_d)$. All tokens are inserted into a perfect hash table P_d to facilitate efficient searching [9]. The share key $\sigma_{u,d}$ consists of the random value and the hash table: $\sigma_{u,d} = (r, P)_d$, which is then sent to the server. This process effectively relinquishes document ownership from the sharer since the recipient constructed own copy. Please note that the perfect hash table is constructed over a document's keywords and not over the documents themselves. The set of documents accessible to a user is not indexed.

Searching: To search for a keyword w, user u calculates $k^w = F_{K_u}(w)$ and sends it to the server. The server performs the search on a per-document basis, doing the following for each share key: it

[4] Meaning both terminology and symbolism.

reads r_d, transforms the user's query k^w into $t^w = F_{k^w}(r_d)$, and checks the perfect hash table P_d for a hit on t^w. Finally, the server returns the identifiers of the documents where a hit occurred.

Since the user uses an unencrypted keyword w to generate k^w, t^w will also be based on an unencrypted keyword. Hamlin et al. state however that "the share key is (syntactically) 'tied' to the set for which it was generated (Share depends not only on the document and user keys, but also on the encrypted document". The described process of share key and keyword generation makes however clear that the document key is never used. To tie the share key to the document for which it was generated, Hamlin et al. use the random value r. The value r is tied to the document and in that sense also to the document key, if the document key is unique.

3.1.1 Performance

- Query size is constant $\mathcal{O}(1)$. Regardless of how many corpora a user has access to, the query remains the same. Constant query size is optimal.
- Storage size is linear per document in the number of share queries: $\mathcal{O}\left(\sum_{d \in D} |W(d)| \cdot |AccList(d)|\right)$, as each document is effectively replicated upon sharing.
- Search time is linear in the number of documents user u has access to: $\mathcal{O}(|D_u|)$, since no index exists.

Further details on the performance of Hamlin et al.'s construction can be found in Table 1 of [33], where it is compared against xu-m [20] and the constructions by Wang et al. [33].

3.1.2 Final Notes

Hamlin et al. assume that each corpus owner owns exactly one document. Note that this does not impose any restriction since a real-world adversary can effectively be modeled by a set of (theoretical) corpus owners.

Hamlin et al. have provided two schemes in their work. The second scheme uses public-coin differing-input obfuscation (cf. [14]) but is considered impractical. It has a search time polynomial in the number of documents. This scheme is however of theoretical interest since Hamlin et al. first theorised that the storage overhead (linear per document in the number of share queries) is inherent to achieving their stronger MKSE definition, but this second construction requires per-document server storage that is approximately the document size *plus* the number of share queries. They suggest this as evidence that a more efficient construction could potentially achieve their strong MKSE-security definition while maintaining optimal server storage overhead.

Hamlin et al. propose an indexless approach that uses a perfect hash table for searching. This hash table is associated with a shared key, which helps to speed up the search process within that shared key. However, this approach does not solve the problem of needing to search through each shared key individually as an index over the corpus does.

As explained in section 2, upon a search hit the PRF-implementation by Hamlin et al. returns the document d and not the document identifier id_d as is the current convention. This raises the question if Hamlin et al.'s use of the (document's) keyword set S could be considered a document in the current convention. The idea is this: if the modern convention is that a hit in the keywords $W(d)$ leads to the returning of id_d – which in turn allows the user to request the document content – and if Hamlin et al. only require S to return the document content, are S and $(id_d, W(d))$ equal to each other? If the

modern convention is to interpret $(id_d, W(d))$ as a document, is S too then a document? The answer to this is that S is equivalent to the list of keywords. A document would be of the form (id_d, S_d). We mention this because there is thus no dedicated symbol for an encrypted corpus/document in [13], and the symbol for an (document's) encrypted keyword set T_S does not perfectly suit any of the categories in the unification tables either (cf. section 4.2/tables 4-7).

It should be noted that Wang et al. consider Hamlin et al.'s PRF-implementation a trivial solution since each document is reuploaded under the user's own key. Hamlin et al. note however that the trivial solution is to have each user generate a new key for each set of documents that he wishes to share/are shared with him. They note that their solution does not require the user to maintain many keys. The share keys are uploaded to the server. Another advantage that their solution has over the trivial solution is that their user's queries are of constant size whereas the trivial solution has them grow linearly in the number of documents shared with the user.

3.2 Patel et al. (2018)

Patel et al. [20] provide three MUSE schemes: "x-uz, that has zero x-user leakage but is very inefficient", "xu-L, that is very efficient but highly insecure with very large x-user leakage", and "xu-m, that is as efficient as x-uL and more efficient than x-uz. At the same time, x-um is considerably more secure than x-uL" [20].[5] Therefore, our discussion will focus on the xu-m implementation. Patel et al. consider an honest-but-curious adversarial server that colludes with a coalition of corrupt users.

Corpus setup: Patel et al. achieve secure document encryption using the decisional Diffie-Hellman assumption with a cyclic group with generator g. The corpus owner stores a document d on the server by calculating $g^{w \cdot d}$ for every keyword $w \in d$ and pairs it with encrypted metadata $\text{Enc}_{K_d}(\text{meta}_d)$ to form the tuple $(g^{w \cdot d}, \text{Enc}_{K_d}(\text{meta}_d))$. He forms these tuples for every document in his corpus before sending these in random order to the server. The server places them in an array $xSet$. The document's metadata is extra information on the document such as it's creation time or it's title. It is only used by [20]. In Patel et al.'s scheme, uploading and encrypting documents is limited to an initial setup phase, after which no additional documents can be added to the database. This is because after the initial setup the output space of the PRF has been determined and adding additional documents will (likely) lead to conflicts.

Document sharing: To share d with user u, the corpus owner calculates a share key $\sigma_{u,d} = d \cdot u^{-1}$ and sends it to the server for storage in $uSet$. (Cf. section 4.2 on why $uSet$ is used instead of Σ).

Searching: To search for a keyword w, user u sends the search query $q = (g^{u \cdot w}, \text{pointer}(\sigma_{u,d}))$, consisting of a user-specific keyword query and for every shared document one pointer to the share key, to the server. Patel et al. do not elaborate upon why the user needs to store the pointers instead of the server storing user-pointer relations but this probably is since now the user-pointer relations leak gradually as the user queries instead of all of these relations being leaked from the start. The user now needs to store these pointers $pointer(\sigma)$ locally. Upon receiving such pointers, the server retrieves the share keys and applies them to $g^{u \cdot w}$ to produce $\{(g^{u \cdot w})^{\sigma_{u,d}}\} = \{g^{w \cdot d} | w \in d\}$. The server returns $\{(g^{w,d}, \text{Enc}_{K_d}(\text{meta}_d)) | w \in d\}$. Notice that $\sigma_{u,d}$ only exists if d has been shared with u.

[5] In older versions these were called zx-u, lx-u and mx-u respectively.

PRF-adaptation: The description above provides the general idea. However, since the dictionary that keywords are taken from is potentially small, the actual construction involves the use of PRF F to make the possible input space much larger. This way, an attacker cannot bruteforce the input space. For encryption two user keys are used per user $K_u = \{K_u^1, K_u^2\}$, and three document keys are used per document $K_d = \{K_d^1, K_d^2, K_d^3\}$.

- Document encryption $(g^{w,d}, \text{Enc}_{K_d}(meta_d))$ becomes $(g^{F(K_d^1,d)\cdot F(K_d^2,w)}, \text{Enc}_{K_d^3}(meta_d))$.
- Share key $d \cdot u^{-1}$ becomes $F(K_d^1, d) \cdot F^{-1}(K_u^2, d)$ with pointer $F(K_u^1, d)$.
- Search query $(g^{u\cdot w}, \{pointer(\sigma_{u,d})\})$ becomes $\{(g^{F(K_u^2,d)\cdot F(K_d^2,w)}, F(K_u^1, d))|d \text{ is shared with } u\}$ where $F(K_u^1, d)$ is a pointer to $\sigma_{u,d}$.

Note that this way the PRF prevents dictionary attacks.

3.2.1 Performance

- Query size is linear in the number of documents shared with the user $\mathcal{O}|D_u|$ due to the inclusion of a list of share key pointers in the query.
- Storage size is linear in the number of share queries $\mathcal{O}\Big(\sum_{d\in D} |AccList(d)|\Big)$ as document sharing requires a share key per user for each document.
- Search time is linear in the length of the array. The array contains all encrypted entries and its length is linear in total number of share queries $\mathcal{O}|\sum_{u\in U} D_u|$

More details about the performance of xu-m [20] can be found in Table 1, [33], where it is compared against the construction of Hamlin et al. [13] and the constructions by Wang et al. [33].

3.2.2 Final notes

Patel et al. make use of an accesslist $Access(u)$ that is stored on the server to indicate which documents a user has access to. The accesslist is updated as needed by the corpus owner. We do not discuss implementation however since there is no mechanism that enforces the server to adhere to this list. Consequently, the server may continue to search over and return documents to a user despite access revocation. Nonetheless, Patel et al. assume an honest-but-curious-server and in this setting this mechanism does provide some security. Even then, as observed by Wang et al., in a static corpus[6] a user is able to download a document as soon as it is shared with him. Now, Patel et al. do actually have an Unshare protocol, but it is not covered by any security guarantee (i.e. it is not part of their security definition). Because of these reasons we have decided to exclude the discussion of the authorisation tokens.

Lastly, it should be noted that in newer versions of their paper, Patel et al. include a section detailing possible modifications to their original implementation that allow for distinguishing between users with search rights and those with both search and edit rights. Additionally, they provide a section on further reducing the leakage of x-um.

[6] I.e. in a corpus in which documents are not added, removed or changed after the initial setup.

3.3 Wang et al. (2021)

Wang et al. [33] provide three MUSE schemes that "are the first ones in this setting that achieve search time linear in the number of documents that contain the searched keyword while eliminating cross-user leakage" [33]. Their schemes offer different security/performance trade-offs and all use a hash map as an index. Wang et al. introduce a new property called share forward privacy. The idea behind share forward privacy is that the server should not be able to tell whether the document being shared contains keywords that have been searched before. The security property share forward privacy is discussed in detail in the section on security definitions 6.3.1. The three schemes provided by Wang et al. are:

1. **NFNU (No Forward privacy and No User storage)**: The "basic" scheme that does not offer share forward privacy but also does not require additional storage.
2. **FU (Forward private and requires User storage)**: Achieves share forward-privacy at the cost of increased user storage (one counter per unique keyword in the dataset).
3. **FNU (Forward private and Not requiring User storage)**: Achieves share forward-private scheme at the cost of server-stored oblivious maps.

Wang et al. assume that PRF F is known to the corpus owner and the users. This PRF is used in all schemes to construct a hash map as an index, i.e. to pseudorandomly encrypt the keywords. None of the schemes provided by Wang et al. allow the corpus owner to upload documents after the initial setup for the same reason as Patel et al.: a fixed PRF output space. The adversarial model assumed by Wang et al. is an active adversarial server that corrupts a subset of users.

3.3.1 NFNU

Corpus setup: In NFNU the corpus owner sets up the online database by encrypting his corpus and uploading it to the server. For each user u, the corpus owner generates a user key K_u, sends it to u for storage in his auxiliary data aux_u, and stores it in his auxiliary data aux.[7] Furthermore, for each u the owner initialises a list of keyword counters Cnt_u of size $|W|$ and stores it in Cnt. An empty index I is sent to the server, whereas Cnt is stored locally by the corpus owner.

Document sharing: To share a document d, the corpus owner increases for each keyword $w \in d$ the user's keyword counter $Cnt_u[w]$ by 1 or initialises it to 1 if $Cnt_u[w]$ did not yet exist. The corpus owner generates the address of the index entry by using the user's key K_u to calculate the address $addr = F_{K_u}(w, Cnt_u[w]||0)$. The value val that is stored at the index address is $val = id(d) \oplus F_{K_u}(w, Cnt_u[w]||1)$ and is an encryption of the document identifier. This way, a list of tuples $(addr, val)$ is generated and they are sent to the server to update I.

Searching: To search for a keyword w, user u sends the address $F_{K_u}(w, i||0)\Big|_{i=1}$ to the server. The server returns the encrypted value retrieved from I and u sends $F_{K_u}(w, i||0)\Big|_{i=2}$ to the server. This repeats until the value found by the server is $null$, indicating that it has not been set before. This

[7] Throughout this document we assume that whenever something is stored in private memory, it is done in a way that the owner of the private memory can recover the information in a meaningful way. For example, the corpus owner can store (u, K_u) in private memory or make a list of user keys $UserKeys$ and insert K_u at $UserKeys[u]$.

prompts the server to send the “stop” message to the client. The client keeps generating addresses until it receives the stop message from the server.

Performance of NFNU

- Query size is linear in the number of keyword occurrences. Since the expected values of a dictionary are expressed in percentages, query size is linear in the number of documents shared with a user: $\mathcal{O}|D_u|$.
- Storage size (of the index) is linear in the size of the keywords $|W|$ and the number users $|U|$ since per user one address is needed per keyword in a document $\mathcal{O}\Big(\sum_{d \in D} |W(d)| \cdot |AccList(d)|\Big)$.
- Search time is linear in the number of documents shared with u that contain the searched keyword: $\mathcal{O}|D_u(w)|$. This is optimal.

3.3.2 FU

User-stored keyword counters: In FU the user’s keyword counters Cnt_u that were stored locally by only the corpus owner in NFNU (3.3.1), are now also sent to the respective user for local storage in his auxiliary data on the corpus aux_u. Therefore, to share a document in FU, the owner now also sends the updated counter to the user.

Searching: To search for a keyword w in FU, user u can now look up $Cnt_u[w]$ and generate the query by computing all the relevant addresses $\{F_{K_u}(w, i||0) \Big| 1 \leq i \leq Cnt_u[w]\}$. Since the keyword counter is known to the user, he does not need to send an address to the server and await if the server returns *null*, for all the computed addresses are initialised. When a document is shared with u the corpus owner sends the updated keyword counters directly to u. Note that this assumes that the user is constantly online.

Performance of FU

- Query size is linear in the number of documents a user has access to $\mathcal{O}|D_u|$.
- Storage size is linear in the number of keywords per document and in the number of share queries: $\mathcal{O}\Big(\sum_{d \in D} |W(d)| \cdot |AccList(d)|\Big)$. User storage size is linear in the number of keywords that require a counter $\mathcal{O}(|W|)$. User storage size is not linear in the size of the dictionary from which the keywords are drawn since the corpus is assumed to be static.
- Search time is linear in the number of documents that contain the searched keyword: $\mathcal{O}|D_u(w)|$. This is optimal.

3.3.3 FNU

Server-stored keyword counters: In FNU the user’s keyword counters are stored in an OMAP $OMAP_u$ on the server. When setting up the online database, the corpus owner now additionally initialises an $OMAP_u$ of size $|W|$ for each user u and sends the key of the OMAP to u for local storage in u’s auxiliary data aux_u. Throughout this work though we will assume that the user key is the key used for the OMAP, just like is done by [4], even though this assumption is not made by [33]. Since the keyword

counters are now stored server-side, to share a document in FNU the owner updates $OMAP_u$ with the updated keyword counters.

Searching: To search for a keyword w in FNU, user u retrieves $Cnt_u[w]$ from $OMAP_u[w]$ and generates the query by computing the addresses as in FU (3.3.2).

Performance of FNU

- Query size is linear in the number of documents a user has access to $\mathcal{O}|D_u|$.
- Storage size is linear in the number of keywords per document and in the number of share queries $\mathcal{O}\Big(\sum_{d \in D} |W(d)| \cdot |AccList(d)|\Big)$ since $|W| \leq |W(d)| \cdot |AccList(d)|$.
- Search time is linear in the number of documents that contain the searched keyword $\mathcal{O}|D_u(w)|$. This is optimal.

3.3.4 Final notes

More details about the performance of NFNU, FNU and FU can be found in Table 1 of [33], where they are compared against xu-m [20] and the construction[8] by Hamlin et al. The different schemes are designed for different use cases. NFNU is used in settings where users have limited permanent storage or use multiple machines for accessing the database, and where forward privacy is a good trade-off. FU is employed when forward privacy is important and users use only one device for accessing the database, as using multiple devices could compromise forward privacy [33]. FNU bridges the two schemes by providing forward security to multiple devices, but at the cost of performance [33]. Thus, the schemes are adapted to different use cases.

Wang et al. make mention of an accesslist. This list is only required during the execution of the Search protocol where it is part of the Server's input, but it is unused as is evidenced by their protocol description ("Algorithm 3 Search" [33]). The usage of $Access$ seems to be only of theoretical interest for the leakage analysis and as such we have omitted it in our description.

Finally, it should be noted that the the storage requirement for the keyword counters – regardless of storage location and method – is linear in the number of users $|U|$ and the number of keywords: $\mathcal{O}\Big(|W| \cdot |U|\Big)$. Either every time a new unique keyword is encountered the counter should be re-encrypted to prevent the adversary of learning the value of the newly added keyword counter (it is added since the keyword is encountered for the first time and thus it is initialised to 1) or the keyword counters should be initialised to the size of the dictionary, which can be large. If English is chosen as the dictionary of choice then more than $500,000$ entries can be required (Oxford online dictionary [24]) or even more than $1,500,000,000$ entries (Cambridge online dictionary [23]).

3.4 Chamani et al. (2023)

Chamani et al. [4] provide two MUSE schemes that are the first to achieve update handling in the presence of corrupted parties [4]. These schemes, O-μSE and Q-μSE, are similar and both eliminate cross-user leakage (even though the definitions of Chamani et al. allow for cross-user leakage [4]). Both schemes start with the replication of the single-user scheme MITRA [10], under the assumption

[8] I.e. the PRF-based construction that we discussed.

that direct communication between the corpus owner and users is not always possible and thus cannot be relied upon. Since both schemes use MITRA-replication and only differ in the way they handle updates, Chamani et al. consider them to be one scheme, μSE, with two implementations. Chamani et al. consider an active adversarial server that corrupts a subset of users.

Now, the schemes have different efficiency trade-offs but offer the same security [4]. The two schemes are:

1. **O-μSE**: Uses server-stored OMAP-keyword counters to address the communication assumption.
2. **Q-μSE**: Stores the counters user-side and uses server-stored update queues to address the communication assumption.

Both schemes can be extended to include verifiability of search results but this is not within the scope of our work. Confer appendix A, section 11.1) for a brief coverage.

As stated, both O-μSE and Q-μSE use the replication of an efficient and secure single-user scheme MITRA. This scheme is dynamic, meaning that documents can be uploaded and updated after the initial setup. We formally discuss dynamic SSE schemes in section 5.2. Since MITRA is a single-user scheme the documents cannot be unshared, but in Chamani et al.'s multi-user scheme documents can be unshared. They replicate MITRA once for each user u and implement an additional unshare protocol. Chamani et al. assume that PRF F is known to the corpus owner and to the users. This PRF is used in all schemes to construct a hash map as an index, i.e. to pseudorandomly encrypt the keywords.

Contrary to earlier implementations, documents can be uploaded to the database after the initial setup. Similar to previous implementations making use of a PRF, the output space is determined at setup; once the output space is determined, it can't be changed. To still allow the uploading of documents after the initial setup Chamani et al. scale the output space with the expected maximum corpus size and the expected final set of users. In essence, despite being "dynamic", the implementation is still of fixed size but with built-in (extra) expansion room. We address this artificial construction for extra corpus space in section 5.3 formally by introducing new terminology to cover this.

3.4.1 MITRA

Corpus setup: In MITRA, the single user scheme, the initial setup involves generating an empty index I and a keyword counter list Cnt of counters $Cnt[w]$, one for each $w \in W$.

Document uploading: The corpus owner can upload a document d after the initial setup by increasing the keyword counter $Cnt[w]$ for each keyword $w \in d$ by 1 or setting $Cnt[w]$ to 1 if the keyword is encountered for the first time. Using F and owner key K he creates for each $w \in d$ an index update of the form $(addr, val)$, where $addr = F_K(w, Cnt[w]||0)$ is the address to be updated and $val = F_K(w, Cnt[w]||1) \oplus (id(d)||add)$ is the (encrypted) updated value. This value is an encrypted document identifier. The authors note that collision between different addresses can occur, but the probability of occurrence can be decreased by ensuring a large enough output space for F.

Document updating: To update a document d the corpus owner creates a list of keywords to be added or deleted $WList$. To add the keywords he generates address-value tuples similarly to uploading. To delete the keywords from the database the corpus owner sends for every $w \in WList$ and user that has access to the document $u \in uSet(d)$ an index update of the form $(addr, val)$, where $addr = F_K(w, Cnt_u[w]||0)$ is the address to be updated and $val = F_K(w, Cnt_u[w]||1) \oplus (d||delete)$ is

the (encrypted) updated value. The keyword counter will not be decreased.

Searching: To search for a keyword w the user looks up counter $Cnt_u[w]$. He gives a query containing addresses $\{F_K(w, i||0) \Big| 1 \leq i \leq Cnt_u[w]\}$ to the server which returns the values at the specified addresses. The client decrypts the returned values by XORing the i-th value with $F_K(w, i||1)$.

3.4.2 Adapting MITRA

Multiple users: To adapt MITRA to the multi-user setting users are given their own key, namely a user key K_u. This user key is used for address generation and encryption when a document is shared with u. Chamani et al. envision that all users have their addresses/values in the same index and thus – to avoid collision – the output space of the PRF needs to be expanded accordingly. To enroll a user into the system, the corpus owner generates a user key K_u, stores it locally, and sends it to the user for local storage.

Document sharing: To share a document d with u the corpus owner generates the addresses and values under the user's key (rather than the owner key) and adds them to the Server: $(addr, val) = (F_{K_u}(w, Cnt_u[w]||0), F_{K_u}(w, Cnt_u[w]||1) \oplus (d||add))$. The corpus owner updates the user's keyword counters and notifies the appropriate parties. Lastly, he adds u to $uSet(d)$.

Document unsharing: To unshare a document d the corpus owner needs to update all addresses with a value containing $id(d)$ and remove u from $uSet(d)$. We will now discuss the two methods of storing the keyword counters.

3.4.3 O-μSE

In O-μSE the keyword counters Cnt_u are stored serverside in OMAPs. Instead of using the counter list Cnt initialised at the enrollment of user u, a user-specific OMAP $OMAP_u$ of size $|W|$ is initialised for each user. $OMAP_u$ contains Cnt_u and is encrypted using K_u – the user key that is known only to the corpus owner and the user – and stored serverside. When a user wants to generate a search query for keyword w he first retrieves the keyword counter value from $OMAP_u[w]$. He uses the retrieved value to construct the query as described in 3.4.2. Note that all actions/updates that change a keyword counter are followed by the corpus owner sending an updated version of $OMAP_u$ to the server. Additionally, an OMAP shuffles blocks after it has been accessed, meaning that search operations also update the OMAP.

Performance of O-μSE

- Query size is linear in the number of documents a user has access to $\mathcal{O}|D_u|$.
- Storage size is determined at setup and the required space is linear in the number of keywords per document and in the number of share queries $\mathcal{O}\Big(\sum_{d \in D} |W(d)| \cdot |AccList(d)|\Big)$.
- Search time is linear in the number of documents that contain the searched keyword $\mathcal{O}|D_u(w)|$. This is optimal.

3.4.4 Q-μSE

In Q-μSE the keyword counters Cnt_u are stored locally at user u. At enrollment of user u a user-specific update list $Queue_u$ of size $|W|$ is initialised by the corpus owner and stored at the server in $Queue$. Whenever the corpus owner changes a keyword counter's value, he stores the new values of the counter in $Queue_u[w]$ and instructs the server to update accordingly. When a user wants to generate a search query for keyword w, he first fetches the contents of its queue from the server. He then updates his keyword counters as specified and uses the appropriate keyword counter values (that have possibly been updated) to construct the query as described in 3.4.2.

The attentive reader might ask why in Q-μSE the keyword counters are stored locally and not at the server if the user is still required to request the queue update values. This is explained in 11.1: it is much simpler to add verifiability to Q-μSE than to O-μSE and this is why Q-μSE is an important implementation.

Performance of Q-μSE

- Query size is linear in the number of documents a user has access to $\mathcal{O}|D_u|$.
- Storage size is determined at setup and the required space is linear in the number of keywords per document and in the number of share queries $\mathcal{O}\Big(\sum_{d\in D}|W(d)|\cdot|AccList(d)|\Big)$. User storage size is stated to be linear in the number of keywords in the corpus $\mathcal{O}(|W|)$. It should be noted though that as the corpus is assumed to be dynamic, a new keyword $w \notin W, (w \in \Delta)$ can be introduced requiring the entire keyword counter list to be reworked. This can be prevented by having an entry for every word in the dictionary though this could be expensive (3.3.4).
- Search time is linear in the number of documents that contain the searched keyword $\mathcal{O}|D_u(w)|$. This is optimal.

3.4.5 Final notes

Chamani et al. use an authorisation list $uSet$ stored on the server to indicate which documents a user can access, similar to Patel et al. They state that it is a set of authorisation tokens though the tokens are never specified. Furthermore, they introduce it as under the symbol $AccList$, a list of list consisting of the user identifiers that have access to a given document. It is used in [4]'s security definitions and – although not indicated in the Update protocol by them – by the Update protocol. The Update protocol uses it to only update the entries of users having access to the updated document. Because of this, the authorisation list is an essential part of the implementation which we have chosen to store serverside as $uSet$.

The authorisation set $uSet$ does indeed not provide security, for Chamani et al. have a necessarily-weakened security guarantee for Unshare: "A first observation regarding unsharing in our security model where corrupt users may collaborate with the server is that this "strong" goal is *unobtainable*: The server can always store the old version of the encrypted dataset ... and keep sharing it with the user. Hence, a more realistic goal in this setting is to ensure that the user *cannot learn any information about future versions of the unshared document*" [4]. Therefore, the only practical use of $uSet$ is to allow the corpus owner to update the entries for all users who have access to the revised document.

In the description given by Chamani et al. document keys K_d are mentioned for backwards compatibility. Chamani et al. do not use document keys and as such we omitted them in our description above and we do not require the storage thereof (cf. section 3.5).

Finally, just like Wang et al. the keyword counters are at least the size of $\mathcal{O}\Big(|W| \cdot |U|\Big)$ (3.3.4), which can be large.

3.5 Auxiliary information and the database

Each of the implementations above expects the corpus owner, user, and Server to store some information locally. For the corpus owner and user this is done in the local information ι and ι_u respectively; for the Server this is done in the database DB. As explained before, auxiliary information contains the attributes that are not shared across *all* implementations. We specifically remarked how this makes auxiliary information dependent on the implementations considered. In our case there is always at least one implementation that differs from the others concerning the storage of an attribute. Therefore, when comparing the four different implementations we have equality between the local information and the auxiliary data. Since the papers differ in the storage requirements, we will now provide an overview of the auxiliary information stored by the corpus owner in table 1, the auxiliary information aux_u stored by the user in table 2, and the information stored by the Server DB in table 3.

Clearly, in MKSE by Hamlin et al. $aux_u \nsubseteq aux$ since $K_u \in aux_u$ but $K_u \notin aux$. Similarly, for Patel et al. $\{pointer(\sigma)\} \in aux_u$ and $pointers(\sigma) \notin aux$. For the other papers $aux_u \subsetneq aux$.

	Hamlin	**Patel**	**Wang**	**Chamani**
K	✗	✓	✓	✓
K_u	✗	✓	✓	✓
K_d	✓	✓	✗	✗
Cnt	✗	✗	✓	✓

Table 1: Information stored by the corpus owner in ι/aux

Note that the corpus owner can also be required to store keyword counters for the entire corpus, which we will denote by $Cnt_{\Theta} \in Cnt$, slightly abusing notation.

	Hamlin	**Patel**	**Wang**	**Chamani**
K_u	✓	✓	✓	✓
K_d	✓	✓	✗	✗
$pointer(\sigma)$	✗	✓	✗	✗
Cnt_u	✗	✗	✗(NFNU) ✓(FU) ✗(FNU)	✗(O-μSE) ✓(Q-μSE)

Table 2: Information stored by a user in ι_u/aux_u

[9] This is the set of document accesslists $AccList(d)$, cf. section 3.4.5

	Hamlin	**Patel**	**Wang**	**Chamani**
$xSet$	✓	✓	✓	✓
$uSet$	$\sigma_{u,d} = (r, P)_d$	$\sigma_{u,d} = F(K_d^1, d) \cdot F^{-1}(K_u^2, d)$	✗	✓[9]
Cnt	✗	✗	OMAP (FNU) ✗ (NFNU, FU)	OMAP (O-μSE) Queue (Q-μSE)

Table 3: Information stored by Server in the database

4 Unification of notation and terminology

4.1 Consistency across symbols

Since Curtmola et al. [6] published the first version of their paper back in 2006, the lexicon has evolved significantly. Changes have occurred in security definitions, standards, and convention regarding symbolism and terminology. These changes stem from author's preferences, the evolution of techniques used, and from different settings lending themselves to different terminology. We will now establish a unifying lexicon to remove any ambiguity from the unified security definition. In the next section we will use these symbols and unify the protocols.

Our unifying lexicon will include all the symbols that are used throughout the papers, only excluding the symbols that occur in specific contexts such as proof environments and are never used again. In these contexts the meaning of the symbol is clear from the immediate context. Our approach focuses on the symbols central to our discussion. The papers referenced are [6] and the four whose security definitions we will unify: [13], [20], [33], and [4]. Generally speaking, we chose the symbols on which the majority of papers agreed. For example, the security parameter is denoted as λ by all papers except [6]. Therefore, λ is the symbol that most clearly represents the security parameter. There are exceptions to this rule however.

Consistency over convention: Sometimes we break this rule for the sake of consistency over convention: we let the choice of one symbol influence other symbols for uniformity. For instance, we use w for a keyword per convention, as w is consistent across *all* papers. However, we then choose to denote the set of keywords by W and the keywords in a given document d by $W(d)$. The notation $W(d)$ differs from the generally agreed upon symbol $Kw(d)$ but is more consistent. It is both self-consistent – meaning that since the keywords are denoted by W then it makes sense to have the keyword set restricted to a document be denoted by $W(d)$ and vice versa – and this notation aligns better with similar cases. For example, $D(u)$ is the set of corpora/documents[10] restricted to those that u has access to.

Now, it should be noted that the reference papers sometimes lack an instance symbol for a given set symbol. For example, both [13] and [20] have a symbol for the set of corrupted users ($\mathcal{D}_c$ and $\mathcal{C}$ respectively) but do not have a symbol referencing a specific corrupted user. Instead they write "for $i \notin \mathcal{D}_c$" (Hamlin et al.) or "for $u \in \mathcal{C}$" (Patel et al.). In such cases, we have added an instance symbol influenced by the set symbol for consistency. In this example, a corrupted user is denoted by c.

[10] This depends on the setting: MKSE or MUSE.

Normal-styled symbols: Among the papers special-styled symbols exists. Examples of these are the mathcal-package symbols such as $\mathcal{A}$ and $\mathcal{C}$. We have almost exclusively chosen the normal-styled symbols with exception to the adversary and simulator, which are algorithms, and the leakage, which is convention.

Now follows an overview of the unified symbols (tables 4-7) alongside the corresponding symbols used by the other papers.

4.2 Unification tables

Term	Our symbol	Curtmola et al.	Hamlin et al.	Patel et al.	Wang et al.	Chamani et al.
Corpus owner set	O	-	$\mathcal{D}$	-	-	-
Corpus owner	$\texttt{o}$	O	i; j;	Manager; Corpus-Owner (old)	Owner	Owner
User set	U	G	$\mathcal{Q}$	U	U; $\mathcal{U}$	U
User universe	-	$\mathcal{U}$	-	$\mathcal{U}$	-	-
User identifier	u	U	i; j	u	u	u
Corrupted user identifier	c	-	-	-	-	-
Coalition (of corrupted users)	C	-	$\mathcal{D}_c$	$\mathcal{C}$	C	$\mathcal{C}$
Set of honest users, corpus owners.	U_{H}; O_{H}	-	-	-	-	-
Instance of an honest user, corpus owners	h	-	-	-	-	-
Adversary	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	-	Adv	Adv
Simulator	$\mathcal{S}$	$\mathcal{S}$	$\mathcal{S}$	$\mathcal{S}$	Sim	Sim
Distinguisher	$\mathcal{D}$	$\mathcal{D}$	D	-	-	-

Table 4: Players/roles

A discussion on the the symbols d, S, T_S and the lack of a corpus symbol by Hamlin et al. can be found in section 3.1.2.

Term	Our symbol	Curtmola et al.	Hamlin et al.	Patel et al.	Wang et al.	Chamani et al.
Corpus	D	$\mathbf{D}$	-	$\mathcal{D}$	D	D
Encrypted corpus	$xSet$	$\mathbf{c}$	-	$xSet$	$\mathbf{DictW}$	$xSet$; entry $xSet'$; $\mathbf{DictW}$
Document identifier	d; $id(d)$	$\mathrm{id}(D)$; D	d	d	id	id
(Perfect) hash table	P	-	D'; entry D	-	-	-
Keyword	w	w	w	w	w	w
Keyword set	W	-	-	-	-	$WList$
Corpus, distinct keywords	*Assumed*	$\delta(\mathbf{D})$	-	-	W	-
Id's of documents in D containing w	$D(w)$	$\mathbf{D}(w)$	-	-	$D(w)$	$DB(w)$
Document's keywords	$W(d)$	$\delta(D)$	S; T_S	$\mathrm{Kw}(d)$	$\mathbf{Kw}(id)$	$Kw(id)$
Dictionary	Δ	Δ	$\mathcal{U}$	$\mathcal{W}$	Λ	Λ
Corpus space	2^{Δ}	2^{Δ}	$\mathcal{U}$ (sic)	-	-	-
Metadata	$meta_d$	-	-	$meta_d$	-	-
Auxiliary information	aux	-	-[11]	-	aux^D	aux^D
Local information	ι	-	-	-	σ	σ

Table 5: Document and keyword related symbols

In section 2 we described how we can use set symbols instead of instance symbols to denote sets. Possible confusion arises then when auxiliary information on the corpus aux^D is an instance symbol. There is no aux^d. As seen in table 5, to avoid confusion we use aux (and aux_u if it belongs to user u).

[11] The auxiliary information used by Hamlin et al. is additional information given to the attacker and not information stored by the user or corpus owner.

Term	Our symbol	Curtmola et al.	Hamlin et al.	Patel et al.	Wang et al.	Chamani et al.
Security parameter	λ	k	λ	λ	λ	λ
Owner key	K; $K_{\odot}$	K; mk; K_O	-	K; $\mathcal{K}$	K	K
User key	K_u	uk_U; K_U	K^u	K_u; $\mathcal{K}_u$; $\tilde{K}_u$	K_u	K_u
User's set of secret keys[12]	$UserKeys$	-	-	-	-	**UserKeys**
Document key	K_d	-	K; K^d	K_d; $\mathcal{K}_d$; $\tilde{K}_d$	K_{id}[13]	-
Share key	σ	-	$\Delta_{u,d}$	$U_{u,d}$	-	-
Encrypted index	I	I	-	-	**DictW**	$xSet$; entry $xSet'$; **DictW**
Trapdoor	t	t	t^w	qSet; $H(u,w)$ (hash function H)	-	-
Authorisation token set	$uSet$	-	-	$uSet$	-	$uSet$; $AccList$
Leakage	$\mathcal{L}$	$\mathcal{L}$	$\mathcal{L}$	$\mathcal{L}$	$\mathcal{L}$	$\mathcal{L}$
History	H	H	-	-	-	-
Access pattern	$\alpha(H)$	$\alpha(H)$	-	-	-	-
Search Pattern	$\sigma(H)$	$\sigma(H)$	-	-	-	-
Graph	G	-	G	G	-	-
Edge set	E	-	E	-	-	-
Negligible function	$negl(\lambda)$	$negl(\lambda)$	$negl(\lambda)$	$negl(\lambda)$	$\upsilon(\lambda)$	$\upsilon(\lambda)$

Table 6: Keys and security symbols

An authorisation token can be considered the generalisation of a share key. The authorisation tokens are actually used by [20] and [4] and indicated with U, hence the choice of $uSet$ in table 6. As such, we have chosen $uSet$ as the authorisation token set which includes the share keys instead of Σ, which is used per convention for an SSE scheme as evidenced by table 7.

We would like to draw the reader's attention to both the history and the set of honest users. Here, H represents the history whilst the H in O_{H} means honest. Note the different styling.

[12] This is the set of all secret keys available to the user, including the user key and document keys.
[13] Document keys are not required by Wang et al. but they allow document keys as part of the more general auxiliary information.

Term	Symbol	Curtmola et al.	Hamlin et al.	Patel et al.	Wang et al.	Chamani et al.
Mode (e.g. *share*)	mod	-	-	-	-	mod; op[14]
Search reply	$Result$	X	-	Result	-	$IdSet$
Query history list	$qSet$	-	-	$qSet$	$\mathcal{Q}$	Q
Query	q	-	q	$\mathbf{H}(u, w)$; qct_d	q	q
User's access list	$Access_u$	-	-	Access(u)	**Access**(u)	$Access(u)$
Document's access list	$AccList(d)$	-	-	AccList(d)	Ulist(id)	$AccList(id)$; $AccessList(id)$
(External) Database	DB	-	-	-	EDB	DB; EDB
Scheme	Σ	SSE	-		Σ	Σ

Table 7: Server-related symbols and miscellaneous

4.3 Synonyms

Besides using different symbols for the same concepts, the papers[15] sometimes vary in vocabulary as well. Below is an alphabetically ordered list of common synonyms for our chosen terminology.

- Auxiliary information; auxiliary data.
- Corpus; document collection; dataset; file set.
- Corpus space; universe (cf. below).
- Dictionary; alphabet; universe (cf. below).
- Document; unprocessed set (i.e. a set of unencrypted keywords; cf. section 3.1.2).
- Encrypted document; processed set (i.e. set of encrypted keywords; cf. section 3.1.2).
- Document key; "private key (associated to the SSE instance for document d)" by [20]; sometimes called data key, which is the preferred nomenclature when one key is used to encrypt multiple documents, i.e. one chunk of data.
- Encrypted corpus; encrypted index (in specific cases; cf. below).
- Encrypted index; encrypted corpus (in specific cases; cf. below).
- Local information; locally stored information; private memory; private storage.
- Owner key; when only one corpus owner is present, the owner key is a master key.

[14] Mode *mod* is used for the Share protocol and *op* for the Update protocol.
[15] Again, the papers [6], [13], [20], [33], [4].

- Share key; Patel et al. also call this a search token or an authorisation token.
- Trapdoor; token; in some papers "query" can refer to both the trapdoor and to the query itself.
- User; querier (cf. below).
- User key; query key; user's "secret key".

We would like to add to this that a document and document identifier are often both denoted by d by the papers. Only when a clear distinction needs to be made between a document and its identifier does the symbol $id(d)$ exist. We use the same convention. As such, a search result can return d instead of $id(d)$. The papers and we as well only use $id(d)$ explicitly when using d would lead to significantly different results.

Corpus space: Now, we would like to make a few observations regarding the language used across the different papers. On such observation is that the term "corpus space" has not previously been used. We have adopted this term to describe what Curtmola et al. refer to as "the set of all possible documents with words in Δ" and what Hamlin et al. call "universe" (N.b. that Hamlin et al. use "universe" both for the set of all possible keywords and the set of all possible documents with words in this keyword space.)

Encrypted index/corpus: Another observation is that "encrypted index" and "encrypted corpus" can become synonyms when the documents are encrypted as a set of keyword-document identifier tuples. Both Wang et al. [33] (section 3.3) and Chamani et al. [4] (section 3.4) uploaded documents as $(addr, val)$ tuples. These tuples served both as part of an index and as an encrypted document. Thus the encrypted corpus is simultaneously an encrypted index. As a result, the encrypted index I and encrypted corpus $xSet$ can be used interchangeably for these two works (but not for [13] and [20] whose works were indexless). This is why [33] and [4] only consider an encrypted index or encrypted corpus and not both. This can be seen in table 5 and 6) where the index and encrypted corpus share the same symbol. In our work we will generally use the symbol for an encrypted corpus $xSet$. For example, when uploading the corpus it means only the encrypted corpus for the indexless implementations but it simultaneously means the index for the implementations with index.

Owner key: The papers that concern themselves with the multi-user setting always have one corpus owner who owns one master key. In the multi-key setting there can be multiple corpus owners. These corpus owners use their keys in the same fashion as the corpus owners in MUSE, though restricted to their own corpora. As such, we have decided to call these keys owner keys. Since MUSE is MKSE restricted to one corpus owner we always use the term owner key despite [13] not using such a key. This is because the owner key is a good generalisation of the master key and it allows amongst others that the protocols we define can be used in both MKSE and MUSE. We use K for the owner key or K_{Θ} if the owner of the key needs to be made explicit.

Auxiliary data, metadata, and local information: We remind the reader that auxiliary data and metadata are not the same. Metadata $meta_d$ is part of a document and contains extra information on the document (e.g. creation time, snippet, etc.) while auxiliary data is locally-stored information that is scheme specific.

Queriers: The concept of users and queriers are closely related. Queriers are however a subset of users. Given a set of queries, they are performed by a set of queriers. Every querier is a user, but not every user is necessarily a querier.

Dictionary: Note that the term "dictionary" as used by Chamani et al. refers to a key-value data structure. This differs from Curtmola et al., who use the term for the keyword space.

5 Unification of protocols

The changes in the lexicon since Curtmola et al. can also be seen in the protocols used and their respective names. As different implementation methods were chosen and more functionalities were added to a scheme, different protocols were needed. Using the unified lexicon (cf. section 4.2/tables 4-7) we will first unify the definition for a static SSE scheme, then we will unify the definition for a dynamic SSE scheme. Finally, we will show how our definition can be applied to the different papers.

5.1 Static SSE

5.1.1 Algorithms and protocols

We begin by defining a static SSE scheme. Static SSE means that the corpus is unchangeable: no documents can be added, removed or updated. Documents can be shared with users on an ad hoc basis, though in practice it does not matter for security if the documents are shared at Setup or afterwards (section 6). Hence, even though the access rights are not static, the SSE scheme is still considered static. We envision the usage of the algorithms and protocols of a static SSE scheme in the following way:

- **OwnerKeyGen** is used by a corpus owner to generate his owner key. Each owner has one owner key that is used by the corpus owner to setup his database and to share documents.
- **DocumentKeyGen** is used by the corpus owner to generate document keys. Each document key is used to encrypt one document (and is thus also used when sharing that document). DocumentKeyGen is expected to be a subalgorithm of the Setup algorithm.
- **UserKeyGen** is used to generate a user's user key. In MKSE this is done by the user himself. In MUSE UserKeyGen is expected to be a subalgorithm of the Setup algorithm, meaning that the keys are generated and distributed by the corpus owner.
- **Setup** is used by a corpus owner to setup his database and enroll a user set into the system.
- **Share** is used by a corpus owner to grant an enrolled user access to – i.e. search rights over – a document. Access granting is done on a per-document basis for one user.
- **Search** is used by a user to enquire from the server which documents that he has been granted access to contain the keyword that he specified.

Our envisionment for a static MUSE scheme is illustrated by figure 5, where DocumentKeyGen and UserKeyGen are subalgorithms of Setup, and where Search is an interactive protocol between the server and a given user. Note that there are multiple users, each of which interacts with and is

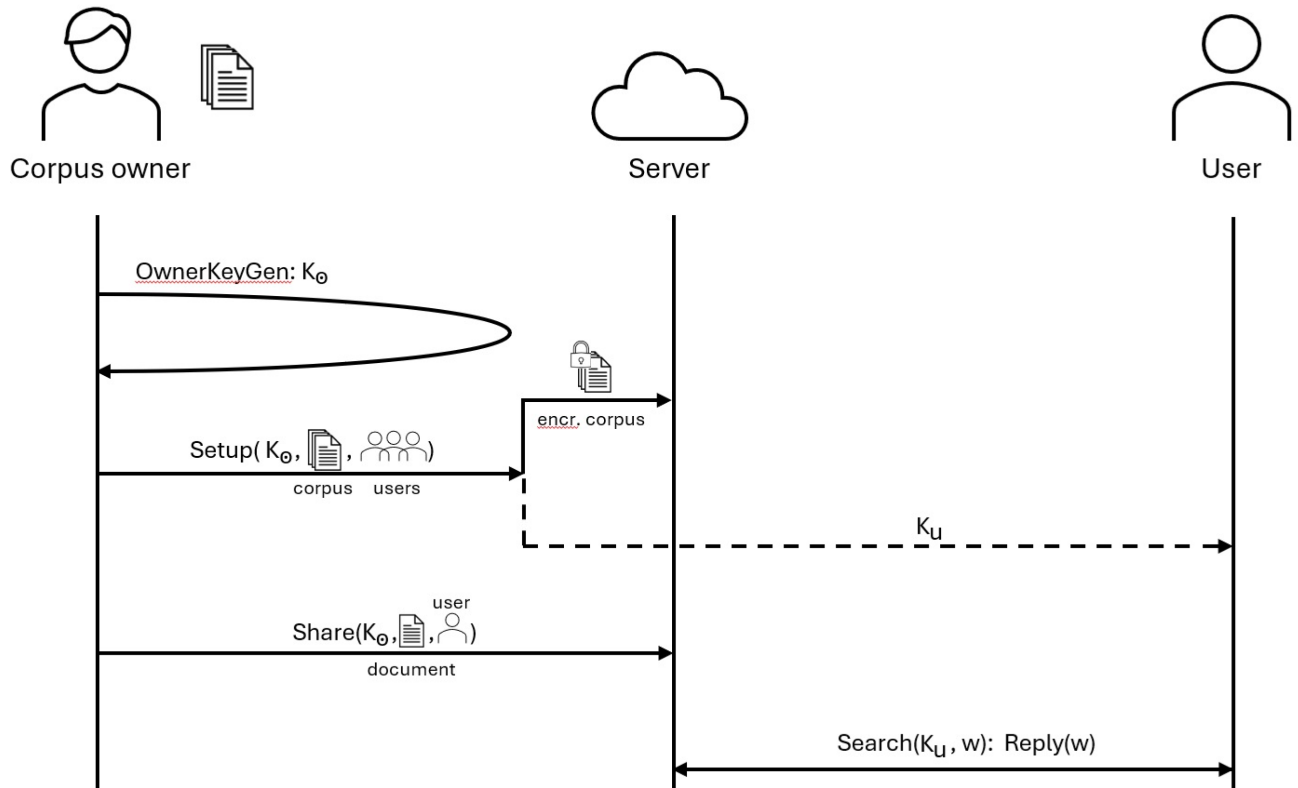

Figure 5: General idea of the static MUSE algorithms/protocols

interacted with the other parties as specified. Now, Share is not restricted to single-use but can be used for each document and user combination separately by the corpus owner. Users are enrolled at Setup but can only (meaningfully) search when at least one document is shared with them.

Our envisionment for a static MKSE scheme is illustrated by figure 6. We stress that the figures illustrate protocol usage and as such that in MKSE *each* corpus owner interacts with the server and with the users according to the protocols. The general idea behind the protocols differs from the MUSE scheme in who generates the user key.

5.1.2 Defining static SSE

Broad interpretation of keyword counters: It is tricky to generalise the keyword counter storage and usage. Often when we talk about keyword counters or the storage thereof we have to list both $Queue$ and $OMAP$ as possible output. The OMAP and queue are however mutually exclusive and thus – to shorten the list of parameters – we have chosen to interpret Cnt broadly: Cnt encompasses both the keyword counters and the methods of storage thereof ($Queue$ and $OMAP$). For example, the Setup algorithm can output to the corpus owner initialised keyword counters Cnt_u but to the Server it can output either queues $Queue_u$ or oblivious maps $OMAP_u$. Regardless of the implementation, it is clear that the Server has to deal with keyword counters or the updates thereof. This is properly

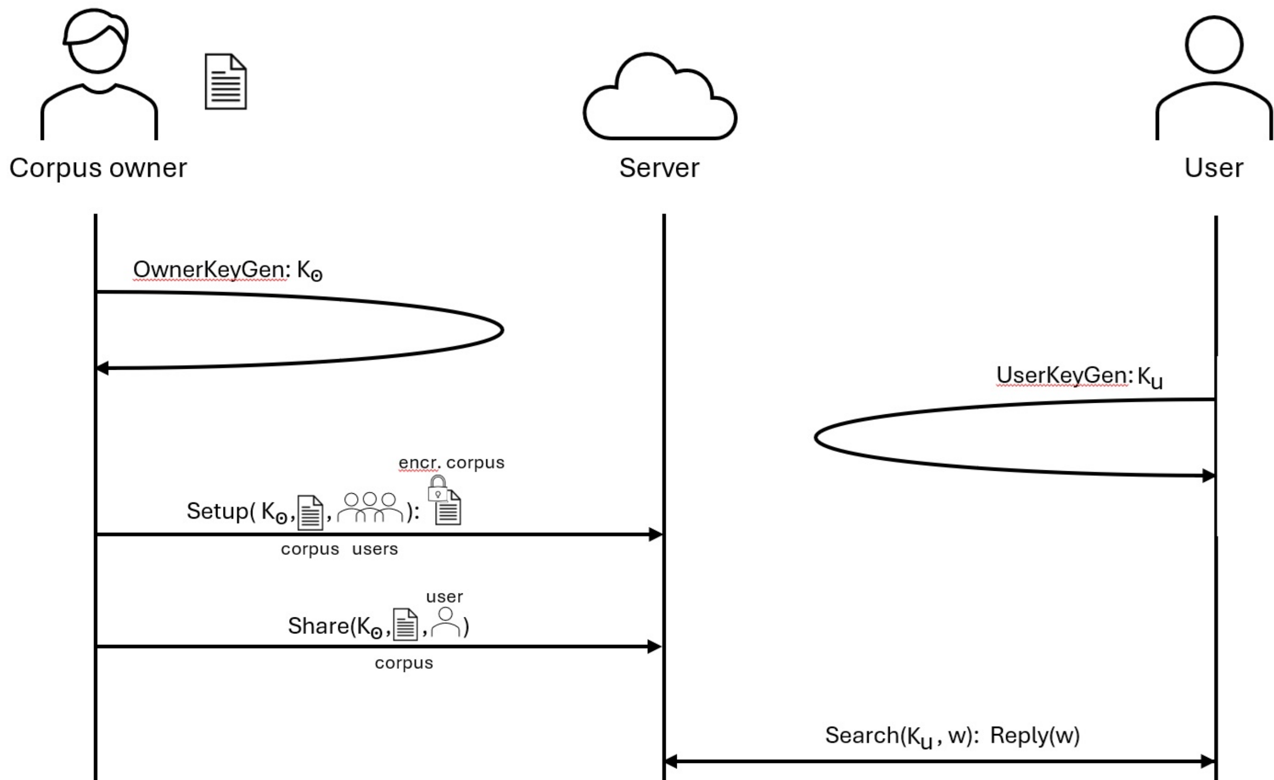

Figure 6: General idea of the static MKSE algorithms/protocols

conveyed by Cnt_u.

Only the first party provides a keyword counter: Another convention regarding the keyword counters is to require the keyword counter input only from the first party that inputs it. Take for example the Search algorithm. It can require the input of the user's keyword counters Cnt_u. These counters can however be stored Serverside in $OMAP_u$ requiring only the OMAP as input, or it requires the user's locally stored keyword counter Cnt_u which requires the Server's $Queue_u$ as input as well. We have decided to require the Cnt_u input only from the user, since ultimately he is the one who uses the keyword counters to construct the query.

Output to the invoker: Note that algorithms output only to the invoker of the algorithm, which generally is the corpus owner. When an algorithm then additionally outputs to a user or to the Server, it means that everything is outputted to the corpus owner and that he is responsible for sending the correct information to the respective party.

Semicolon to separate parties: In our definition for static SSE the in- and outputs of different parties are separated by semicolons (;), The in-/output before the semicolon is from/to the corpus owner and the users, and the in-/output after the semicolon is from/to the Server.

Definition 3 (Static Searchable Symmetric Encryption)**.** *An Searchable Symmetric Encryption (SSE) scheme over a dictionary Δ is a collection of 6 polynomial-time algorithms and protocols $\Sigma = ($OwnerKeyGen, DocumentKeyGen, UserKeyGen, Setup, Share, Search$)$. It contains the key-generation protocols:*

$K \leftarrow$ *OwnerKeyGen*(1^λ)*: is a probabilistic key generation algorithm. It takes as input security parameter λ, and outputs an owner key K.*

$K_d \leftarrow$ *DocumentKeyGen*$(1^\lambda, d)$*: is a probabilistic key generation algorithm. It takes as input security parameter 1^λ and document d, and outputs a document key K_d.*

$K_u \leftarrow$ *UserKeyGen*$(1^\lambda, u)$*: is a probabilistic key generation algorithm. It takes as input security parameter 1^λ and user identifier u, and outputs the user key K_u.*

and furthermore the algorithms/protocols

$(K_U, [K_D], [Cnt]; xSet, [uSet]) \leftarrow$ *Setup*$_K(D, U, [|W|], [N])$*: is a probabilistic algorithm to setup an initial database and enroll a user set. It takes as input owner key K, initial corpus D, an a user set U for enrollment. Depending on the implementation it also requires as input the final number of keywords in the corpus $|W|$, and/or the total size of the final corpus N. It outputs an encrypted corpus $xSet$ to the Server; it possibly outputs to the Server an authorisation token set $uSet$ and a list of keyword counters Cnt; for each user $u \in U$ it invokes the UserKeyGen algorithm and it outputs to the corpus owner the users keys K_U; it possibly outputs to the corpus owner the document keys K_D, and the keyword counters Cnt.*

$\big([pointer(\sigma_{u,d})], [K_d], [Cnt'_u]; xSet', [uSet']\big) \leftarrow$ *Share*$_K(u, d, [K_d], [Cnt_u])$*: is a probabilistic protocol to share an uploaded document with an enrolled user. It takes as input owner key K, user identifier u and document d. For some implementations it also takes as input the document key K_d and the user's keyword counters Cnt_u. It outputs to the Server encrypted entries $xSet'$, and for some implementations also an update to the authorisation token set $uSet'$. For some implementations it outputs to the user a pointer to the share key $pointer(\sigma_{u,d})$, document key K_d, and an update to u's keyword counters Cnt'_u.*

$(Result, [Cnt_u]) \leftarrow$ *Search*$_{K_u}(w, [(id(d), K_d)_{d \in D_u}], [Cnt_u]; xSet, [uSet])$*: is a deterministic protocol to search over the part of the database accessible to the user/querier. It takes as input from the user his key K_u and keyword w, and from the Server the encrypted documents $xSet$; it possibly takes as input from the user the document identifier and key $(id(d), K_d)$ and/or the user's keyword counters Cnt_u; and it possibly takes as input from the Server the authorisation token set $uSet$. It outputs to the user the search result $Result$, which is the document identifiers that u has access to and that contain the queried keywords: $D_u(w)$. It possibly outputs to the Server and/or the user an updated keyword counter Cnt_u.*

No enroll algorithm: We note that our definition does not use an Enroll algorithm. This is because, security-wise, all covered papers that implement an Enroll algorithm assume in their security game that all users are enrolled immediately after the uploading of the corpus. Thus, security-wise there is only a document-upload-and-user-enrollment phase – which we implement via the Setup algorithm – followed by a Share phase, followed by a Search phase.

Interactive Search protocol: For SSE schemes we use an interactive Search protocol run by the user and the Server. In [13] and [20] there are always 2 rounds of communication: the user sends his

search query to the Server and the Server sends back his reply. In these cases we will split the Search protocol in two algorithms: Query and Search. Query is run by the user to generate the input for the Search query q. He then sends q to the server, who executes the Search algorithm and sends back the response $Result$. For the exact in and outputs for [13] and [20] we refer back to sections 3.1 and 3.2 respectively.

No Decrypt: The algorithm "Dec[rypt]" is used by Curtmola et al. to decrypt retrieved documents. It is therefore not part of newer SSE schemes (cf. section **??**) since it does not concern itself with security. Furthermore, "SearchReply" is used by Patel et al. to let the server send its reply to a querying user. The other papers consider an interactive Search-protocol which includes the Server's reply. We have chosen the interactive Search protocol over requiring that the invokement of Search is immediately followed SearchReply.

5.2 Dynamic SSE

A static SSE scheme can be extended to become dynamic. This means that the corpus is no longer static and nor are the access rights. In dynamic SSE:

- Documents can be added to the corpus after the initial setup of the database.
- Documents can be updated, i.e. the keywords can be modified.
- Documents can be unshared from users, i.e. their search/access rights can be revoked on a per-document basis.

We envision the usage of the algorithms and protocols of a DSSE scheme in the following way:

- **Upload** is used by a corpus owner to add a new document to the database after the initial upload. It has DocumentKeyGen as a subalgorithm. Note that Upload does not grant any access rights. This is done by the Share algorithm.
- **Update** is used by a corpus owner to add keywords to or delete keywords from a document. It subsequently outputs updated encrypted entries for all users that the document has been shared with.
- **Unshare** is used by a corpus owner to revoke a user's access to a document.

Our envisionment for the extending protocols of a dynamic SSE scheme is illustrated by figure 7. Note that the protocols Update and Unshare do affect users, but do not provide output to the affected user(s). For an Unshare operation it is practically speaking desirable to notify a user, but theoretically speaking not necessary. For an Update operation the users do not even have to be notified.

Definition 4 (Dynamic Searchable Symmetric Encryption)**.** *An index-based Dynamic Searchable Symmetric Encryption (DSSE) scheme over a dictionary Δ extends an index-based SSE scheme (def. 3) with the following polynomial-time protocols (Upload, Update, Unshare) such that*

$xSet' \leftarrow$ *Upload*$_K(d)$*: is a probabilistic algorithm to add documents to an existing corpus. It takes as input owner key K or document key K_d and document d. It outputs encrypted entries $xSet'$.*

$(Cnt'; xSet') \leftarrow$ *Update*$_K(id, WList, mod, K_U, Cnt_U; uSet)$*: is a probabilistic protocol to change an uploaded document's keywords. It takes as input owner key* K*, document identifier* id*, a list of keywords that are added or removed* $WList$*, a mode indicator* $mod \in \{add, delete\}$*, all affected users their user keys* K_U *and keyword counters* Cnt'*, specifically including the corpus owners keyword counters; the Server provides input* $uSet$*. It outputs updated keyword counters* Cnt' *to the corpus owner, the Server and the respective user; it outputs updated encrypted entries* $xSet'$ *to the Server.*

$(xSet', uSet') \leftarrow$ *Unshare*$_K(d, u, Cnt_u)$*: is a deterministic protocol to revoke an enrolled user's access from an uploaded document. It takes as input owner key* K*, document* d*, user identifier* u*, and* u*'s keyword counters* Cnt_u*. To the server it outputs updated (i.e. overwritten) encrypted entries* $xSet'$ *and an updated authorisation token set entry* $uSet'$ *that now excludes* u *from access to* d*.*

On the usage of encrypted entries $xSet$ and its relation to encrypted index I, see section 4.3. Now, we would like to point out that Hamlin et al. do have an Upload algorithm, just like in Patel et al. the server can however perform a replay attack after a document has been shared. For security purposes then all Share queries occur effectively before any Search query happens. Effectively, this means that there first is a Setup phase and only after its completion is can users Search. Hamlin et al.'s implementation is security-wise then static, but the scheme does allow the uploading of documents after Share queries have happened, meaning that implementation wise it is partly-dynamic. "Partly-", because even though there is an Upload algorithm, there is no Update or Unshare algorithm. This paper

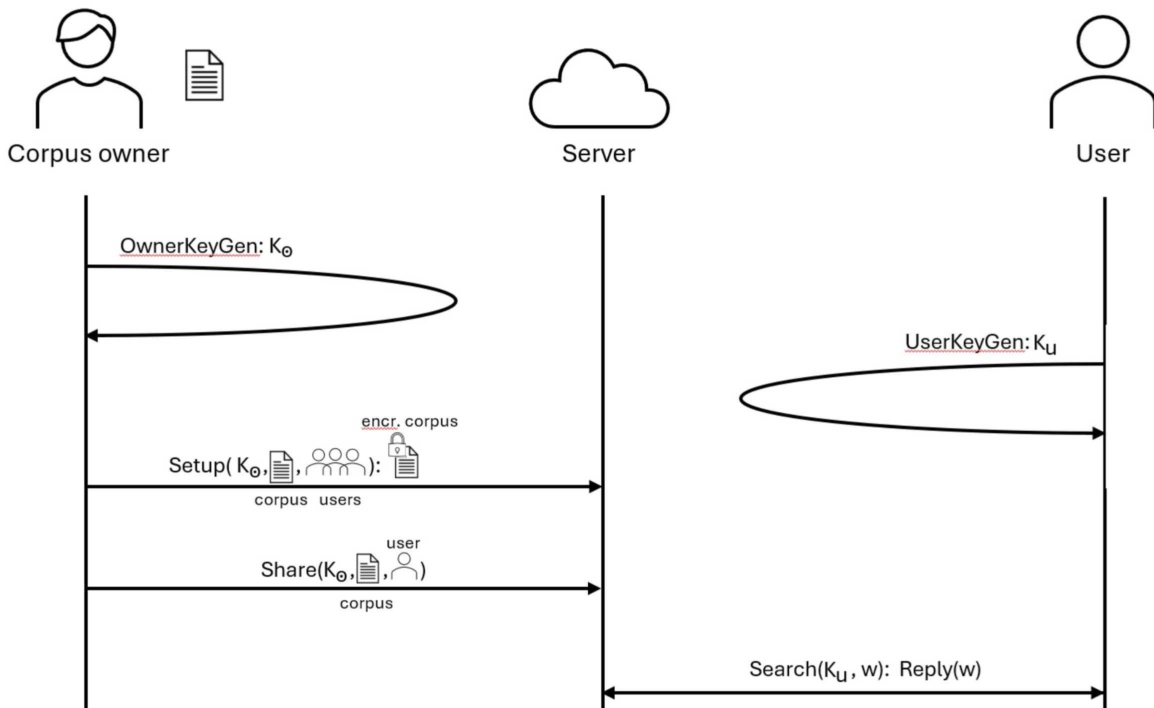

Figure 7: General idea of the static MKSE algorithms/protocols

covers the security of the different implementations and as such we consider [13]'s implementation to be static (cf. section 5.4).

5.3 Fixed-size and resizable DSSE

We would like to point out that according to our definition of DSSE it is possible for the implementation to require the corpus owner to indicate the final size of the corpus and (related to this) the final number of users. The paper by [4] requires this and is the only paper that allows the uploading of document after the initial setup (security-wise). As such, none of the papers have a dynamic SSE scheme that can grow as the circumstances change. We will now introduce the new notions of "fixed-size dynamic SSE" and "resizable dynamic SSE".

Definition 5 (Fixed-Size Dynamic Searchable Symmetric Encryption)**.** *A dynamic searchable symmetric encryption scheme that determines the final total size of the corpus with the invoking of the Setup algorithm is called fixed-size dynamic searchable symmetric encryption (FS-DSSE) scheme.*

Definition 6 (Resizable Dynamic Searchable Symmetric Encryption)**.** *A dynamic searchable symmetric encryption scheme whose total size changes as a result of invoking the Share and Unshare protocols and whose size is not determined during the invoking of the Setup algorithm is called a resizable dynamic searchable symmetric encryption (RS-DSSE) scheme.*

With the definitions above we can now distinguish between different implementations. It is important to distinguish between FS-DSSE and RS-DSSE for the following reasons:

- If the number of users is higher than initially suspected, the entire database needs to be redeployed. This requires a new Setup phase and the Sharing of the documents again.
- If the number of documents is higher than initially suspected or when many keywords are added to the documents, then the entire database needs to be redeployed as well.
- If the number of users is much lower than expected for at least a significant amount of time, then the database size could have been lower and the company could have saved on storage costs.
- If the number of documents is much lower than expected or the adding of documents to the database takes a significant amount of time, then the database size could have been lower and the company could have saved on storage costs.

The limitations mentioned above are indeed significant. As Chamani et al. note, that to "efficiently support updates on the outsourced database (without re-running the "expensive" setup), ... is a necessary property for most real-world applications." As such, applications with an ever-growing number of users or documents would consider "resizable" to be a necessary property. We note that [4] did not identify the fixed-size DSSE scheme as an open research question into a resizable DSSE scheme. We therefore consider this to be a novel identification of an open research question.

5.4 Application of our definition

Our protocols do not match one on one with the protocols used in the different papers. Therefore, we will now give a list-overview relating our definition to the protocols used by the different papers. For

the exact workings of the different protocols we refer back to section 3.

Empty initialisation: Note that algorithms and protocols can initialise a list, counter, etc. without giving it a concrete value, and that author often assume this initialisation without making mention of it anywhere. In our paper the initialisation always happens when it is the output of a given algorithm and protocol. This can mean that it is initialised without a value. For example, in Chamani et al. no keyword counter values are given when a user is enrolled into the system during Setup, and thus the user is given empty but initialised keyword counters in the case of Q-μ or the server is given an OMAP without values in O-μSE. This process is described in section 3.

A reminder: We remind the reader that auxiliary information aux_u is all information stored locally by the user, aux is all information stored locally by the corpus owner, and DB is all information stored by the Server. Their contents can be found in tables 1, 2 and 3. These tables are found in section 3, which also describes the different implementations covered in this paper. Furthermore, we are going to use the same convention for Cnt_u as used in section 5 (cf. section 5.1 for an explanation). This convention is for example apparent and of great use in the Search protocol by [4]. Note that [33] only have an implementation that stores the keyword counters serverside in OMAPs, thus allowing us to use the more specific $OMAP_u$.

Document keys in [20]: A specific note on document keys in Patel et al.: if a user inputs document key K_d to a protocol we assume that it is the document key $K_d = (K_d^1, K_d^3) \in aux_u$ and not $K_d = (K_d^1, K_d^2, K_d^3) \in aux$ (cf. section 3.2).

List-overview of the application of our protocols: We will now link our algorithms/protocols to those used by Hamlin et al. [13], Patel et al. [20], Wang et al. [33], and Chamani et al. [4].

OwnerKeyGen is used by a corpus owner to generate a owner key that can be used to setup an initial database with Setup.

- In [13] it is not implemented. Instead, document keys are used to setup the corpus and share it. Hamlin et al. elaborate a lot on the necessity of tying the document key to the share key and thus readers who are familiar with [13] might find it hard to adjust to these keys being called owner keys. Additionally, real-life corpus owners are modeled in the security game by a set of corpus owners, and therefore the real-life corpus owner currently has one document key for each document and not one owner key per document. One owner key per corpus would be contrary to our envisionment of an owner key as laid out in section 4.3.
- In [20] it is part of the protocol "EncryptDoc". EncryptDoc takes as input security parameter 1^λ and corpus D. It outputs owner key K to the corpus owner, and database DB to the Server.
- In [33] it is part of the protocol "Setup". Setup takes as input security parameter 1^λ, user set U for enrollment, and corpus D. It outputs auxiliary information aux to the corpus owner; to $u \in U$ it outputs aux_u; and it outputs DB to the Server.
- In [4] it is implemented by protocol "Setup". Setup takes as input security parameter 1^λ, final size of the index N, number of distinct keywords $|W|$, total number of users $|U|$, and initial corpus D. It outputs auxiliary information on the corpus aux to the corpus owner; and it outputs database DB to the Server.

DocumentKeyGen is used by a corpus owner to generate a document key that can be used to encrypt an individual document with Upload.

- In [13] it is implemented by the protocol "DataKeyGen". DataKeyGen takes in security parameter 1^λ, and outputs to the corpus owner a document key K_d.
- In [20] it is part of the protocol "EncryptDoc". EncryptDoc takes as input security parameter 1^λ and corpus D. It outputs owner key K to the corpus owner, and database DB to the Server.
- In [33] it is not implemented.
- In [4] it is not implemented.

UserKeyGen is used to generate a user key that can be used to encrypt in a way that grants access control.

- In [13] a user is assumed to have his own user key but the keys their distribution is not explained. Since there are multiple corpus owners, each user could well be expected to generate his own user key. UserKeyGen is implemented by "QueryKeyGen". QueryKeyGen takes as input security parameter 1^λ, and outputs to the invoking user the user key K_u.
- In [20] it is used by the corpus owner as part of the protocol "Enroll". Enroll takes as input security parameter 1^λ and user identifier u. It outputs user key K_u to both the corpus owner and u.
- In [33] it is part of the protocol "Setup". Setup takes as input security parameter 1^λ, user set U for enrollment, and corpus D. It outputs auxiliary information aux to the corpus owner; to $u \in U$ it outputs aux_u; and it outputs DB to the Server.
- In [4] it is used by the corpus owner as part of the protocol "Enroll". Enroll takes as input security parameter 1^λ and user identifier u. It outputs user key K_u. The tuple (u, K_u) is stored by the corpus owner locally, and K_u is stored by u in private memory.

Setup is used by a corpus owner to setup an initial database with initial documents and initial users. Note that the users do not have access rights at this point.

- In [13] it is implemented by "ProcessSet", which is used to upload a document. Remember that each corpus has a size of 1 document. As such, our envisionment of Setup is congruent with the ProcessSet algorithm by [13]. Contrary to the owner key in our envisionment, ProcessSet takes as input document key K_d and document d. It outputs an encrypted document $xSet'$ to the Server for inclusion in the database.
- In [20] it is part of the protocol "EncryptDoc". EncryptDoc takes as input security parameter 1^λ and corpus D. It outputs owner key K to the corpus owner, and database DB to the Server.
- In [33] it is part of the protocol "Setup". Setup takes as input security parameter 1^λ, user set U for enrollment, and corpus D. It outputs auxiliary information aux to the corpus owner; to $u \in U$ it outputs aux_u; and it outputs DB to the Server.
- In [4] it is implemented by protocol "Setup". Setup takes as input security parameter 1^λ, final size of the index N, number of distinct keywords $|W|$, total number of users $|U|$, and initial corpus D. It outputs auxiliary information on the corpus aux to the corpus owner, and it outputs database DB to the Server. Setup in [4] does not enroll an initial set of users.

Upload is used by a corpus owner to upload a document to an existing database.

- In [13] the corpus is static security-wise and therefore – from a security point of view – it is not implemented. Cf. section 5.2. For this reason the algorithm "ProcessSet" does not implement the Upload algorithm.

- In [20] the corpus is static and it is not implemented.
- In [33] the corpus is static and it is not implemented.
- In [4] it is part of the protocol “Share”. Share takes as input owner key K, user identifier u, document d, and mode $mod \in \{share, add\&share\}$. Share with $mod = add\&share$ outputs an encrypted document $xSet'$ for inclusion in $xSet$ to the Server; and it outputs updated keyword counters Cnt_u to the Server *and* to the corpus owner.

Search is initialised by a user once to search through all the document he has access to. The server returns all relevant document identifiers.

- In [13] it is implemented by the combination of “Query” and “Search”.
  * Query is run by a user and takes as input keyword w and user key K_u. It outputs query q to the user.
  * In Hamlin et al. Search is run by the server and takes as input a user-specific share key $\sigma_{u,d}$, query q and an encrypted document $xSet'$. In [13] the search protocol needs to be repeated over all documents a user has access to D_u, but since our Search protocol is invoked once by the user we assume that the server iterates Hamlin et al.'s Search protocol over all documents u has access to on its own. It then outputs $Result$ to the querier, which are the document identifiers of the documents that u has access to and that contain the queried keyword: $D_u(w)$.
- In [20] it is implemented by the combination of “SearchQuery” and “SearchReply”.
  * SearchQuery is run by the user and takes as input keyword w, user identifier u, user keys $K_u = (K_u^1, K_u^2)$, and for every document that u has access to the tuple $(id(d), K_d^1, K_d^3)$ consisting of the document identifier and two document keys. It outputs a query set $qSet$ to the user.
  * SearchReply is run by the server and takes as input a query set $qSet$. It outputs $Result$ to the querier, which are the document identifiers of the documents that u has access to and that contain the queried keyword: $D_u(w)$.
- In [33] it is implemented by “Search”. Search takes as input the user's key K_u, his auxiliary data aux_u, and keyword w, and from the Server DB. It then outputs $Result$ to the querier, which are the document identifiers of the documents that u has access to and that contain the queried keyword: $D_u(w)$. Depending on the implementation it also outputs $OMAP'$ to the server, even though this is not mentioned by [33].
- In [4] it is implemented by “Search”. Search takes as input user key K_u, auxiliary data aux_u and keyword w, and from the server DB. It outputs Cnt_u to the querier first – who uses it for address generation – and later it outputs to the user $Result$, which are the document identifiers of the documents that u has access to and that contain the queried keyword: $D_u(w)$. Finally, it outputs Cnt_u to the Server.

Share is used by a corpus owner to share an uploaded document with an enrolled user.

- In [13] it is implemented by “Share”. It is invoked by a corpus owner and takes as input an owner key K, a recipient u and his key K_u, and an encrypted document $xSet'$. It outputs share key $\sigma_{u,d}$ to the server.

- In [20] it is implemented by "AuthComputing". AuthComputing takes as input user identifier u, user key $K_u = (K_u^1, K_u^2)$, document d, and owner key $K = (K_1, K_2, K_3)$. It outputs share key $\sigma_{u,d}$ and pointer $pointer(\sigma_{u,d})$ to the Server; it outputs to u document key $K_d = (K_d^1, K_d^3)$ and pointer $pointer(\sigma_{u,d})$.
- In [33] it is implemented by "Share". Share takes as input document d, user identifier u, the corpus owner's auxiliary data aux, and database DB. It outputs encrypted entries $xSet'$ to the server, and updated keyword counters Cnt_u to the appropriate parties.
- In [4] it is part of the protocol "Share". Share takes as input owner key K, user identifier u, document d, and mode $mod \in \{share, add\&share\}$. Share with $mod = share$ outputs updated keyword counters Cnt_u to the Server *and* to the corpus owner; and it outputs an update $uSet'$ to $uSet$.

Enroll is used by a corpus owner to enroll a new user in the system. The user is granted access to document through the Share protocol.

- In [13] a user must generate his own user key by invoking the QueryKeyGen protocol (cf. our UserKeyGen).
- In [20] it is implemented by "Enroll". Enroll takes as input security parameter 1^λ and user identifier u. It outputs user key K_u to the user and K_u to the corpus owner.
- In [33] it is not implemented.
- In [4] it is implemented by "Enroll". Enroll takes as input security parameter 1^λ and user identifier u. It outputs K_u to to the corpus owner, and auxiliary information aux_u to u.

Update is used by a corpus owner to insert/remove a list of keywords in/from an existing document.

- In [13] it is not implemented
- In [20] it is not implemented.
- In [33] it is not implemented.
- In [4] it is implemented by "Update". Update takes as input owner key K, document identifier $id(d)$, a list of keywords $WList$, a mode $mod \in \{add, del\}$, the set of users that have access to the document $AccList(d) = uSet(d)$ and for every $u \in AccList(d)$ auxiliary information aux_u. It outputs the set of entries that the server needs to update $xSet'$ to the Server, and it outputs the users their updated auxiliary data aux_U to the respective user and to the corpus owner.

Unshare is used by a corpus owner to remove an enrolled user his access to an uploaded document.

- In [13] it is not implemented.
- In [20] it is not securely implemented
- In [33] it is not implemented.
- In [4] it is implemented by "Unshare". Unshare takes as input owner key K, user identifier u, document d, and u's keyword counters Cnt_u. It outputs to the Server updated – i.e. overwritten – encrypted entries $xSet'$ and an update to the authorisation token set $uSet'$ that excludes u from access to d. Note that the keyword counters are not updated (and thus not given as output).

6 Current security definitions

Different security definitions: Each of the papers currently uses his own security definition. The definition by Hamlin et al. is indistinguishability-based and written as a sequence of actions undertaken by the Adversary followed by actions undertaken by the Challenger; Wang et al. and Chamani et al. take the same approach as each other and have semantic security, which is a simulation-based definition where the real world is compared against an simulated world; and Patel et al. have a definition that compares the server's view against a simulated view. Patel et al.'s definition does differ slightly from that of [33] and [4] in that their definition does not involve an adversary with advantage but rather involves computational indistinguishability. We provide more information on that in section 6.2.

Indistinguishability: The intuitive notion behind indistinguishability-based security definitions is that an adversary provides two inputs of his choice to a challenger who executes the algorithms/protocols as normal, but that the adversary cannot distinguish between the returned outputs. The reasoning behind this is that if this is so for any input the adversary gives, then any discernible link between the input and output is broken. In SSE, the basic idea is that the adversary provides a challenger with two options for the corpus and a corresponding sets of queries. The challenger randomly selects one of these options and executes the scheme as instructed: he encrypts the corpus and executes the queries. The results are then returned to the adversary. The adversary then guesses which of the two options was executed by the challenger. If the adversary cannot know or make an educated guess on which option the challenger selected then the adversary has a $1/2$ probability of guessing correctly. To measure the security of the implementation we can thus examine how much better the adversary can guess correctly than $1/2$. What he can do better than $1/2$ is called his advantage. We say then that an implementation is secure if the adversary's advantage is not "significant". An indistinguishability game thus analyses an attacker's advantage to determine if a scheme is indistinguishably secure. Of course, an attacker with infinite computational power or unlimited time could always improve his odds significantly over $1/2$ by computing the outputs for all possible inputs. To prevent this, the computational power of an adversary is bounded polynomially in the security parameter (λ).

"Significantly" formalised: The question then remains what is a "significant" advantage? The adversary's success can always be written as $1/2 + \varepsilon(\lambda)$ where $\varepsilon(\lambda) : \mathbb{N} \to [0, 1/2]$ denotes the advantage function. Here, a success of $1/2$ represents random guessing. If the adversary consistently performs worse than random guessing, he can simply guess complement to achieve a reliable advantage. The function $\varepsilon(\lambda)$ is a measure of how much better than random guessing the adversary is capable of. An advantage of $\varepsilon(\lambda) \leq negl(\lambda)$, where the function $negl : \mathbb{N} \to \mathbb{R}$ is a negligible function, ensuring that $\varepsilon(\lambda)$ becomes insignificant (syn. negligible) as λ grows large. Formally, a negligible function $negl(\lambda)$ is such that for any positive polynomial $poly(.)$ and sufficiently large λ it holds that $|negl(\lambda)| < 1/poly(\lambda)$. If there does not exists such a negligible function for a given advantage $\varepsilon(\lambda)$ then the advantage is considered significant.

Semantic security: The intuitive notion behind semantic security is that an adversary provides input of his choice to a challenger, who either executes the algorithms/protocols as normal *or* simulates their executions. The execution of the protocols are called the "real world" and the simulated protocols are called the "simulated world" or the "ideal world". The goal is to ensure that the adversary cannot distinguish between the real world and the ideal world. If an adversary cannot distinguish

between the real and the ideal world, then the simulator running the ideal world must have access to all sources of information leakage. Suppose that there is a mechanism that leaks information to the adversary but that the mechanism is not given to the simulator; the adversary can then use this mechanism to gain a significant advantage over the simulator. This contradicts our initial assumption that the adversary cannot distinguish. Thus, when defining semantic security the leakage of the scheme needs to be made explicit and be given to the simulator. To show the security of a scheme, the information/leakage given to the simulator is kept to a minimum. A modern approach to semantic security is to define the simulator using a leakage function $\mathcal{L}$. Then, when an implementation is discussed, there is a section describing the leakage function for the given implementation. As Patel et al. note, this leakage function is defined as a description of information instead of a quantifiable value. This makes leakage hard to compare across schemes.

(Non-)adaptive adversary: Now, post-Curtmola there are two types of adversaries: selective/non-adaptive adversaries and adaptive adversaries. Adaptive adversaries are able to determine a query after the execution of previous queries and selective adversaries determine all queries before any of them is executed. In the case of Curtmola et al. where only Search queries existed, it meant that non-adaptive definitions only provided security to clients that generated their keywords in one batch, whilst adaptive definitions provided privacy even to clients who generated keywords as a function of previous search outcomes. For such definitions, it was shown by [6] that for non-adaptive adversaries semantic security and indistinguishability are equivalent. It was then shown by them that semantic security implies indistinguishbability when the adversary is adaptive, but equivalence was not shown nor disproven. It has since been assumed that semantic security is stronger than indistinguishability. In the following section we will discuss the individual security definitions. In these security definitions there are besides Search queries also Share queries, and in the case of dynamic SSE also Upload, Update and Unshare queries. All of these queries are generated in one batch by a nonadaptive adversary, whereas each query generated can depend on the outcome of all previously-generated queries in the case of an adaptive adversary.

6.1 Hamlin et al.

Hamlin et al. consider the setting where corpus owners store their encrypted documents on a remote server and can selectively share their documents with each other. The adversary is assumed to be the Server colluding with a coalition of malicious corpus owners C. Notable design choices by Hamlin et al. are:

- There is one security definition that covers both document content privacy and search query privacy. Document content privacy is covered by the adversary providing two choices in corpora sets and then trying to guess which choice the challenger made. Search query privacy is covered by providing the adversary with the appropiate queries/trapdoors. Curtmola et al. had already shown that these two security properties should not be separated since knowing the contents of the documents leaks information about the contents of the query and vice versa. Despite this, previous definitions such as [21] potentially still defined the two separately.
- There is no oracle access for the adversary. This is similar to the design choice by [6]. This lack of access is does not weaken the security guarantee of [6] and neither does it here. Since the definition of SSE is with respect to document collections as opposed to individual documents, it is sufficient for their security. The idea is this: instead of accessing the oracle a polynomial

number of times, the queries to the oracle are instead included in both choices given to the challenger.

- The security against cross user leakage is tested by giving the adversary access to all information stored by a coalition of corrupted corpus owners. As explained in section 2, is this a very strong adversarial model.
- The explicit consideration of data owners that can share documents with honest users. It was shown by [12] that the proposed definition of [21] – who explicitly prevented the adversary from sharing documents with honest users – allowed the security of a user to be compromised when a malicious corpus owner shares documents with him. The problem arose from the share key not depending on the shared document, which allowed any query under the honest user's key to be transformed into a query under the key of the malicious corpus owner. To address this, [13] make the share keys dependent on the document key of the document shared (cf. section 3.1). To guarantee security they allow the coalition of corrupt corpus owners to share documents with users.
- In the security game all documents that are shared, are assumed to be accepted by the recipient. Thus the security game assumes that documents that would be rejected by a user due to leaking too much information (as explained in section 2), are not shared.
- Users are required to make distinct Search queries. Thus, if a real-world user makes non-unique queries then this is leaked to the adversary.
- The definition by [13] is selective and indistinguishability-based and not simulation-based.
- Corpus owners do not have access to the user keys. As such, when a document is shared the user needs to generate and upload the share key. The contents of the share key are thus unknown to the (malicious) corpus owner.

Definition 7 (Secure SSE (Hamlin et al.)). *A SSE scheme is secure if every PPT Adversary $\mathcal{A}$ has only $negl(\lambda)$ advantage in the following security game with a challenger $\mathcal{C}$, given dictionary Δ and security parameter λ:*

1. *$\mathcal{A}$ sends to $\mathcal{C}$:*
   (a) *A set of users U, a set of corpus owners O, and a subset of of corrupted data owners O_C. We use $O_H = O \backslash O_C$ for the honest corpus owners.*
   (b) *For every corrupted corpus owner $c \in O_C$ his document key K_d^c.*
   (c) *For every corpus owner $\mathbf{o} \in O$ two sets of keywords $W_{\mathbf{o}}^0$ and $W_{\mathbf{o}}^1$ are chosen from the corpus space 2^{Δ} to test indistinguishability, with the restriction for $c \in O_C$ that $W_c^0 = W_c^1$ and for $h \in O_H$ that $|W_h^1| = |W_h^1|$.*
   (d) *A bipartite share graph $G = (U, O, E)$ with edges defining the access relationships between the corpus owners (each having one document only) and the users.*
   (e) *For every user $u \in U$ two sequences of keywords $(w_1^0, \ldots, w_n^0)_u$ and $(w_1^1, \ldots, w_n^1)_u$ for some $n \in \mathbb{N}_{>0}$. Given u and the corresponding two sequences, if u has been granted access – i.e. edge $(u, \mathbf{o}) \in E$ – then a search hit in one set results in a search hit in the other set – i.e. $(w_i^0 \in W_{\mathbf{o}}^0) \iff (w_i^1 \in W_{\mathbf{o}}^1)$.*

2. *$\mathcal{C}$ performs the following:*

   (a) *Chooses a random bit $b \in \{0, 1\}$.*

   (b) *For each user $u \in U$ he generates $K_u \leftarrow UserKeyGen(1^\lambda)$.*

   (c) *For each $\odot \in O$ he generates $K_d^{\odot} \leftarrow DocumentKeyGen(1^\lambda)$.*

   (d) *For each corpus owner $\odot \in O$ he uploads the corpus $xSet_{\odot} \leftarrow Setup_{K_d^{\odot}}(D_{\odot})$*

   (e) *For each encrypted document $xSet_{\odot}$ with $\odot \in O$ he generates a share key $\sigma_{u,\odot} \leftarrow Share_{K_u}(K_d^{\odot}, xSet_{\odot})$.*

   (f) *For each user u and keyword $w \in (w_1^b, \ldots, w_{n_u}^b)_u$ he generates a query $q \leftarrow Query_{K_u}(w)$ and adds it to the initially empty list $qSet$.*

   (g) *Sends $\{xSet\}_O, \{\sigma_{u,\odot}\}_{(u,\odot)\in E}$, and $qSet$ to $\mathcal{A}$.*

3. *$\mathcal{A}$ outputs guess b', and its advantage is $Adv_{\mathcal{A}}(1^\lambda) = p(b = b')$.*

Here we have combined [13]'s protocols Share and ProcessSet into one. The share keys are assumed to be generated by the users themselves, therefore the corpus owner sends his data key K_d to the querier via a secure channel. Searching is performed by searching through $\{xSet\}_O$.

6.2 Patel et al.

Patel et al. consider the setting where one corpus owners store his encrypted documents on a remote server and can selectively share his documents with users. The Adversary is assumed to be an honest-but-curious Server that corrupts a subset of the users C. Notable design choices by Patel et al. are:

- The absence of unsharing. "The ability to revoke access is crucial" according to [20], but at the same time they "assume no revocation (unsharing) is made [in their security definition] as a curious Server may keep all authorisation tokens provided".

- All share queries occur before any Search query takes place since "a curious server can always postpone or duplicate the execution of a query" [20].

- Contrary to the other security definitions, they do not use an adversary but rather define security in terms of computational indistinguishability. This computational indistinguishability must hold for "instance", where the instance is what the adversary offers to the challenger in the security games by the other authors. Two probability distributions are considered computationally indistinguishable when there is no efficient algorithm that can distinguish between them. As such, if we provide the output of the real world (i.e. the "server view sView") and the ideal worl to a PPT adversary – where PPT is the efficiency requirement and the adversary is an algorithm – then he cannot distinguish[16] between the two if and only if the probability ensembles are computationally indistinguishable. Proving this formally is outside the scope of this paper, but the idea is that instead of requiring computational indistinguishability for any given instance, we can allow the adversary to determine the instance and then make a guess given the output of the experiment.

- The queries are defined in their entirety before any of the queries are executed. As a result, the security guarantee is for a selective adversary, not an adaptive adversary.

[16] I.e., with a significant advantage.

- The cross-user leakage of the implementation by [20] is considerable: If an honest and a corrupted user share a document, the queried keywords belonging to this document can be leaked to the server [20]. Such leakage is allowed by their security definition as it makes use of a leakage function $\mathcal{L}$.

Definition 8 (Instance)**.** *An instance of SSE is defined as $\mathcal{I} = (D, Access, (u, w)_{i\in[n]})$ consisting of a corpus D, collection of subsets of document identifiers $Access$, and n tuples of user and keyword combinations $(u, w)_{i\in[n]}$. The role of $Access$ is to define the document-access rights. For each $id \in Access_u$ a Share query needs to be executed. Thus $Access$ servers the same purpose as the share graph in [13].*

Patel's et al.'s definition explicitly the corpus and the documents in the corpus as part of an instance. We have removed this double-sidedness since a corpus consists of documents: $D = \{id(d), W(d), meta_d\}_{d\in D}$, where the metadata is unique to Patel et al (cf. section 2). For the same reason Patel et al.'s security definition found below does not explicitly contain the set of documents. We assume a dictionary Δ and PPT Adversary. As discussed in section 5.1, we cover the Enroll algorithm by the Setup protocol.

Definition 9 (Secure SSE (Patel et al.))**.** *Let the view of Server corrupting users C for instance $\mathcal{I}$ be the output of the following experiment:*

sView$^{U,C}(\lambda, \mathcal{I})$:

1. $K \leftarrow OwnerKeyGen(1^\lambda)$;
2. $(K_U, xSet) \leftarrow Setup_K(D, U)$;
3. $(uSet, \{\{K_d\}_{d\in Access_u}\}_{u\in U}) \leftarrow Share_K(\{(K_u, Access_u)\}_{u\in U})$;
4. *For each* $i \in [n]$:

   $q_i \leftarrow Query_{K_{u\,i}}(w_i, \{(d, K_d^1, K_d^3)\}_{d\in Access_{u_i}})$;

   $Result_i \leftarrow Search(q_i)$;
5. *Output* $(K_C, xSet, uSet, (q_i, Result_i)_{i\in[n]})$;

A searchable symmetric encryption scheme is secure with respect to leakage $\mathcal{L}$ if there exists an efficient simulator $\mathcal{S}$ such that for every coalition C and every instance $\mathcal{I}$

$$\{sView^{U,C}(1^\lambda, \mathcal{I})\} \approx_c \{\mathcal{S}(1^\lambda, \mathcal{L}(\mathcal{I}, C))\}$$

Here the sets are taken over C and $\mathcal{I}$, and $\approx_c$ means that the two distributions are computationally indistinguishable.

It should also be noted that Patel et al. consider each user to be enrolled before any of the access granting and querying has taken place because "a curious server can always postpone or duplicate the execution of a query" [20]. Since the set of users does not change after initial enrollment, we enroll all users during setup.

6.3 Wang et al.

Wang et al. consider the setting where one corpus owners store his encrypted documents on a remote server and can selectively share his documents with users. The Adversary is assumed to be an honest-but-curious Server that corrupts a subset of the users C. Notable design choices by Wang et al. are:

- Whereas the previously mentioned definitions assumed all Share queries to happen before any Search query happens, [33] allow Search queries and Share queries to intermingle. This is because the schemes introduced by them can be "share forward-private" (cf. section 6.3.1). If the scheme is share forward-private then the order of the queries matters security-wise. If the scheme is not share forward-private then we note that the definition requires that the adversary's advantage must be negligible "for *any* PPT adversary". In particular this must hold for an adversary that does not intermingle Search and Share queries but performs all Share queries first.

- Whereas the previous definitions were for selective adversaries, here the adversary queries adaptively. This is represented by the adversary choosing the next query to execute after seeing the result of his previous choice.

6.3.1 Share forward-private

Wang et al. introduce the concept of share forward-privacy. The idea behind share forward-privacy is that the server cannot relate a Share operation to previous operations at the time it takes place. In other words, the Server should not be able to tell whether the document being shared contains keywords that have been searched before. The formal definition below makes use of the following auxiliary function:

$$Crp_C^{Shr}(d,u)) = \begin{cases} W(d) & \text{if } u \in C \\ |W(d)| & \text{otherwise} \end{cases}$$

Definition 10 (Share forward-private)**.** *A multi-user searchable symmetric encryption scheme with compromised users C is share forward-private if and only if the Share leakage function $\mathcal{L}^{Shr}$ can be written as*

$$\mathcal{L}^{Shr}(K, d, u, \iota_{\Theta,u}) = \mathcal{L}'(id, u, Crp_C^{Shr}(d,u)),$$

where ι_Θ is the locally stored/private information of the corpus owner and user u.

The definition above prevents the leakage of the owner key K and does not leak any of the private information $\iota_{\Theta,u}$. Concerning the document it only leaks the number of keywords if the user is not corrupted. If the user is corrupted, it leaks all keywords to the Server.

We will now examine the share forward-privacy of the three implementations by [33] (cf. section 3.3).

- NFNU: is not share forward-private since the user generates query addresses until he is told to stop by the Server. If user u searches for keyword w with a_w results then he queries $(w,1),\ldots,(w,a_w+1)$ assuming an honest (!) server. The next time that a document containing w is shared with u then the address corresponding to (w,a_w+1) is set and the Server learns a keyword of the document even if w is never searched for anymore.

- FU: the counters are stored privately by the user. As such, the user limits his queries to the set addresses, never accessing unset/uninitialised addresses. Thus FU is share forward-private.
- FNU: the counters are now stored at the server in an oblivious map. The oblivious map hides the access patterns related to the keyword counters thus achieving the same leakage as FU. FNU is therefore also share forward-private.

6.3.2 Wang et al.'s definition for security

A query q generated by the Adversary has properties $\{type, user, document\}$ in case the type is Share, and properties $\{type, user, keyword\}$ in case the type is Search. In the following definition the differences between the real and the ideal security game are coloured in red, i.e. all values that depend on secret information or the simulation thereof.

Definition 11 (Secure SSE (Wang et al.)). *Given security parameter λ, dictionary Δ and number of queries n, consider the following security games.*

$b \leftarrow Real_{\mathcal{A}}^{U,C,\Sigma}$:

1. $(D, U, C) \leftarrow \mathcal{A}(1^\lambda)$;
2. $(\iota_0 = K) \leftarrow OwnerKeyGen(1^\lambda)$;
3. $((\iota_0 = \iota_0 \cup aux), aux_C; DB_0 = \{xSet, Access_U\}) \leftarrow Setup_K(D, U)$;
4. ***for*** $i \in [n]$:

   $q_i \leftarrow \mathcal{A}(1^\lambda, DB_0, aux_C, \tau_{i-1})$; *The Adversary chooses the query adaptively*

   if $q_i.type$ ***is*** *Share* ***then***:

   $(\iota_i = aux_{q_i.u}; t_i, DB_i) \leftarrow Share_K(q_i.d, q_i.u, \iota_{i-1}; DB_{i-1})$;

   if $q_i.type$ ***is*** *Search* ***then***:

   $(\iota_i = aux_{q_i.u}, D_{q_i.u}(q_i.w); t_i, DB_i) \leftarrow Search_{K_{q_i.u}}(q_i.w, aux_{q_i.u}, \iota_{i-1}; q_i.trace, DB_{i-1})$;
5. ***return*** $b \leftarrow \mathcal{A}(1^\lambda, DB_0, aux_C, \tau_n)$;

Here $\tau_i = (t_0, \ldots, t_i)$ *with* t_k *being the messages from the user/owner to Server in the* k*'th query.*

$b \leftarrow Ideal_{\mathcal{A}}^{U,C,\Sigma}$:

1. $(D, U, C) \leftarrow \mathcal{A}(1^\lambda)$;
2. $st_{\mathcal{S}} \leftarrow SimOwnerKeyGen(1^\lambda)$;
3. $(st_{\mathcal{S}}, aux_C; DB_0) \leftarrow SimSetup_{st_{\mathcal{S}}}(D, U)$;
4. ***for*** $i \in [n]$:

   $q_i \leftarrow \mathcal{A}(1^\lambda, DB_0, aux_C, \tau_{i-1})$; *The Adversary chooses the query adaptively*

   if $q_i.type$ ***is*** *Share* ***then***:

$(st_{\mathcal{S}}; t_i, DB_i) \leftarrow SimShare_{st_{\mathcal{S}}}(\mathcal{L}^{Shr}(q_i.d, q_i.u); DB_{i-1})$;

if $q_i.type$ ***is*** *Search* ***then:***

$(st_{\mathcal{S}}; t_i, DB_i) \leftarrow SimSearch_{st_{\mathcal{S}}}(\mathcal{L}^{Srch}(q_i.w, q_i.u); DB_{i-1})$;

5. ***return*** $b \leftarrow \mathcal{A}(1^\lambda, DB_0, aux_C, \tau_n)$;

A SSE scheme is secure with respect to leakage function $\mathcal{L}$ if and only if for any PPT Adversary $\mathcal{A}$ that issues a polynomial number of search queries q there exists a stateful PPT simulator $\mathcal{S}$ such that the Adversary has only negligible advantage in distinguishing between the real and the ideal security game:

$$|p(Real_{\mathcal{A}}^{U,C,\Sigma}(\lambda, q) = 1) - p(Ideal_{\mathcal{A},\mathcal{S},\mathcal{L}}^{U,C,\Sigma}(\lambda, q) = 1)| \leq negl(\lambda).$$

Explicit $\mathbf{aux_C}$ output: Note that the corpus owner's auxiliary information aux includes the tuples (u, aux_u) for every $u \in U$, where aux_u includes K_u and Cnt_u. If $Cnt_u \notin aux_u$ then $aux = \{(u, aux_u, Cnt_u)\}_{u \in U}$. Thus the corrupted users' auxiliary informaton aux_C is part of aux and includes their user keys. Nevertheless, we have explicitly output the algorithms aux_C since this information is accessible to the Adversary.

Other choices: Furthermore, the document keys have been removed from the definition since these were only present for backwards compatibility, and the owner key input has explicitly been required by Share. Lastly, we haved added messages from the user/owner to the server in the real game.

6.4 Chamani et al.

The Adversary by Chamani et al. is assumed to be the Server that corrupts a subset of the users C. Notable design choices by Chamani et al. are:

- The adversary queries adaptively.
- The security game is made for dynamic SSE scheme (albeit that their implementations is of fixed-size (cf. section 5.3)). This means they require secure Upload, Update and Unshare protocols.
- They define forward security, backward security, and verifiability. Despite being notable, this is not within the scope of our work as it does not require unification. Information regarding these properties can be found in the appendix.
- Enrollment can still be assumed to happen in the Setup phase since the forward and backward privacy definitions do not concern themselves with the Enrollment leakage but with Share and Search leakage.

A query q generated by the Adversary has properties $\{Share, user, document\}$ if $type = Share$; properties $\{Search, user, keyword\}$ if $type = Search$; properties $\{Upload, document\}$ if $type = Upload$; properties $\{Update, document, WList, mode\}$ if $type = Update$; and properties $\{Unshare, user, document\}$ if $type = Unshare$. In the following definition the differences between the real and the ideal security game are coloured in red to aid in readability. Note that the differences are all depend on the secret information or the simulation thereof. As discussed in section 5.1, we cover the Enroll algorithm by the Setup protocol.

Definition 12 (Secure DMUSSE (Chamani et al.)). *Given security parameter λ, dictionary Δ and number of queries n, consider the following security games.*

$b \leftarrow Real_{\mathcal{A}}^{U,C,\Sigma}$:

1. $(D, U, C) \leftarrow \mathcal{A}(1^\lambda)$;
2. $(\iota_0 = K) \leftarrow OwnerKeyGen(1^\lambda)$;
3. $((\iota_0 = \iota_0 \cup aux), aux_C; DB_0 = \{xSet, uSet\}) \leftarrow Setup_K(D, U, N, |W|)$;
4. ***for*** $i \in [n]$:

    $q_i \leftarrow \mathcal{A}(1^\lambda, DB_0, aux_C, \tau_{i-1})$; \\The Adversary chooses the query adaptively

    if $q_i.type$ ***is*** *Share* ***then***:

    $(\iota_i = aux_{q_i.u}; t_i, DB_i) \leftarrow Share_K(q_i.d, q_i.u, \iota_{i-1}; DB_{i-1})$;

    if $q_i.type$ ***is*** *Search* ***then***:

    $(\iota_i = aux_{q_i.u}, D_{q_i.u}(q_i.w); t_i, DB_i) \leftarrow Search_{K_{q_i.u}}(q_i.w, aux_{q_i.u}, \iota_{i-1}; DB_{i-1})$;

    if $q_i.type$ ***is*** *Upload* ***then***:

    $(t_i, DB_i) \leftarrow Upload_K(q_i.d, \iota_{i-1}; DB_{i-1})$;

    if $q_i.type$ ***is*** *Update* ***then***:

    $(\iota_i = aux_U; t_i, DB_i) \leftarrow Update_K(q_i.d, q_i.WList, q_i.mode, aux_U, \iota_{i-1}; DB_{i-1})$;

    if $q_i.type$ ***is*** *Unshare* ***then***:

    $(\iota_i = aux_{q_i.u}; t_i, DB_i) \leftarrow Unshare_K(q_i.d, q_i.u, \iota_{i-1}; DB_{i-1})$;
5. ***return*** $b \leftarrow \mathcal{A}(1^\lambda, DB_0, aux_C, \tau_n)$;

Here $\tau_i = (t_0, \ldots, t_i)$ with t_k being the messages from the user/owner to Server in the k'th query.

$b \leftarrow Ideal_{\mathcal{A}}^{U,C,\Sigma}$:

1. $(D, U, C) \leftarrow \mathcal{A}(1^\lambda)$;
2. $st_{\mathcal{S}} \leftarrow SimOwnerKeyGen(1^\lambda)$;
3. $(st_{\mathcal{S}}, aux_C; DB_0) \leftarrow SimSetup_{st_{\mathcal{S}}}(D, U, N, |W|)$;
4. ***for*** $i \in [n]$:

    $q_i \leftarrow \mathcal{A}(1^\lambda, DB_0, aux_C, \tau_{i-1})$;

    if $q_i.type$ ***is*** *Share* ***then***:

    $(st_{\mathcal{S}}; t_i, DB_i) \leftarrow SimShare_{st_{\mathcal{S}}}(\mathcal{L}^{Shr}(q_i.d, q_i.u); DB_{i-1})$;

    if $q_i.type$ ***is*** *Search* ***then***:

$(st_{\mathcal{S}}; t_i, DB_i) \leftarrow SimSearch_{st_{\mathcal{S}}}(\mathcal{L}^{Srch}(q_i.w, q_i.u); DB_{i-1})$;

if $q_i.type$ ***is*** *Upload* ***then:***

$(t_i, DB_i) \leftarrow SimUpload_{st_{\mathcal{S}}}(\mathcal{L}^{Upl}(q_i.d); DB_{i-1})$;

if $q_i.type$ ***is*** *Update* ***then:***

$(\iota_i = aux_U; t_i, DB_i) \leftarrow SimUpdate_{st_{\mathcal{S}}}(\mathcal{L}^{Upd}(q_i.d, q_i.WList, q_i.mode); DB_{i-1})$;

if $q_i.type$ ***is*** *Unshare* ***then:***

$(\iota_i = aux_{q_i.u}; t_i, DB_i) \leftarrow SimUnshare_{st_{\mathcal{S}}}(\mathcal{L}^{Unsh}(q_i.d, q_i.u); DB_{i-1})$;

5. ***return*** $b \leftarrow \mathcal{A}(1^\lambda, DB_0, aux_C, \tau_n)$;

A dynamic multi-user SSE (DMUSSE) scheme is secure with respect to leakage function $\mathcal{L}$ if and only if for any PPT Adversary $\mathcal{A}$ that issues a polynomial number of search queries q there exists a stateful PPT simulator $\mathcal{S}$ such that the Adversary has only negligible advantage in distinguishing between the real and the ideal security game:

$$|p(Real^{U,C,\Sigma}_{\mathcal{A}}(\lambda, q) = 1) - p(Ideal^{U,C,\Sigma}_{\mathcal{A},\mathcal{S},\mathcal{L}}(\lambda, q) = 1)| \leq negl(\lambda).$$

In our definition we have added the simulator state $\mathcal{S}$ as input for SimSetup and to SimEnroll.

7 Defining security (MKSE)

In section 6 the definition by Hamlin et al. is the only one that does not provide semantic security but indistinguishability. As discussed in section 6, Curtmola et al. have shown that in the multi-user setting semantic security implies indistinguishability for an adaptive adversary, and that semantic security is equivalent with indistinguishability for a non-adaptive adversary in the multi-user setting. In section 1 we have explained how the multi-key setting is much more complex than the multi-user setting. We have also explained how this leads to the natural question if the indistinguishability-based definition by Hamlin et al. is equivalent with the semantic security definition of Curtmola et al. adapted to the multi-key setting (assuming a non-adaptive adversary).

As it turns out, this equivalence holds for the MKSE-adapted definitions just like it did in MUSE. In this section we will introduce the notion of non-adaptive (NA) semantic security for MKSE and prove its equivalence with NA-indistinguishability.

- First, we will redefine Curtmola et al.'s "auxiliary notions" for MUSE such as the history and search pattern and transform them in auxiliary notions for MKSE.
- Then, using these definitions, we will define non-adaptive multi-key (MK) indistinguishability and NA-MK-semantic security. NA-MK-indistinguishability is the formal definition of Hamlin et al.'s security requirement and NA-MK-semantic security is the introduction of a definition for semantic security in the non-adaptive multi-key setting.
- Lastly, we will prove the equivalence between the two security definitions. Note that every multi-user definition in this section was defined by Curtmola et al. [6] – who did not implement sharing on a per-document basis – and is listed here for reference.

We will use the same definition for a dictionary as Curtmola et al.:

Definition 13 (Dictionary)**.** *Let $\Delta = (w_1, \dots, w_d)$ be a dictionary of d words in lexicographic order, and 2^Δ be the set of all possible documents with words in Δ. We assume $d = poly(\lambda)$ and that all words $w \in \Delta$ are of length polynomial in λ.*

Throughout this section we will assume that that D_O is of polynomial length in λ, each $D_{\mathbf{o}}$ consisting of a polynomial number of keywords $poly(\lambda)$.

7.1 Redefining multi-user auxiliary notions

Here we will introduce four auxiliary notions used in our MK-definitions.

Multi-key history: In MKSE, the interaction between a set of clients and the server is described by a set of corpus owners/corpora, a share graph which is a graph indicating access rights, and a sequence of queries. An instance of such an interaction is a multi-key history. Informally, a history is everything that is necessary to construct a (current) state of the database and search queries.

Share graph in [13]: A share graph is used to indicate which corpus owners granted which users access to their corpora; it is a graph indicated access rights. Note that in Hamlin et al. [13] every Share query is assumed to happen before the first Search query (cf. section 5.2). Because of this, we assume that the share graph is provided at the beginning of the security games and is fixed. Now, since the Share graph defines the access relations between users and corpus owners, i.e. the Share queries, the n-query history[17] can be assumed to consist solely of a set of n Search queries $qSet \in H$. This assumption can be made since the share graph is fixed at the beginning of the game – meaning that there are no Share queries performed after the setting has been established – and since the share graph is of polynomial size just like $qSet$, meaning that there are "no size problems". Thus, we make the assumption that for Hamlin et al. the set of all queries $qSet$ consists solely of Search queries, simplifying the definitions. Search queries are issued by a user u and are for a keyword w. Thus every query $q \in qSet$ can be represented by the tuple (w, u).

We will now provide the definition for Curtmola et al.'s MU-history, discuss why it cannot be used in our MKSE setting, and then construct a definition for MKSE that addresses these limitations.

Definition 14 (Multi-user history)**.** *Let Δ be a dictionary and $D \subseteq 2^\Delta$ be a corpus over Δ. A n-query multi-user history over D is a tuple $H = (D, qSet)$ that includes the corpus D and a tuple of n keywords $qSet = (w_1, \dots, w_n)$.*

The issues with the definition above are:

1. it is with respect to one corpus (owner).
2. each user is assumed to have access to every document in the corpus.
3. each user's search query for keyword w is the same and thus the query can adequately be described by the keyword w.

[17] We have chosen a "n-query" history since it sounds like the noun "enquiry", which means to ask for information.

We will address these issues in our multi-key definition. We will make use of an auxiliary notion: the share graph. The share graph defines access relation ships between corpus owners and users and will thus help to address the first two issues.

Definition 15 (Share graph)**.** *Let O be a set of corpus owners, U be a set of users, and E be a set of edges $(\odot, u)$ between owners $\odot \in O$ and users $u \in U$; then the (bipartite) graph $G = (O, U, E)$ is a share graph where the edges indicate that u has been granted access to the corpus of $\odot$.*

Our share graph contains a set of corpus owners, addressing the first issue, and shares the corpora (of size "1 document") on a per document basis, addressing issue 2. We will now define the multi-key history, addressing all three issues.

Definition 16 (Multi-key history)**.** *Let $G = (O, U, E)$ be a share graph, Δ be a dictionary and D_O a set of corpora. Each corpus owner $\odot \in O$ has a corpus $D_{\odot} \subseteq 2^{\Delta}$. An n-query multi-key history over D_O is a tuple $H = (G, D_O, qSet)$ that includes a share graph G, corpora D_O, and a tuple of n queries $qSet = ((w,u)_1, \ \ldots, \ (w,u)_n)$ where $w \in \Delta$ and $u \in U$.*

The MK-history clearly differs from the MU-history in that users can have differing access rights. This is defined by the share graph, which allows each real-world corpus owner to share on a per document basis (3.1.2). Furthermore, if two queries are for the same keyword – say the first and second query – but for another user then we now have $q_1 = (w_1, u_1) \neq (w_1, u_2) = q_2$. Henceforth there are two types of histories: multi-key and multi-user. If the context does not make clear which history is meant we will always use 'MK-history' or 'MU-history'; if the context however is clear then we can simply use 'history'.

Now, the multi-user search pattern is defined as follows:

Definition 17 (Multi-user search pattern)**.** *Let Δ be a dictionary and $D \subseteq 2^{\Delta}$ a corpus over Δ. The search pattern induced by an n-query multi-user history $H = (D, qSet)$ is a symmetric binary matrix $\sigma(H)$ such that for $1 \leq i, j \leq n$ the element in the ith row and j^{th} column is 1 if $w_i = w_j$ and 0 otherwise.*

The issues with the definition above are that our history:

1. is with respect to multiple corpus owners/corpora instead of one corpus owner/corpus.
2. defines access relations between users and corpora instead of granting all users complete access.
3. distinguishes between queries (w_i, u_j) and (w_k, u_l) if $i \neq k$ or $j \neq l$, not only when $i \neq j$.

We will now address these issues in our MK-search pattern definition.

Definition 18 (Multi-key search pattern)**.** *Let Δ be a dictionary, O a set of corpus owners, U a set of users and $D_O \subseteq 2^{\Delta}$ a set of corpora, each over Δ. The search pattern induced by an n-query multi-key history $H = (G, D_O, qSet)$ is a symmetric binary matrix $\sigma(H)$ such that for $1 \leq i, j \leq n$ the element in the ith row and j^{th} column is 1 if $(w,u)_i = (w,u)_j$ and 0 otherwise.*

Our MK-definition for security will take cross-user leakage into account. When defining security with cross-user leakage, an adversary must not be able to tell if two users query for the same keyword. If an adversary is given the search pattern as defined by [6], he is trivially able to do this since the search pattern indicates equivalence between keywords. To remove this leakage, we have tied the user to his query.

Now, the MU-access pattern is defined as follows:

Definition 19 (Multi-user access pattern)**.** *Let Δ be a dictionary and $D \subseteq 2^{\Delta}$ a corpus over Δ. The access pattern induced by a n-query multi-user history $H = (D, qSet)$ is the tuple $\alpha(H) = (D(w_1), \ldots, D(w_q))$.*

The issues with the definition above are that our history:

1. is with respect to multiple corpus owners/corpora instead of one corpus owner/corpus.
2. defines access relations between users and corpora instead of granting all users complete access. This means that a document containing the queried keyword is only returned if the user has been granted access to it. This makes the query result not only depend on the documents, but also on the querying user. Thus, a document $D_{\odot}$ is only returned to u upon querying if $(u, \odot) \in E$ with $E \in G$ and $G \in H$.

We will now address these issues in our MK-search pattern definition.

Definition 20 (Multi-key access pattern)**.** *Let Δ be a dictionary, O a set of corpus owners and $D_{\odot} \in 2^{\Delta}$ a corpus belonging to corpus owner $\odot$. The multi-key access pattern induced by an n-query multi-key history $H = (G, D_O, qSet)$ is the tuple*

$$
\begin{aligned}
\alpha(H) = \Big(& \big\{ (D_{\odot}(w_1) \big| (u_1, \odot) \in E) \;\Big|\; \odot \in O \big\}, \\
& \big\{ (D_{\odot}(w_2) \big| (u_2, \odot) \in E) \;\Big|\; \odot \in O \big\}, \\
& \ldots, \\
& \big\{ (D_{\odot}(w_n) \big| (u_n, \odot) \in E) \;\Big|\; \odot \in O \big\} \Big)
\end{aligned}
$$

The tuple $\alpha(H)$ consists of n entries. The access pattern entry i induced by query q_i are all documents that u has access to and that contain the queried keyword w_i. Usually we use D_u to denote the set of documents that u has access to and use $D_u(w_i)$ to restrict this set to the documents containing keyword w_i, but to stress how the MK-access pattern is dependent on the share graph we have given the set as defined by its elements. Note that we will use $D_u(w_i) \in \alpha(H)$ later this section. Note that the MK-access pattern clearly differs from the MU-access pattern in its user dependency: each user has its own access rights to different corpora.

We have now seen how the history, access pattern, and search pattern differ in MKSE from MUSE. The trace is dependent on all of these auxiliary notions and as such will need to be redefined as well.

Definition 21 (Multi-user trace)**.** *Let Δ be a dictionary and $D \subseteq 2^{\Delta}$ be a corpus over Δ. The trace induced by a n-query multi-user history $H = (D, qSet)$ is a sequence $\tau(H) = (|D_1|, \ldots, |D_n|, \alpha(H), \sigma(H))$*

compromised of the lengths of the documents in D and the access and search patterns induced by H.

The issues with the definition above are inherited from all previous auxiliary functions. The trace is induced by a n-query MU-history, inheriting all its issues, and is a sequence containing the MU-access pattern and MU-search pattern, inheriting all their issues. The trace is what is considered the acceptable minimum leakage by Curtmola et al. [6] and the issues that the MU-trace has can largely be addressed by replacing the MU-auxiliary notions by their MK-auxiliary counterparts. There is however one thing that is then not accounted for: the share graph. As seen in section 6.1, the share graph is determined by the adversary and, as such, is known to the adversary. Therefore our trace also contains the share graph.

Definition 22 (Multi-key trace)**.** *Let Δ be a dictionary and $D_O \subseteq 2^{\Delta}$ a collection of m corpora. The multi-key trace induced by a n-query multi-key history $H = (G, D_O, qSet)$ is a sequence*

$$\tau(H) = (G, |D_1|, \ldots, |D_m|, \alpha(H), \sigma(H))$$

comprised of the share graph, the lengths of the documents, the MK-access and MK-query patterns induced by H, and the set of share keys.

Now, the definitions above are somewhat specific to the situation as described by Hamlin et al [13]. There, the length of all corpora is 1 (document) and thus for m corpus owners we end up with m documents of length $|W(d_i)|$ for $i \in [m]$ .Let the reader be aware that the same notation is used for the MK-access and MK-query pattern as for the MU-variants.

Finally, Curtmola et al. [6] require throughout their work that the history is non-singular as defined below:

Definition 23 (Non-singular history (MUSE))**.** *A MU-history $H = (D, qSet)$ is non-singular if (1) there exists at least one MK-history $H' = (D', qSet')$ such that $\tau(H') = \tau(H)$; and if (2) such a history can be found in polynomial-time given $\tau(H)$.*

The issue with this definition is that it does not consider the share graph. In the MKSE the share graph influences (as explained above) the search pattern and access pattern; it influences the search results a user gets. In indistinguishability-based definitions are meant to test the security of the documents/corpora and the queries. Note that this remains the goal in the multi-key setting. As such, the share graph needs to be considered but not changed: the share graph is considered constant. In our work we will assume that the dictionary Δ and the trace are such that all histories H over Δ are non-singular as defined below:

Definition 24 (Non-singular history (MKSE))**.** *A MK-history $H = (G, D_O, qSet)$ is non-singular if (1) there exists at least one MK-history $H' = (G, D'_O, qSet')$ such that $\tau(H') = \tau(H)$; and if (2) such a history can be found in polynomial-time given $\tau(H)$.*

We stress that even though the definitions for non-singular MK-history and non-singular MU-history seem similar, they are very different indeed. Not only are the histories defined significantly different upon closer examination (see the issues with the MU-history above), also the MK-trace and MU-trace are very different to each other (see the issues with the MU-trace definition above).

Now, note that in MKSE the share graph is assumed to be constant (cf. section 6.1). This does not differ from Curtmola et al., where the setting is fixed as well: one corpus owner shares his corpus with a set of users. Note then that in both settings the security setting is fixed and that in both settings the non-singular-history requirement concerns itself solely with the documents and searched keywords. This is however where Hamlin et al. note that this assumption for the non-singular MK-history is a stronger assumption than it is in MUSE. Since the share graph, access pattern and search pattern are assumed fixed in the trace, the simulator in an ideal world needs to find an assignment for the corpora (of the honest corpus owners) to the honest corpus owners. This needs to be done in a way that – given the share graph – the access and search patterns remain fixed as well as the sizes of the corpora/documents. The challenge here is to find one configuration that works for all of the given queries satisfying all requirements.

The problem boils down to the following:

1. We have a bipartite graph $G = (O \cup U, E)$ where O is the set of corpus owners, and U is the set of users. Both O and U are of polynomial size, as is $E \subseteq O \times U$ representing the relations between corpus owners and users.
2. Each owner should be allocated exactly one corpus, i.e. one document/set of keywords.
3. Given the requirement $\{(w, u, D_u(w)) : (w, u) \in \sigma(H) \text{ and } D_u(w) \in \alpha(H)\}$, we now have to find an allocation of documents to corpus owners such that this requirement is met as well as the given sizes of the sets.

The question that now remains is if this can be done in polynomial-time, since the problem is what is known as a constraint satisfaction problem.

It turns out however that there are special cases of MKSE where the non-singularity assumption is a lot weaker. For example, in the PRF-construction by Hamlin et al. (section 3.1) the Search algorithm does not search over the documents but instead over the share keys. The Search algorithm does not use the document at all and thus no issues of consistency across different queries arise. Searching over the share keys does not suffer from the same problem since the share keys are unique for each user/document combination. This is in contrast to the simulation of the documents explained above, which were not unique for each user and thus had to stay consistent across queriers. In the case of the PRF-construction the assumptions are equally strong:

- In Curtmola et al. there are a polynomial number of documents. Each document is returned for a polynomial number of Search queries, independently of the other documents and users. It is independent of the other users since each user has full access to the corpus and constructs the same search query.
- In the PRF-construction there a polynomial number of share keys. This is because there are a polynomial number of documents and a polynomial of users.[18] Each of these share keys returns

[18] The multiplicative order of the documents and users is still polynomial.

its associated document independently of the other share keys/documents and only without considering the other users. It is independent of the other users since share keys only allow its owner to query on it (this is because they are constructed under the uploader's user key).

7.2 Indistinguishability and semantic security (MKSE)

The first security definition for MKSE is that of non-adaptive indistinguishability. Our definition is a formalised version of the security requirement by [13], definition 7. We will briefly explain how the definitions relate to one another. The first line of our definition covers the adversaries requests to the challenger (line 1, covering lines 1a-e of definition 7). The challenger then sets up the corpus by first generating the lacking document keys (l2, cov. l2c), randomly choosing the corpus to set up (l3, cov. l2a), and finishing its setup (l4, cov. l2d; slightly abusing notation by invoking "Setup" sequentially for each corpus owner and denoting the total output by $xSet^b$). Then our security game enters a phase that covers all requirements for corpora to be shared with users. First we generate all the user keys (l5, cov. l2b), and then share the corpora according to the share graph (l6-6, cov. l2e). We then enter the query phase (l8-9, cov. l2f) and give the adversary the leakage (l10, cov. l2g). We both conclude with the adversary making a guess. We note that we have covered the lines 1a-e, 2a-g, and the final guess by the adversary, indeed being all the lines of the game as proposed by definition 6.1.

Definition 25 (Non-adaptive indistinguishability (MKSE))**.** *Let* $\Sigma =$ (DataKeyGen, UserKeyGen, Setup, Query, Search, Share) *be an MKSE scheme,* $\lambda \in \mathbb{N}_{>0}$ *the security parameter,* $\mathcal{A}$ *a PPT adversary, and consider the following probabilistic experiment in the multi-key setting:*

$\mathsf{Ind}^{\Sigma}_{\mathcal{A}}(\lambda)$

1 : $(H^0, H^1, C \subseteq O, \{K_d\}_C, st_{\mathcal{A}}) \leftarrow \mathcal{A}(1^\lambda)$;
2 : $\{K_d\}_{O_H} \leftarrow \mathsf{DocumentKeyGen}(1^\lambda)$;
3 : $K_U \leftarrow \mathsf{UserKeyGen}(1^\lambda)$;
4 : $b \xleftarrow{\$} \{0,1\}$;
5 : $xSet^b \leftarrow \mathsf{Setup}((K_d, D^b)_{\odot \in O})$;
6 : $uSet = \{\}$;
7 : **foreach** $(\odot, u)$ **in** E :
$uSet \leftarrow uSet \cup \mathsf{Share}((K_d, xSet^b)_{\odot}, K_u, u)$;
8 : $qSet' = \{\}$;
9 : **foreach** (w, u) **in** $qSet$:
$qSet' \leftarrow qSet' \cup \mathsf{Query}_{K_u}(w)$;
10 : $b' \leftarrow \mathcal{A}(st_{\mathcal{A}}, (xSet^b, qSet', uSet))$;
11 : *if* $b' = b$, *output 1*
otherwise output 0

with the restriction that $\tau(H^0) = \tau(H^1)$*, and where* $st_{\mathcal{A}}$ *is a string that captures the adversary's state and* $xSet^b = \{xSet^b_{\odot} : \odot \in O\}$ *is the set of all encrypted corpora. We say that* SSE *is secure in the sense of non-adaptive indistinguishability if for all polynomial-size adversaries* $\mathcal{A}$

$$p\Big(\mathsf{Ind}_{SSE,\mathcal{A}}(\lambda) = 1\Big) \leq \frac{1}{2} + \mathrm{negl}(\lambda)$$

where the probability is taken over the choice of b and the coins of Gen *and* Enc*.*

Note that in the non-singularity definition for the MK-history we required that the two histories have the same share graph. Also, as explained in section 6.1, letting the adversary determine the keys of the coalition has advantages for the definition of the security game: it makes the proof in section 7.3 simpler and shorter. As such, our definition too allows the adversary to determine the keys of the coalition.

We note that our representation of definition 3.1 is more clear, and is according to the expectations that the readers of [6] might have. As such, it is also a good preparation for our proof. As stated before, it remains to prove that NA-MK-indistinguishability is equivalent to [6]'s semantic security adapted to MKSE for a non-adaptive adversary. We note that H refers to a MK-history (cf. definition 16) and that the subscript H appended to corpus owners and users is used to mark them as honest (as discussed in section 4.1. We also remind the reader that $O_C \cup O_\text{H} = O$.) Furthermore, in the case of Hamlin et al.'s implementation $uSet$ refers to the set of share keys $\sigma_{u,d}$. As discussed above, the share graph is determined by the adversary as part of the history.

Definition 26 (Non-adaptive semantic security (MKSE))**.** *Let* $\Sigma =$ (DataKeyGen, UserKeyGen, Setup, Query, Search, Share) *be an MKSE scheme,* $\lambda \in \mathbb{N}_{>0}$ *the security parameter,* $\mathcal{A}$ *a PPT adversary,* $\mathcal{S}$ *a simulator, and consider the following probabilistic experiments in the multi-key setting:*

$(\nu, st_\mathcal{A}) \leftarrow \text{Real}^\Sigma_\mathcal{A}(\lambda)$

1 : $(H, C \subseteq O, \{K_d\}_C, st_\mathcal{A}) \leftarrow \mathcal{A}(1^\lambda)$;
2 : $\{K_d\}_{O_H} \leftarrow \text{DocumentKeyGen}(1^\lambda)$;
3 : $K_U \leftarrow \text{UserKeyGen}(1^\lambda)$;
4 : $xSet \leftarrow \text{Setup}((K_d, D)_{\odot \in O})$;
5 : $uSet = \{\}$;
6 : **foreach** $(\odot, u)$ **in** E :
  $uSet \leftarrow uSet \cup \text{Share}((K_d, xSet)_\odot, K_u, u)$;
7 : $qSet' = \{\}$;
  foreach (w, u) **in** $\{(w, u)\}$:
8 : $qSet' \leftarrow qSet' \cup \text{Query}_{K_u}(w)$;
9 : *output* $\nu = (xSet, qSet', uSet)$ *and* $st_\mathcal{A}$

$(\nu, st_\mathcal{A}) \leftarrow \text{Sim}^\Sigma_{\mathcal{A},\mathcal{S}}(\lambda)$

1 : $(H, C \subseteq O, \{K_d\}_C, st_\mathcal{A}) \leftarrow \mathcal{A}(1^\lambda)$;
2 : $\nu \leftarrow \mathcal{S}(\tau(H))$;
3 : *output* $\nu = (xSet, qSet', uSet)$ *and* $st_\mathcal{A}$

We say that a given scheme Σ is semantically secure in the multi-key setting if for all polynomial-size adversaries $\mathcal{A}$ there exists a polynomial-size simulator $\mathcal{S}$ such that for all polynomial-size distinguishers $\mathcal{D}$

$$p\Big(\mathcal{D}(\nu, st_\mathcal{A}) = 1 : (\nu, st_\mathcal{A}) \leftarrow \text{Real}^\Sigma_\mathcal{A}(\lambda)\Big) - p\Big(\mathcal{D}(\nu, st_\mathcal{A}) = 1 : (\nu, st_\mathcal{A}) \leftarrow \text{Sim}^\Sigma_{\mathcal{A},\mathcal{S}}(\lambda)\Big) \leq negl(\lambda),$$

where the probabilities are over the coins of DocumentKeyGen, UserKeyGen, and Setup.

We would like to point out that our definition significantly differs from the one given by Curtmola et al. As explained in section 7 our formulation:

- is with respect to multiple corpus owners instead of one.
- takes access rights into account. Documents are shared with users on a per-document basis instead of the binary access by Curtmola et al.
- address cross-user leakage.

7.3 Proving equivalence

We will now prove that our MK-NA-indistinguishability definition and our MK-NA-semantic security definition are equivalent. We assume throughout this section thus that the adversary is non-adaptive (cf. section 6) and that Σ = (DocumentKeyGen, UserKeyGen, Setup, Search, Share) is a static SSE scheme. Note that for Hamlin et al. we split the Search protocol in a Query and a Search algorithm (cf. section 5.1).

Lemma 1.1. *MK-non-adaptive indistinguishability implies MK-non-adaptive semantic security.*

Proof. We will show using contraposition that if there exist a polynomial-size adversary $\mathcal{A}$ such that there exists a polynomial-size distinguisher $\mathcal{D}$ that succeeds in an $\text{Ind}^{\Sigma}_{\mathcal{A}}(\lambda)$ experiment with non-negligible advantage for all polynomial-size simulators $\mathcal{S}$, then there also exists a polynomial-size adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ that succeeds in an $\text{Ind}^{\Sigma}_{\mathcal{B}(\lambda)}$ experiment with non-negligible probability.

Let H and $st_{\mathcal{A}}$ be the output of $\mathcal{A}(1^\lambda)$. Since H is required to be non-singular there exists at least one history $H' \neq H$ with $\tau(H') = \tau(H)$ that can be found in polynomial-time. Consider a simulator $\mathcal{S}*$ that does the following:

- it generates one document key per corpus (owner): $K_D \leftarrow$ DocumentKeyGen(1^λ)
- it generates one user key per user: $K_U^* \leftarrow$ UserKeyGen(1^λ).
- it finds in polynomial-time $H' \neq H$ such that $\tau(H') = \tau(H)$, given $\tau(H)$.
- it builds a sequence of encrypted documents $xSet^*$, each under a different document key $K_d^* \in K_D^*$
- it builds a sequence of queries $qSet^*$ from H', each under a different user key $K_u^* \in K_U^*$
- it builds a sequence of share keys $uSet^*$, each under a different document key K_d^* and user key K_u^*.
- it outputs $\nu = (xSet^*, qSet^*, uSet^*)$ and $st^* = st_{\mathcal{A}}$

Let now $\mathcal{D}^*$ be the polynomial-size distinguisher that was assumed to succeed in the real-ideal experiment. $\mathcal{D}^*$ depends on $\mathcal{S}^*$. Without loss of generality we assume that $\mathcal{D}^*$ outputs 0 when given the experiment of $\text{Real}^{\Sigma}_{\mathcal{A}}(\lambda)$.

Let $\mathcal{B}_1$ be the adversary that computes $(H, st_{\mathcal{A}}) \leftarrow \mathcal{A}(1^\lambda)$, uses the trace $\tau(H)$ to find H', and returns $(H, H', st_{\mathcal{A}})$ as its output. Let $\mathcal{B}_2$ now be the adversary that is given $st_{\mathcal{A}}$ and $(xSet^b, qSet, uSet)$, and sets $\nu = (xSet^b, qSet, uSet)$ and outputs the bit b obtained by running $\mathcal{D}^*(\nu, st_{\mathcal{A}})$.

Since $\mathcal{A}$ is assumed to be of polynomial size, $\mathcal{B}$ is clearly of polynomial size. It thus remains to analyse $\mathcal{B}$'s success probability. Since b is chosen uniformly at random we have

$$p\Big(\text{Ind}^{\Sigma}_{\mathcal{B}}(\lambda) = 1\Big) = \frac{1}{2}\Big(p\big(\text{Ind}^{\Sigma}_{\mathcal{B}}(\lambda) = 1 : b = 0\big) + p\big(\text{Ind}^{\Sigma}_{\mathcal{B}}(\lambda) = 1 : b = 1\big)\Big) \tag{1}$$

There are two options: 1) It is clear that if $b = 0$ then $\mathcal{B}$ succeeds if and only if $\mathcal{D}^*(\nu, st_\mathcal{A})$ outputs 0. This is because $\mathcal{B}$ outputs the bit b that he is given by $\mathcal{D}^*$. Notice, however, that ν and $st_\mathcal{A}$ are generated as in $\mathsf{Real}^\Sigma_\mathcal{A}(\lambda)$ from which it follows that

$$p\Big(\mathsf{Ind}^\Sigma_\mathcal{B}(\lambda) = 1 : b = 0\Big) = p\Big(\mathcal{D}^*(\nu, st_\mathcal{A}) = 0 : (\nu, st_\mathcal{A}) \leftarrow \mathsf{Real}^\Sigma_\mathcal{A}(\lambda)\Big) \tag{2}$$

2) It is clear that if $b = 1$ then $\mathcal{B}$ succeeds if and only if $\mathcal{D}^*(\nu, st_\mathcal{A})$ outputs 1. Notice, however, that in this case $st_\mathcal{A}$ and ν are constructed as in $\mathsf{Sim}^\Sigma_{\mathcal{A},\mathcal{S}^*}(\lambda)$ from which it follows that

$$p\Big(\mathsf{Ind}^\Sigma_\mathcal{B}(\lambda) = 1 : b = 1\Big) = p\Big(\mathcal{D}^*(\nu, st_\mathcal{A}) = 1 : (\nu, st_\mathcal{A}) \leftarrow \mathsf{Sim}^\Sigma_{\mathcal{A},\mathcal{S}^*}(\lambda)\Big) \tag{3}$$

Abbreviating $\mathcal{D}^*(\nu, st_\mathcal{A})$ to $\mathcal{D}^*(.)$, it then follows from equations 1-3 that

$$\begin{aligned}
p\Big(\mathsf{Ind}^\Sigma_\mathcal{B}(\lambda) = 1\Big) &= \frac{1}{2}\Big(1 - p\big(\mathcal{D}^*(.) = 1 : (\nu, st_\mathcal{A}) \leftarrow \mathsf{Real}^\Sigma_\mathcal{A}(\lambda)\big) + p\big(\mathcal{D}^*(.) = 1 : (\nu, st_\mathcal{A}) \leftarrow \mathsf{Sim}^\Sigma_{\mathcal{A},\mathcal{S}^*}(\lambda)\big)\Big) \\
&= \frac{1}{2} + \frac{1}{2}\Big(p\big(\mathcal{D}^*(.) = 1 : (\nu, st_\mathcal{A}) \leftarrow \mathsf{Sim}^\Sigma_{\mathcal{A},\mathcal{S}^*}(\lambda)\big) - p\big(\mathcal{D}^*(.) = 1 : (\nu, st_\mathcal{A}) \leftarrow \mathsf{Real}^\Sigma_\mathcal{A}(\lambda)\big)\Big) \\
&\geq \frac{1}{2} + \varepsilon(\lambda)
\end{aligned}$$

where $\varepsilon(\lambda)$ is a non-negligible function in λ and where the inequality follows from that $\mathcal{A}$ can distinguish between the outputs of $\mathsf{Real}^\Sigma_\mathcal{A}(\lambda)$ and $\mathsf{Sim}^\Sigma_{\mathcal{A},\mathcal{S}}(\lambda)$ per our assumption. □

We will now prove the other way.

Lemma 1.2. *MK-non-adaptive semantic security implies MK-non-adaptive indistinguishability.*

Proof. We will show using contra-position that if there exists a polynomial-size adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that succeeds in an $\mathsf{Ind}^\Sigma_\mathcal{A}(\lambda)$ experiment with non-negligible advantage over 1/2, then there also exists a polynomial-size adversary $\mathcal{B}$ and a polynomial-size distinguisher $\mathcal{D}$ that can distinguish in an experiment between the outputs of $\mathsf{Real}^\Sigma_\mathcal{A}(\lambda)$ and $\mathsf{Sim}^\Sigma_{\mathcal{A},\mathcal{S}}(\lambda)$.

Let $\mathcal{B}$ be the adversary that computes $(H^0, H^1, C \subseteq O, \{K_d\}_C, st_\mathcal{A}) \leftarrow \mathcal{A}_1(1^\lambda)$. This tuple consists of two histories for the indistinguishability game (which requires that they share the same trace), a choice in which corpus owners are colluding with the adversary $C \subseteq O$, the document keys they own $\{K_d\}_C$, and the state of the adversary. Adversary $\mathcal{B}$ will now sample $b \xleftarrow{\$} \{0, 1\}$ in the experiment and he then outputs the history H^b and state $st_\mathcal{B} = (st_\mathcal{A}, b)$. Let $\mathcal{D}$ be the distinguisher that is given the output $(\nu, st_\mathcal{B})$ of either $\mathsf{Real}^\Sigma_\mathcal{A}(\lambda)$ or $\mathsf{Sim}^\Sigma_{\mathcal{A},\mathcal{S}}(\lambda)$. $\mathcal{D}$ works as follows:

1. he parses $st_\mathcal{B}$ into $(st_\mathcal{A}, b)$ and ν into $(xSet, qSet, uSet)$ respectively.
2. he computes $b' \leftarrow \mathcal{A}_2(st_\mathcal{A}, xSet, qSet, uSet)$
3. he outputs 1 if $b' = b$ and 0 otherwise.

Clearly if $\mathcal{A}_2$ has negligible advantage then $b' = b$ for about half of the guesses. Thus, to gain an advantage $\mathcal{A}_2$ needs to have an advantage. We will now prove this. Note though that since $\mathcal{A}_1$ and $\mathcal{A}_2$ are of polynomial size, so are $\mathcal{B}$ and $\mathcal{D}$. Thus it indeed only remains to analyse the success probability of $\mathcal{D}$. There are two options:

- If the pair $(\nu, st_{\mathcal{B}})$ is the the output of $\text{Real}^{\Sigma}_{\mathcal{A}}(\lambda)$ then $\nu = (xSet^b_O, qSet, uSet)$ and $st_{\mathcal{B}} = (st_{\mathcal{A}}, b)$. This means that $\mathcal{D}$ will output 1 if and only if $\mathcal{A}_2(st_{\mathcal{A}}, xSet^b, qSet, uSet)$ succeeds in guessing b. Notice that $\mathcal{A}_1$ and $\mathcal{A}_2$'s views while being simulated by $\mathcal{B}$ and $\mathcal{D}$ respectively are identical to the views they would have during an $\text{Ind}^{\Sigma}_{\mathcal{A}}(\lambda)$ experiment. Thus

$$p\Big(\mathcal{D}(\nu, st_{\mathcal{B}}) = 1 : (\nu, st_{\mathcal{B}}) \leftarrow \text{Real}^{\Sigma}_{\mathcal{A}}(\lambda)\Big) = p\Big(\text{Sim}^{\Sigma}_{\mathcal{A},\mathcal{S}}(\lambda) = 1\Big)$$
$$\geq 1/2 + \varepsilon(\lambda),$$

  where $\varepsilon(\lambda)$ is some non-negligible function in λ. The inequality follows from our assumption that $\mathcal{A}$ can distinguish with non-negligible advantage in the indistinguishbility experiment.

- If the pair $(\nu, st_{\mathcal{B}})$ is the the output of $\text{Sim}^{\Sigma}_{\mathcal{A},\mathcal{S}}(\lambda)$ for some arbitrary polynomial-size simulator $\mathcal{S}$, then ν will be independent of b because $\tau(H^0) = \tau(H^1)$ and $st_{\mathcal{A}}$ outputted by $\mathcal{A}_1$ is also independent of b. It follows then that $\mathcal{A}_2$ will guess b with probability $1/2$ (a pure random guess). Thus

$$p\Big(\mathcal{D}(\nu, st_{\mathcal{B}}) = 1 : (\nu, st_{\mathcal{B}}) \leftarrow \text{Sim}^{\Sigma}_{\mathcal{A},\mathcal{S}}(\lambda)\Big) = 1/2$$

By combining the equations from the two cases we get that

$$p\Big(\mathcal{D}(\nu, st_{\mathcal{B}}) = 1 : (\nu, st_{\mathcal{B}}) \leftarrow \text{Real}^{\Sigma}_{\mathcal{A}}(\lambda)\Big) - p\Big(\mathcal{D}(\nu, st_{\mathcal{B}}) = 1 : (\nu, st_{\mathcal{B}}) \leftarrow \text{Sim}^{\Sigma}_{\mathcal{A},\mathcal{S}}(\lambda)\Big) \geq \varepsilon(\lambda)$$

□

Theorem 1. *MK-non-adaptive indistinguishability is equivalent to MK-non-adaptive semantic security.*

Proof. The proof follows direcdtly from lemma 1.1 and 1.2 □

8 Unified security definition

Having defined semantic security for MKSE and proven that [13]'s definition is equivalent to semantic security, we will now try to unify the different security definitions. Immediately we note however that in defining semantic security for MKSE we ended up with a more complicated definition than semantic security in MUSE as defined by [6]. As we have seen, a large part of the issue is that MKSE has multiple corpus owners rather than the one corpus owner in MUSE. As such, we will not unify our MKSE definition. Instead, we will give one unified MUSE definition of security. Therefore, the unification of the four security definitions will not result in one set of real-ideal games that describes NA-semantic security as a whole. Rather, we will have one definition for NA-MK-semantic security (definition 26) and two definitions for MU-semantic security, namely an adaptive and non-adaptive version.
We will now unify the MUSE security definitions. We will do so by first unifying the security games by [33] and [4], and then we will generalise it further in our definition.

8.1 Unifying semantic security (MUSE)

When looking at the security definitions by Wang et al. [33] and Chamani et al. [4] they seem similar, though there is some vagueness. The auxiliary information for example is described/introduced by

[33] as more general information than document keys. It's usage however seems to suggest that the auxiliary information is used to capture the scheme specifics such as the OMAP and keyword counters. The auxiliary data is introduced by [4] as "aux_u which is auxiliary data for user u and contains $\{K_d\}_{d \in Access(u)}$". Remember however that their implementation does not use document keys, so what does it contain? It is not specified what aux_u is or exactly contains. Concerning aux they write "**UserKeys** and **AccessList** are stored locally in σ (which plays the role of aux in DMUSSE definition and is given to all other procedures)". This would suggest that $\sigma = aux = \{\textbf{UserKeys}, \textbf{AccessList}\}$ but contrary to this $\sigma = \{K, aux, K_D\}$ in the real-world security game.

In section section 3.5/tables 1-3 we have explained what the local and auxiliary information are, what they contain, and what is stored by which party. In that section however there was equality between local information and auxiliary data since we compared four different papers. Since in the current unification we are going to look at the unification of the security games by [33] and [4] we will limit the local information and the auxiliary information now to this papers. Thus we get:

- $\iota = (K, Cnt, \{\iota_u, aux_u\}_{u \in U})$.
- $aux = \emptyset$.
- $\iota_u = (K_u, aux_u)$.
- $aux_u = \emptyset$ for NFNU and FNU, and $aux_u = Cnt_u$ for FU [33].
- $aux_u = \emptyset$ for O-μSE, and $aux_u = Cnt_u$ for Q-μSE [4].
- DB as in section 3.5/tables 1-3.

where ι_u is the locally stored information by u. We now have that the auxiliary data is scheme-specific just like in [33] and we assume that we use the auxiliary data as intended by [4] though we can't guarantee this since their usage of auxiliary information remains unclear.

Throughout this section we use the broad interpretation of keyword counters Cnt as specified in section 5.1. Note that in that section we also explained how keyword counters that were given to multiple parties were only written down once. In this section we will use the same convention, meaning that aux'_u possibly updates the server.

When updates are made by a protocol to auxiliary data, the database, counters, etc., we will prime it. As such, when aux' is outputted it means that an update needs to be applied to aux. Note that when a users local information or auxiliary information is updated, it is also part of the corpus owner's information and he needs to update it as well. Finally, we remember the reader that the Enroll protocol is covered by Setup (cf. section 5.1). We are now going to unify the security definitions by [33] and [4] (cf. 6.3 and 6.4 respectively). Using $Stp = \emptyset$ for [33] and $Stp = \{N, |W|\}$ for [4] and we get:

Definition 27 (Adaptive semantic security (MUSE))**.** *Given a leakage function $\mathcal{L} = \{\mathcal{L}^{Stp}, \mathcal{L}^{Srch}, \mathcal{L}^{Shr}\}$ for static SSE schemes or leakage function $\mathcal{L} = \{\mathcal{L}^{Stp}, \mathcal{L}^{Srch}, \mathcal{L}^{Shr}, \mathcal{L}^{Unsh}, \mathcal{L}^{Upl}, \mathcal{L}^{Upd}\}$ for dynamic SSE schemes, where $\mathcal{L}^{Stp}$ corresponds to the leakage during Setup and likewise for the rest of the leakages and functions. A multi-user SSE scheme Σ is adaptively-secure in the presence of corrupted participants with respect to the leakage function if and only if for any PPT adversary $\mathcal{A}$ issuing a polynomial number*

of queries q there exists a stateful PPT simulator $\mathcal{S}$ = (SimSetup, SimSearch, SimShare,SimUnshare, SimUpload, SimUpdate, SimEnroll) such that

$$p\Big(\text{Real}_{\mathcal{A}}^{U,C,\Sigma}(\lambda,n)=1\Big)-p\Big(\text{Ideal}_{\mathcal{A},\mathcal{S},\mathcal{L}}^{U,C,\Sigma}(\lambda,n)\Big)\leq negl(\lambda),$$

with the real and ideal security games as defined below.

$b \leftarrow \text{Real}_{\mathcal{A}}^{U,C,\Sigma}(\lambda,n)$

1 : $(D, U, C, st_{\mathcal{A}}) \leftarrow \mathcal{A}(1^\lambda)$;
2 : $K \leftarrow$ OwnerKeyGen(1^λ);
3 : $K_U \leftarrow$ UserKeyGen(1^λ);
4 : $(Cnt, aux_U; DB) \leftarrow$ Setup$_K(D, K_U, Stp)$;
5 : **for** $j \in [n]$:
6 : $q_j \leftarrow \mathcal{A}(1^\lambda, \iota_C, DB, \tau_{j-1}, st_{\mathcal{A}})$;
7 : **if** $q_j.type$ *is* Search **then** $([Queue_{q_j.u}], [OMAP_{q_j.u}], D_{q_j.u}(q_j.w)) \leftarrow$ Search$(K_{q_j.u}, q_j.w, aux_{q_j.u}; DB)$;
8 : **if** $q_j.type$ *is* Share **then** $(aux'_{q_j.u}; DB') \leftarrow$ Share$_K(q_j.u, q_j.d, aux_{q_j.u}; DB)$;
9 : **if** $q_j.type$ *is* Unshare **then** $(aux'_{q_j.u}; DB') \leftarrow$ Unshare$_K(q_j.u, q_j.d, aux_{q_j.u}; DB)$; \\ *dynamic only*
10 : **if** $q_j.type$ *is* Upload **then** $(Cnt'; DB') \leftarrow$ Upload$_K(q_j.d, Cnt)$; \\ *dynamic only*
11 : **if** $q_j.type$ *is* Update **then**
$(Cnt', aux'_U; DB') \leftarrow$ Update$_K(q_j.id, q_j.WList, q_i.op, Cnt, K_U; DB)$; \\ *dynamic only*
12 : **return** $b \leftarrow \mathcal{A}(1^\lambda, DB, \iota_C, \tau_n, st_{\mathcal{A}})$

Here $\tau_j = (t_0, \ldots, t_j)$ with t_i being the messages from the user/owner to Server in the i'th query. We assume without loss of generality that $b = 1$ is the Real game.

$b \leftarrow \text{Ideal}_{\mathcal{A},\mathcal{S},\mathcal{L}}^{U,C,\Sigma}(\lambda,n)$

1 : $(D, U, C, st_{\mathcal{A}}) \leftarrow \mathcal{A}(1^\lambda)$;
2 : $st_{\mathcal{S}} \leftarrow$ SimOwnerKeyGen;
3 : $(st_{\mathcal{S}}, K_C) \leftarrow$ SimUserKeyGen(1^λ);
4 : $(st_{\mathcal{S}}, aux_C; DB) \leftarrow st_{\mathcal{S}} \cup$ SimSetup$_K((c, K_c)_{c \in C}, D, Stp)$;
5 : **for** $j \in [n]$:
6 : $q_j \leftarrow \mathcal{A}(1^\lambda, \iota_C, DB, \tau_{j-1}, st_{\mathcal{A}})$;
7 : **if** $q_j.type$ *is* Search **then** $([Queue_{q_j.c}], [OMAP_{q_j.c}], st_{\mathcal{S}}; t_j) \leftarrow$ SimSearch$(st_{\mathcal{S}}, \mathcal{L}^{Srch}(q_j, q_j.u); DB)$;
8 : **if** $q_j.type$ *is* Share **then** $(aux'_c, st_{\mathcal{S}}; DB', t_j) \leftarrow$ SimShare$(st_{\mathcal{S}}, \mathcal{L}^{Shr}(q_j, q_j.u); DB)$;
9 : **if** $q_j.type$ *is* Unshare **then** $(aux'_c, st_{\mathcal{S}}; DB', t_j) \leftarrow$ SimUnshare$(st_{\mathcal{S}}, \mathcal{L}^{Unsh}(q_j, q_j.u); DB)$; \\ *dyn. only*
10 : **if** $q_j.type$ *is* Update **then** $(st_{\mathcal{S}}; DB', t_j)) \leftarrow$ SimUpload$(st_{\mathcal{S}}, \mathcal{L}^{Upl}(q_j))$; \\ *dynamic only*
11 : **if** $q_j.type$ *is* Update **then** $(aux'_C, st_{\mathcal{S}}; DB', t_j)) \leftarrow$ SimUpdate$(st_{\mathcal{S}}, \mathcal{L}^{Upd}(q_j, q_j.u); DB)$; \\ *dyn. only*
12 : **return** $b \leftarrow \mathcal{A}(1^\lambda, DB, \iota_C, \tau_n, st_{\mathcal{A}})$

Changes to [33] and [4]: Our simulated protocols return aux'_c (or aux'_C for SimUpdate). This differs from [4], but it is necessary since the algorithms/protocols potentially update a corrupted user's auxiliary information. The adversary is assumed to have access to all information available to the corrupted users and thus the auxiliary information of corrupted users must be part of the output: the adversary can observe the changes. Note that we have given the adversary access to ι_C instead of aux_C, which [33] and [4] do. This is because the adversary is supposed to receive the user keys, which are stored in ι_u and not in aux_u. Note though that only $aux_u \subsetneq \iota_u$ can get updated and as such we output aux_u instead of ι_u in the algorithms/protocols. We have however not only added aux_c, we have also added the adversary's state $st_{\mathcal{A}}$.

Additional security requirements: We would like to point out that some definitions require the protocols to happen in a certain order. They can require for example that all Share queries occur before any Search query occurs. The reason behind this is that if there is no additional security requirement then the adversary can perform a replay attack after all Share queries have taken place. This possibly leaks additional information. Share queries affecting the Search results is only one example of how queries can affect each other. Now, our definition requires that the inequality holds "for any PPT adversary". It requires then in particular that it holds for an adversary that queries in a particular order. It does not matter that such an adversary can first ask Share queries and then ask Search queries. As such, our security definition is a generalisation; it covers every query order possible.

One example of a security property that influences the order of queries we have already seen and is share-forward privacy. We discussed it in section 6.3.1. Other examples of such properties are "forward privacy" and "backward privacy" and fall out of the scope of our paper. Some information regarding these properties can however be found in the appendix or in the literature overview (cf. section 9).

One restriction, one change: Our generalised definition for adaptive semantic security in MUSE is easily restricted to the non-adaptive setting. We only impose that the adversary generates all queries before any of them are executed. To implement this, we move the adversary's query generation outside of the for-loop. We then get:

Definition 28 (Non-adaptive semantic security (MUSE))**.** *Given a leakage function $\mathcal{L} = \{\mathcal{L}^{Stp}, \mathcal{L}^{Srch}, \mathcal{L}^{Shr}\}$ for static SSE schemes or leakage function $\mathcal{L} = \{\mathcal{L}^{Stp}, \mathcal{L}^{Srch}, \mathcal{L}^{Shr}, \mathcal{L}^{Unsh}, \mathcal{L}^{Upl}, \mathcal{L}^{Upd}\}$ for dynamic SSE schemes, where $\mathcal{L}^{Stp}$ corresponds to the leakage during Setup and likewise for the rest of the leakages and functions. A multi-user SSE scheme Σ is adaptively-secure in the presence of corrupted participants with respect to the leakage function if and only if for any PPT adversary $\mathcal{A}$ issuing a polynomial number of queries q there exists a stateful PPT simulator $\mathcal{S} =$* (SimSetup, SimSearch, SimShare, SimUnshare, SimUpload, SimUpdate) *such that*

$$p[\text{Real}_{\mathcal{A}}^{U,C,\Sigma}(\lambda, n) = 1] - p[\text{Ideal}_{\mathcal{A},\mathcal{S},\mathcal{L}}^{U,C,\Sigma}(\lambda, n)] \leq negl(\lambda),$$

with the real and ideal security games as defined below.

$b \leftarrow \mathsf{Real}^{U,C,\Sigma}_{\mathcal{A}}(\lambda, n)$

1 : $(D, U, C, st_{\mathcal{A}}) \leftarrow \mathcal{A}(1^\lambda)$;
2 : $K \leftarrow \mathsf{OwnerKeyGen}(1^\lambda)$;
3 : $K_U \leftarrow \mathsf{UserKeyGen}(1^\lambda)$;
4 : $([Cnt], aux_U; DB) \leftarrow \mathsf{Setup}_K(K_U, D, Stp)$;
5 : $qSet \leftarrow \mathcal{A}(1^\lambda, \iota_C, DB, \tau_{j-1}, st_{\mathcal{A}})$; \\ $|qSet| = n$
6 : **foreach** $q \in qSet$:
7 : **if** $q.type$ is Search **then** $([Queue_{q.u}], [OMAP_{q.u}], D_{q.u}(q.w)) \leftarrow \mathsf{Search}(K_{q.u}, q.w, aux_{q.u}; DB)$;
8 : **if** $q.type$ is Share **then** $(aux'_{q.u}; DB') \leftarrow \mathsf{Share}_K(q.u, q.d, aux_{q.u}; DB)$;
9 : **if** $q.type$ is Unshare **then** $(aux'_{q.u}; DB') \leftarrow \mathsf{Unshare}_K(q.u, q.d, aux_{q.u}; DB)$; \\ *dynamic only*
10 : **if** $q.type$ is Upload **then** $(Cnt'; DB') \leftarrow \mathsf{Upload}_K(q.d, Cnt)$; \\ *dynamic only*
11 : **if** $q.type$ is Update **then** $(Cnt', aux'_U; DB') \leftarrow \mathsf{Update}_K(q.id, q.WList, q_i.op, Cnt, K_U; DB)$; \\ *dyn. only*
12 : **return** $b \leftarrow \mathcal{A}(1^\lambda, DB, \iota_C, \tau_n, st_{\mathcal{A}})$

Here $\tau_j = (t_0, \ldots, t_j)$ *with* t_i *being the messages from the user/owner to Server in the* i*'th query. We assume without loss of generality that* $b = 1$ *is the Real game.*

$b \leftarrow \mathsf{Ideal}^{U,C,\Sigma}_{\mathcal{A},\mathcal{S},\mathcal{L}}(\lambda, n)$

1 : $(D, U, C, st_{\mathcal{A}}) \leftarrow \mathcal{A}(1^\lambda)$;
2 : $st_{\mathcal{S}} \leftarrow \mathsf{SimOwnerKeyGen}$;
3 : $(st_{\mathcal{S}}, K_C) \leftarrow \mathsf{SimUserKeyGen}(1^\lambda)$;
4 : $(st_{\mathcal{S}}, aux_C; DB) \leftarrow st_{\mathcal{S}} \cup \mathsf{SimSetup}_K((c, K_c)_{c \in C}, D, Stp)$;
5 : $qSet \leftarrow \mathcal{A}(1^\lambda, \iota_C, DB, \tau_{j-1}, st_{\mathcal{A}})$; \\ $|qSet| = n$
6 : **foreach** $q \in qSet$:
7 : **if** $q.type$ is Search **then** $([Queue_{q.c}], [OMAP_{q.c}], st_{\mathcal{S}}; t_j) \leftarrow \mathsf{SimSearch}(st_{\mathcal{S}}, \mathcal{L}^{Srch}(q, q.u); DB)$;
8 : **if** $q.type$ is Share **then** $(aux'_c, st_{\mathcal{S}}; DB', t_j) \leftarrow \mathsf{SimShare}(st_{\mathcal{S}}, \mathcal{L}^{Shr}(q, q.u); DB)$;
9 : **if** $q.type$ is Unshare **then** $(aux'_c, st_{\mathcal{S}}; DB', t_j) \leftarrow \mathsf{SimUnshare}(st_{\mathcal{S}}, \mathcal{L}^{Unsh}(q, q.u); DB)$; \\ *dynamic only*
10 : **if** $q.type$ is Update **then** $(st_{\mathcal{S}}; DB', t_j)) \leftarrow \mathsf{SimUpload}(st_{\mathcal{S}}, \mathcal{L}^{Upl}(q))$; \\ *dynamic only*
if $q.type$ is Update **then** $(aux'_C, st_{\mathcal{S}}; DB', t_j)) \leftarrow \mathsf{SimUpdate}(st_{\mathcal{S}}, \mathcal{L}^{Upd}(q, q.u); DB)$; \\ *dynamic only*
11 : **return** $b \leftarrow \mathcal{A}(1^\lambda, DB, \iota_C, \tau_n, st_{\mathcal{A}})$

Now, our definition 28 above is indeed a unified definition for non-adaptive semantic security (MUSE). It can for example immediately by applied to the security definition by Patel et al. [20] (who provide a static SSE scheme). In order to apply our definition to Patel et al. we:

- do not output any of the optional output.
- use $Stp = \emptyset$.

- use $\iota = \{K, K_U, K_D\}$.
- require that the output of the Query subalgorithm and the Search result are both part of the Search-leakage function $\mathcal{L}^{Srch}$.
- do not use any of the dynamic protocols
- use $\iota_u = \{K_u, aux_u, \{K_d\}_{d \in Access_u}, \}$.
- use $aux_u = \{pointer(\sigma_{u,d}) : d \in Access_u\}$.

where $Access_u$ is the set of document identifiers of the documents that u has been granted access to.

9 Literature overview

The idea of SSE was put forth by Song et al. [26] and was a response to practical limitations to oblivious RAM (ORAM) methods. Their work had some limitations such as that it was proven to be secure but not proven to be searchable [6], vulnerable to statistical attacks and that it had a search time linear in the corpus size [26].

Two notable papers were written with a primary focus on addressing these limitations. The first paper was by Goh et al. [11], who defined IND-CKA secure and IND2-CKA secure, though it was not specifically designed for SSE and as such it did not prevent trapdoor leakage. Its security requirement was that no information about the underlying documents can be learned that cannot be learned from a given trapdoor. Curtmola et al. [6] showed in their paper that their security definitions were insufficient, as they notice that the security requirement specifically holds "against adversaries that can convince the client to generate index and trapdoors chosen by the adversary". This way they proved that an IND2-CKA SSE scheme with provably secure trapdoors can still leak keywords.

The second paper was by Chang and Mitzenmacher [5], who had a security requirement similar to the definitions by [11]. Curtmola et al. [6] prove that any SSE scheme and specifically insecure SSE schemes can trivially satisfy this security requirement. The issue with the definition of [5] was that for all corpus and keyword sets they required the existence of a simulator satisfying some probability rather than – as it should have been – requiring that there exists a simulator that satisfies the same probability but for all corpus and keywords sets. The difference here is in the order of determining the simulator and the corpus-keywords sets.

In the history of SSE research the paper by Curtmola et al. [6] is a landmark event that divides the security standard in a clear pre-Curtmola time period and a post-Curtmola time period. Before Curtmola et al. published the first version of their paper in 2006, the SSE security definitions typically revolved around an adversary gaining a trapdoor oracle and being unable to distinguish between two sets of corpus-trapdoors. To address the newly identified weaknesses in [11] and [5], Curtmola et al. came up with their own security definitions. They defined indistinguishability and semantic security for SSE schemes. For the existing adversarial framework they proved equivalence between the two definitions. Additionally, they introduced a new adversarial model. In this new model the adversary is not required to make his queries known beforehand but is instead allowed to choose queries based on previously acquired trapdoors and search outcomes. This new adversary is called an adaptive adversary, whereas the old adversary is called a non-adaptive adversary. In this new adversarial model adaptive semantic security implies adaptive indistinguishability but not vice versa.

With this new adversary there is a clear distinction between the pre- and post-Curtmola time. In conclusion, [6] addrssed previous limitations with their two new security definitions and they even unified them by showing equivalence in the non-adaptive case. More importantly, they introduced a new adversarial model which allowed for additional leakage and then came up with new “adaptive” security definitions to combat this, with adaptive semantic security being new security standard.

Now, Curtmola et al. considered a multi-user setting (MUSE) where the corpus was shared in its entirety with the recipient. Popa et al. [21] were the first to consider sharing a (sub)set with users rather than the entire corpus. They came up with a new setting called the multi-key setting (MKSE). MKSE distinguishes itself from MUSE in the number of encryption keys: in MKSE one query token is sent to the server which is able to search over multiple document sets – each possibly encrypted with a different key – whereas in MUSE one query token is sent to the server which is able to search over all document sets encrypted with the same key. Popa et al.’s [21] MKSE construction was improved by several works in terms of server storage optimisation, elimination of the required trusted third party, mitigation of dictionary sharing by a malicious corpus owner, and security improvements ([16], [28]).

The security guarantee [21] had in mind was only allowing a per document hit-or-miss given a keyword. However, [12] devised an attack that broke the proposed security. The principle allowing the attack was that the share keys were generated independently of the shared set. As a result, the aforementioned improvements to [21] were vulnerable to the same attack because they too had set-independent share keys. Besides this, an additional vulnerability was demonstrated by the attack from [29]. The principle allowing the attack was that each keyword in the document was encrypted separately. A query hit thus revealed the matching encrypted keyword. This vulnerability too remained present in the follow up works.

The work by Hamlin et al. [13] also outline problems in [21]’s their line of work. They explained the security flaws and addressed them in their definition and implementation. In doing so, they created a point of reference for other works in terms of security.[19] The problems outlined by them are:

1) the separation of data and query privacy. Although already identified by [6] in MUSE, the separation was again an issue in MKSE. Leaking contents of documents leaks information about the contents of the query and vice versa.

2) the share key being independent of the shared set. A malicious user was able to share a dictionary with an honest user and thus convert the recipient’s queries to the sharer’s queries due to the absence of set dependence in the share key generation. Surprisingly, honest users their information was leaked if a malicious corpus owner shared a document with them.

3) searching by comparing queries and encrypted keywords. The MKSE definition required comparing a given query to individual encrypted keywords. This leakage therefore transcended users and documents [29].

As is readily seen from the second security problem addressed by them, this reference point for security includes what is called cross-user leakage (even though it is not named as such yet by them). They are among the early works considering a server colluding with malicious or corrupted corpus owners (users in MUSE). It should be noted though that the security requirement by [13] is non-adaptive indistinguishability for MKSE rather than the standard adaptive semantic security for MUSE as introduced by [6]. The construction by [13] was recently enhanced to achieve result verifiability

[19] See [20], [33], [4].

– allowing a user to verify that the server returns the proper search result without having omitted or erroneously added any document – using a garbled bloom filter by [27].

As discussed in the introduction, determining when access pattern leakage is acceptable is outside of the scope of Hamlin et al. [13]. This leakage is experimentally assessed however by papers such as [34], [3], and [7].

Curtmola et al. still allowed some leakage, namely that of the trace. The trace consists of a list of trapdoor-document matches called the access pattern; a symmetric binary matrix indicating if keyword i in row i equals keyword j in column j called the search pattern; and the sizes of the documents. The leakage of the trace still invites some attacks. Islam et al. [15] used co-occurrence from the access pattern leakage to formulate an attack on the remaining keywords. This attack was improved by [3] and [22]. It was shown by [17] however that even only leaking the search pattern – i.e. no access pattern leakage – can leak information to formulate an attack, inferring the keywords. Oya et al. [19] formulated new attacks that could leverage search and access pattern leakages, or it could even leverage only the search pattern leakage. It should be noted that attacks and defenses against them are constantly developed and that despite flaws, [13] notice that SSE can provide meaningful security for certain data sets even if imperfect.

Newer works often come with some additional security guarantees. Notable amongst these is "backward privacy". Backward privacy was first introduced by Bost et al. [2] and they defined three types of backward privacy: weak backward privacy, backward privacy leaking the update pattern, and backward privacy leaking the insertion pattern (ordered from weakest to strongest). They also provided several implementations, each with different trade-offs. Notably, they did not consider cross-user leakage.

Besides the different security properties that can be considered, there are also works that use an additional party in a multi-server setting to achieve security. They (necessarily) assume non-colluding servers. Van Rompay et al. [25] are of particular interest as they cover cross-user leakage. Interestingly, they provide different security requirements for different servers. Other papers using a multi-server architecture are [31] who use token-adjustment to preserver search functionality among multi-indexes – which is the support of searching over multiple indexes simultaneously – and use key sharing to implement identity-based encryption; and [30] who focuses privacy in the face of colluding users and low complexity for users.

Lastly, while our work addresses some of the challenges in comparing papers by enabling easier analysis, the study by [8] examines the impact of various leakage-abuse attacks and develops a risk assessment protocol to facilitate comparison.

10 Discussion & conclusion

Our work focused on unifying the theoretical framework used in SSE that supports multiple users. Among the primary contributions is a new definition of semantic security in the multi-key setting, along with a proof of its equivalence to indistinguishability in the presence of a selective adversary colluding with a set of corrupt users. This generalises the proof by [6] and encompasses the unification of terminology, symbolism, and security definitions, which is crucial for the advancement of the field.

A significant concern in SSE is cross-user leakage, where sensitive information might be inferred across different users. Our study specifically addresses this issue by incorporating it into the unified definitions, including our new definition of semantic security for the multi-key setting, and the sub-

sequent equivalence proof. The contributions of this thesis provide clarity to the field of SSE in the following way. Firstly, with our new semantic security (MKSE) we have formulated what semantic security encompassing cross-user leakage would look like in the multi-key setting. Secondly, we have addressed the question if semantic security is stronger than indistinguishability in the MKSE – assuming a selective adversary – by proving that the definitions are equivalent. A limitation to our new definition for semantic security is that the assumption for a non-singular multi-key history can be strong, meaning that it can require a simulator that is more powerful than polynomial. Thirdly, with our unification we offer a framework that accommodates diverse applications.Comparing our unifying framework to previous works by [20], [13], [33], and [4], we find that our framework encompasses and extends these approaches. Our analysis of their design choices and security formulations reveals gaps in the existing implementations. Now, our final semantic security definition aligns closely with that of [4], though we emphasise the need for a resizable dynamic SSE scheme – a limitation we found in the current research.

One limitation to the current approach to defining security that remains is that comparison between implementations remains difficult due to the leakage function being a description. Despite our contribution of a unified framework addressing this problem and indeed improving the situation, it did not address the descriptive issue. There has been other progress in the field, where security properties can require a leakage function to be rewritable in a certain in order to have that security property (e.g. share forward-private (section 6.3), and forward private and backward private (appendix)), but this too did not address the descriptive nature of the leakage function. An alternative way of comparing schemes is by quantifying the cross-user leakage as described by [20]. It should be noted though that none of the papers mentioned in our work, with the natural exception of [20] itself, have quantified the cross-user leakage using this method. Thus the difficulty of comparing schemes remains, albeit that now our unified framework allows for easier comparison between papers even though it did not address the issue of the leakage function. As such, this remains an open problem.

Our research focused on papers addressing cross-user leakage. Hamlin et al. addressed how sharing a document with a user necessarily leaks information about the user's subsequent queries. The issue of when the leakage caused by accepting a shared document is acceptable remains an open option (cf. section 2).

Furthermore, Hamlin et al. assumed a non-adaptive adversary in the multi-key setting and so did our expansion of the theoretical framework. Future work could explore an adaptive adversary in the multi-key setting, as it was shown by Curtmola et al. that in the case of an adaptive adversary semantic security is stronger than indistinguishability. Another option for future research is to extend the adversarial model by allowing adaptive corruption of users (both in the multi-key and the multi-user setting) and explore its influence on security.

Other options for future work are differing design choices – such as incorporating a multi-server approach like [25], who also address cross-user leakage – or additional security properties – such as verifiability [4]. An interesting property to consider could be that of user anonymity, for example to limit the personal data gathered by tech companies who host the database. These directions could extend the current unified framework.

In summary, this thesis advances the field of SSE by providing a unified theoretical framework, addressing critical security concerns, and identifying possibilities for future research.

References

[1] Raphael Bost, Pierre-Alain Fouque, and David Pointcheval. Verifiable dynamic symmetric searchable encryption: Optimality and forward security. *Cryptology ePrint Archive*, 2016.

[2] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1465–1482. ACM, 2017. ISBN 978-1-4503-4946-8. doi: 10.1145/3133956.3133980. URL https://dl.acm.org/doi/10.1145/3133956.3133980.

[3] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. *Cryptology ePrint Archive*, 2016.

[4] Javad Ghareh Chamani, Yun Wang, Dimitrios Papadopoulos, Mingyang Zhang, and Rasool Jalili. Multi-user dynamic searchable symmetric encryption with corrupted participants. *IEEE Transactions on Dependable and Secure Computing*, 20(1):114–130, 2023. ISSN 1545-5971, 1941-0018, 2160-9209. doi: 10.1109/TDSC.2021.3127546. URL https://ieeexplore.ieee.org/document/9613781/.

[5] Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. *Cryptology ePrint Archive*, 2004.

[6] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. *Journal of Computer Security*, 19 (5):895–934, 2011. ISSN 18758924, 0926227X. doi: 10.3233/JCS-2011-0426. URL https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/JCS-2011-0426.

[7] Marc Damie, Florian Hahn, and Andreas Peter. A highly accurate {Query-Recovery} attack against searchable encryption using {Non-Indexed} documents. In *30th USENIX security symposium (USENIX Security 21)*, pages 143–160, 2021.

[8] Marc Damie, Jean-Benoist Leger, Florian Hahn, and Andreas Peter. The statistical nature of leakage in sse schemes and its role in passive attacks. *Cryptology ePrint Archive*, 2023.

[9] Michael L Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0 (1) worst case access time. *Journal of the ACM (JACM)*, 31(3):538–544, 1984.

[10] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. New constructions for forward and backward private symmetric searchable encryption. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1038–1055. ACM, 2018. ISBN 978-1-4503-5693-0. doi: 10.1145/3243734.3243833. URL https://dl.acm.org/doi/10.1145/3243734.3243833.

[11] Eu-Jin Goh. Secure indexes, 2004. URL https://eprint.iacr.org/2003/216.

[12] Paul Grubbs, Richard McPherson, Muhammad Naveed, Thomas Ristenpart, and Vitaly Shmatikov. Breaking web applications built on top of encrypted data. Cryptology ePrint Archive, Paper 2016/920, 2016. URL https://eprint.iacr.org/2016/920. https://eprint.iacr.org/2016/920.

[13] Ariel Hamlin, Abhi Shelat, Mor Weiss, and Daniel Wichs. Multi-key searchable encryption, revisited. In Michel Abdalla and Ricardo Dahab, editors, *Public-Key Cryptography – PKC 2018*, volume 10769, pages 95–124. Springer International Publishing, 2018. ISBN 978-3-319-76577-8 978-3-319-76578-5. doi: 10.1007/978-3-319-76578-5_4. URL http://link.springer.com/10.1007/978-3-319-76578-5_4. Series Title: Lecture Notes in Computer Science.

[14] Yuval Ishai, Omkant Pandey, and Amit Sahai. Public-coin differing-inputs obfuscation and its applications. In *Theory of Cryptography: 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II 12*, pages 668–697. Springer, 2015.

[15] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In *Ndss*, volume 20, page 12. Citeseer, 2012.

[16] Aggelos Kiayias, Ozgur Oksuz, Alexander Russell, Qiang Tang, and Bing Wang. Efficient encrypted keyword search for multi-user data sharing. In *Computer Security–ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part I 21*, pages 173–195. Springer, 2016.

[17] Chang Liu, Liehuang Zhu, Mingzhong Wang, and Yu-an Tan. Search pattern leakage in searchable encryption: Attacks and new construction. *Cryptology ePrint Archive*, 2013.

[18] Haojun Liu, Xinbo Luo, Hongrui Liu, and Xubo Xia. Merkle tree: A fundamental component of blockchains. In *2021 International Conference on Electronic Information Engineering and Computer Science (EIECS)*, pages 556–561. IEEE, 2021.

[19] Simon Oya and Florian Kerschbaum. Hiding the Access Pattern is Not Enough: Exploiting Search Pattern Leakage in Searchable Encryption. *arXiv e-prints*, art. arXiv:2010.03465, October 2020. doi: 10.48550/arXiv.2010.03465.

[20] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Symmetric searchable encryption with sharing and unsharing. *Cryptology ePrint Archive*, 2017. URL https://eprint.iacr.org/2017/973.pdf.

[21] Raluca Ada Popa and Nickolai Zeldovich. Multi-key searchable encryption. Cryptology ePrint Archive, Paper 2013/508, 2013. URL https://eprint.iacr.org/2013/508. https://eprint.iacr.org/2013/508.

[22] David Pouliot and Charles V Wright. The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1341–1352, 2016.

[23] Cambridge University Press. About cambridge dictionary, 2024. URL https://dictionary.cambridge.org/about.html.

[24] Oxford University Press. About the oed, 2024. URL https://www.oed.com/information/about-the-oed.

[25] Cédric Van Rompay, Refik Molva, and Melek Önen. Multi-user searchable encryption in the cloud. In *Information Security Conference*, 2015. URL https://api.semanticscholar.org/CorpusID:17915694.

[26] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proceeding 2000 IEEE symposium on security and privacy. S&P 2000*, pages 44–55. IEEE, 2000.

[27] Yaping Su, Jianfeng Wang, Yunling Wang, and Meixia Miao. Efficient verifiable multi-key searchable encryption in cloud computing. *IEEE Access*, 7:141352–141362, 2019.

[28] Qiang Tang. Nothing is for free: Security in searching shared and encrypted data. *IEEE Transactions on Information Forensics and Security*, 9(11):1943–1952, 2014.

[29] Cédric Van Rompay, Refik Molva, and Melek Önen. A leakage-abuse attack against multi-user searchable encryption. *Proceedings on Privacy Enhancing Technologies*, 2017.

[30] Cédric Van Rompay, Refik Molva, and Melek Önen. Secure and scalable multi-user searchable encryption. In *Proceedings of the 6th International Workshop on Security in Cloud Computing*, pages 15–25, 2018.

[31] Guofeng Wang, Chuanyi Liu, Yingfei Dong, Peiyi Han, Hezhong Pan, and Binxing Fang. IDCrypt: A multi-user searchable symmetric encryption scheme for cloud applications. *IEEE Access*, 6:2908–2921, 2018. ISSN 2169-3536. doi: 10.1109/ACCESS.2017.2786026. URL http://ieeexplore.ieee.org/document/8240885/.

[32] Xiao Shaun Wang, Kartik Nayak, Chang Liu, TH Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. Oblivious data structures. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 215–226, 2014.

[33] Yun Wang and Dimitrios Papadopoulos. Multi-user collusion-resistant searchable encryption with optimal search time. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pages 252–264. ACM, 2021. ISBN 978-1-4503-8287-8. doi: 10.1145/3433210.3437535. URL https://dl.acm.org/doi/10.1145/3433210.3437535.

[34] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: the power of {File-Injection} attacks on searchable encryption. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 707–720, 2016.

11 Appendix

11.1 Verifiability extension to μSE

To transform the Dynamic Multi-User Searchable Encryption (DMUSE) into a verifiable DMUSE (VD-MUSE) Chamani et al. protect the integrity of the enrypted data using a Merkle tree [1], [18]. To do so, the encrypted index I is treated as an array and a Merkle tree is computed over it. The root is then published as the verification digest, allowing users to verify search results.

Besides protecting the corpus, the integrity of the keyword counters also needs to be guaranteed. In Q-μSE the digest must be updated whenever $Queue$ is updated by the corpus owner in document addition, updating, or removal operations, as well as share operations.

In the case of O-μSE, which uses OMAPs that are not only updated after keyword updates but also after search queries, the digest needs to be updated for the same situations as Q-μSE but additionally it needs to be updated after search queries.

The verifiable variants of the schemes are named VQ-μSE and VO-μSE.

Lastly, in order to use the verification protocol and transform a DSSE scheme into a VDSSE scheme, additional output and actions are required.

- Setup additionally outputs a verification token V at the beginning.
- Enroll additionally outputs a verification token V for local storage at the enrolled user.
- Search additionally outputs verification proof Π.
- Every time the verification token is updated – be it whether Share, Update or Unshare updated the amongst others the index, or Search changed an OMAP – all users need to be notified.

11.2 Forward and backward privacy

Whereas the papers before [4] were largely or even fully static, Chamani et al. provide security for updates. Their notion of forward privacy is – informally – that Upload, Share, Unshare and Update operations cannot be connected to previous Search operations. E.g., if a new document is uploaded it should not be possible to tell if this new document contains a keyword that has previously been searched already except for when an authorised user repeats his query containing this keyword. Using the helping functions

$$WLeakage(u,d,t) = \begin{cases} W^t(d) & \text{if } u \text{ is corrupted} \\ |W^t(d)| & \text{if } u \text{ is honest} \end{cases}$$
$$WListLeakage(u,WList) = \begin{cases} WList & \text{if } u \text{ is corrupted} \\ |WList| & \text{if } u \text{ is honest} \end{cases}$$

forward privacy is defined as

Definition 29 (Forward-private). *A dynamic multi-user searchable symmetric encryption scheme with a coalition C of corrupted users is forward-private if and only if the Upload, Share, Unshare, and Update*

leakage functions $(\mathcal{L}^{Upl}, \mathcal{L}^{Shr}, \mathcal{L}^{Unsh}, \mathcal{L}^{Upd})$ *can be written as:*

$$\mathcal{L}^{Upl}(K, d) = \mathcal{L}'(id(d))$$
$$\mathcal{L}^{Shr}(K, u, d^t, t) = \mathcal{L}'(id(d), u, WLeakage(u, d, t))$$
$$\mathcal{L}^{Unsh}(K, u, d, aux_u^D, t) = \mathcal{L}'(id(d), u)$$
$$\mathcal{L}^{Upd}(K, d, WList, mod, aux_U^D, K_U, t) = \mathcal{L}'(d, WListLeakage(u, WList), mode, AccList(d))$$

where $\mathcal{L}'$ *denotes a stateless function.*

The definition above has split [4]'s requirement for $\mathcal{L}^{Shr}$ in a requirement for $\mathcal{L}^{Shr}$ and a requirement for $\mathcal{L}^{Upl}$ due to the protocol differences. Thus the input of $mode$ for $\mathcal{L}^{Shr}$ by [4] is not required.

Let $qSet$ be a list of queries q.

- For Search queries the entry q is given by $(t, w, u, Search)$ where t is the timestamp, w is the searched keyword and u is the user executing the Search query.
- For Update queries the entry is $(t, d, WList, mod, U, Update)$ where d is the document, $WList$ is the set of keywords to be added to or deleted from the document, modus $mod \in \{add, del\}$ indicates addition or deletion, and U is the set of users who are affected by the operation $U = AccList(d)$.
- For Share queries the entry is $(t, d, u, Share)$.
- For Upload queries the entry is (t, d).
- For Unshare queries the entry is $(t, d, u, Unshare)$.

First we define helping functions.

$$\begin{aligned} Time(w, user) = \{(t, id(d)) \Big| & \Big[\big((t, id(d), W^t(d), u, Share) \in qSet : w \in W^t(d)\big) \bigvee \\ & ((t, id(d), WList, add, U, Update) \in qSet : [w \in WList \wedge user \in U]) \Big] \bigwedge \\ & \Big[\forall t' \Big[\big((t', id(d), WList, del, U, Update) \in qSet : [w \in WList \wedge user \in U]\big) \bigvee \\ & (t', id(d), u, Unshare) \in qSet : w \in W^t(d) \Big] \\ & \rightarrow (t' < t) \Big] \} \end{aligned}$$

Here brackets have been rematched and stylistic choices such as colour and different bracket types have been made to improve readability. In words, this is the function that returns all tuples (timestamp, file-identfier) tuples of keyword w for user $user$ that have been added by either a Share or Update operation to DB and that have not been deleted or Unshared afterwards.

$$\begin{aligned} Update(w, user) = \{t \Big| & \Big[(t, d, mod, u, Share) \in qSet : w \in W^t(d) \Big] \bigvee \\ & \Big[(t, d, WList, mod, U, Update) \in qSet : [w \in WList \wedge user \in U] \Big] \} \end{aligned}$$

In words, this function returns the timestamps of sharing and of each addition/deletion operation related to keyword w for user u. We will now define the final helping function:

$$SrchLeakge(w,u) = \begin{cases} w & \text{if } u \text{ is corrupted} \\ \emptyset & \text{if } u \text{ is honest} \end{cases}$$

We will now define backward-privacy.

Definition 30 (Backward-private)**.** *A dynamic multi-user searchable symmetric encryption scheme with a coalition C of corrupted users is backward-private if and only if the Search leakage functions $\mathcal{L}^{Srch}$ can be written as:*

$$\mathcal{L}^{Srch}(K_u, w, aux_u^D; xSet, uSet) = \mathcal{L}'(u, Time(w,u), Update(w,u), SrchLeakage(w,u)).$$

where $\mathcal{L}'$ denotes a stateless function.

Here backward privacy is obtained when the only information that is leaked (if the user is not corrupted) is the files that currently contain w and that u has access to – this is $Time(w,u)$ – and the timestamps of all previous updates for w affecting u – this is $Update(w,u)$. Of course, the searched keyword is leaked as well if the user is corrupted – this is $SrchLeakage(w,u)$.

11.3 Verifiability

Chamani et al. provide an extension to their security definition to achieve verifiability of results. They do so with a Verify protocol:

> $bool \leftarrow$ Verify$_{K_u}(w, V, IdSet, \Pi)$: is a deterministic protocol to verifiy search results. It takes as input user key K_u, searched keyword w, verification token V, the reply to the search keyword $IdSet$, and verification proof Π. It outputs the result $bool \in \{true, false\}$.

Verify is used by a querier to verify a search result. Verify returns “true” if the verification test passes, and “false” if the test fails. Failure can be due to changed keywords (incorrect timestamps), omitted results and falsely included results. With V and Π the user can verify the search result for correctness. We will now define verifiability. The Adversary is the Server colluding with a coalition C of users.

Definition 31 (Verifiable)**.** *An SSE scheme is verifiable if every PPT Adversary $\mathcal{A}$ has only negligible $negl(\lambda)$ advantage in the following security game with a challenger $\mathcal{C}$, given dictionary Δ and security parameter λ:*

1. *$\mathcal{A}$ sends to $\mathcal{C}$ a corpus D, a set of users U and a coalition of corrupted users $C \subsetneq U$.*
2. *$\mathcal{C}$ sends to $\mathcal{A}$ the information available to the coalition aux_C^D and most of Setup's output: $(I, Access, uSet, V)$.*
3. *$\mathcal{A}$ sends query set $qSet$ to $\mathcal{C}$, who executes these queries while recording the (current) state of the database including all scheme variables. $\mathcal{C}$ sends back the appropriate responses.*
4. *$\mathcal{A}$ sends to $\mathcal{C}$ an honest user h, a search result set $IdSet$, and proof Π'. $\mathcal{C}$ runs $Verify_{K_h}(w, V, IdSet, \Pi')$.*

$\mathcal{A}$ wins if Verify returns $true$ whilst $IdSet \neq DB_h(w)$.

In our definition above we do not give the entire output of Setup to the Adversary since Setup also outputs aux^D, the auxiliary information of the corpus owner containing amongst others the master key K and the keys of all users $K_u \in aux_u^D \in aux^D$. Furthermore, in our definition the Adversary also has to send a keyword w.

The basic idea behind Verifiable DMUSSE (VDMUSSE) is to use an authenticated data structure that the user can use to check the search result. Chamani et al. have dedicated a separate section to extending O-μSE and Q-μSE so that the schemes are verifiable as defined above. This extension is given in a high-level approach. The following steps are required:

- The encrypted index I is treated as an array. The Setup protocol calculates a Merkle tree over it and publishes the root as the verification digest V.
- Enroll shares V with the user.
- Users issuing search queries get the corresponding tree proofs from the server and verify each access of I with respect to this digest.
- The integrity of the keyword counters needs to be protected as well. Updates do the digest are required whenever the keyword counter mechanism is changed. Chamani et al. claim that in Q-μSE queue entries are only updated by the owner during Update and Share queries, however – since the queue is flushed – they are also updated by the user after retrieving the queue entries. In O-μSE the keyword counters are stored in OMAPs, and OMAP blocks are shuffled after being in a Search query.
- Upload, Share, Update, and Unshare change I. This requires an update in the verification token and all users are notified of the updated value V'.
- A Search query outputs verificiation proof Π to the user.

The verifiable version of Q-μSE is called VQ-μSE and the verifiable version of O-μSE is called VO-μSE. The verifiable μSE schemes are deployed using a blockchain. The blockchain technology and its implementation is outside of the scope of this thesis and the reader is referred to [4] for the details.