NOVEL PROOFS AND ALGORITHMS FOR RANGE SEARCHABLE ENCRYPTION

Richard Ong Jun Quan[1], Guan Keer[2], Claire-Leia Ng Shean Ee[3], Ruth Ng Ii-Yung[4], and John Khoo Teng Fong[4]

[1] NUS High School of Mathematics and Science, 20 Clementi Ave 1, Singapore 129957
[2] Hwa Chong Institution, 661 Bukit Timah Rd, Singapore 269734
[3] Raffles Girls' School, 2 Braddell Rise, Singapore 318871
[4] DSO National Laboratories, 12 Science Park Drive, Singapore 118225

Abstract. Range Searchable Encryption (RSE) is a cryptographic scheme with applications in cloud storage. It allows users to request a range of data on an untrusted server without leaking sensitive information. Many current RSE schemes make use of a binary tree structure to achieve range search and a cover algorithm to convert a range query to a small set of tokens.

In our paper, we present novel contributions in three areas. Firstly, we prove the security of our novel RSE_Ω scheme through binding it to the security of its underlying primitive Multi-Map Encryption (MME). Secondly, we formalize a widely used result from the security proof of the MME_π scheme, addressing issues with prior proofs that relied on nonstandard assumptions and were difficult to extend. These proofs combine to show that our RSE_Ω scheme is secure. Lastly, we present a novel optimal c-cover algorithm, which is used in range queries, that is generic, efficient and secure where previous algorithms have been limited. We also prove its correctness and show its efficiency improvements. Collectively, these contributions advance the state of RSE and its practical applications.

1 Introduction

Motivation Recent advancements in cloud computing technologies have made it increasingly common for individuals and organisations to outsource their data to cloud services. The use of cloud computing enables users to access their outsourced data conveniently over the internet and scale up services flexibly. Given that the outsourced data could be of a sensitive nature, there is a growing demand for cryptographic schemes that preserve user privacy without it coming at the expense of practical efficiency.

Range Searchable Encryption (RSE) [3, 4] addresses this need by enabling range queries on encrypted databases without revealing information about documents stored and ranges searched. For instance, a hospital using a cloud service such as Google Cloud should use a RSE scheme to search through confidential patient records by patient age.

Background Many current RSE schemes [3, 4, 5, 9, 6] comprise of 2 parts: a cover algorithm which reduces range queries to a small set of tokens and an encrypted binary tree structure used to obtain relevant documents built with a Multimap Encryption Scheme (MME) [2, 7]. Most RSE schemes support cover algorithms which determine the query and results bandwidth, as well as the security and efficiency of the scheme overall.

In this work, we use the Provable Security Framework [1] to quantify the security of cryptographic schemes. In contrast to Practical Security wherein the security of a scheme is shown by testing it against all cryptanalysis techniques, Provable Security aims to mathematically prove the security of a scheme given a security condition. Provable Security allows for the security of a new

complex scheme to be broken down into the security of the sum of its underlying components which can be well-tested secure schemes like AES-CTR and HMAC-SHA-256.

1.1 Our Contributions

1. We introduce a **novel RSE scheme** RSE_Ω which is built upon the MME scheme and present **security proofs** relating the security of our RSE_Ω scheme to the MME scheme. In comparison to our previous scheme R-RSE [8], RSE_Ω provides security guarantees and more efficient search times.
2. We present a **novel proof** for a result bounding the **security of** MME_π [7], a type of MME scheme which supports encrypted keyword search, to the underlying Symmetric Encryption (SE) and Function Family (FF) schemes used to construct it. This result has been used in other works to prove the security of other schemes but has yet to been formalised. We also use this result to provide security guarantees for our RSE_Ω scheme.
3. We present a **novel optimal** (proven to be correct) **cover algorithm** that is generic, efficient and compatible with several state-of-the-art RSE schemes. Previous works [3, 4, 5] have only designed optimal cover algorithms limited to cover sizes $(c = 1, 3)$ or generic ones with unacceptable runtime complexities [8]. We also provide a **proof of optimality**.

1.2 Related Work

In our previous work [8], we designed and implemented a novel optimal cover algorithm (DP-cover) using a dynamic programming approach. DP-cover was significantly faster than brute force and mathematically proven to minimise overhead. We also presented a novel RSE scheme, Ratcheting RSE (R-RSE) which improved upon previous schemes by reducing server-side storage. Demertzis et al. were the first to design RSE schemes [3, 4] and prove their security. Their Log and Constant schemes represent data in a binary tree structure and are used in conjunction with Single Range Cover (SRC), Best Range Cover (BRC) and Universal Range Cover (URC) cover algorithms. This is detailed in Table 1.

Cover Algorithm	Query Size	Query Time	Overhead	Generic
SRC	O(1)	O()	O(n)	✗
BRC	O(logR)	-	-	✗
URC	O(logR)	-	-	✗
DP-cover	O(c)	O(R^c)	O(R)	✓
c-cover (Ours)	O(c)	O(c^2)	O(R)	✓

Table 1: Comparison of existing cover algorithms to our novel c-cover algorithm
R: Range query size, n: dataset size, c: cover size (only applicable for c-cover)
Query Size: size of query sent to server, Query Time: time taken to compute query, Overhead: additional bandwidth beyond requested range

However, BRC and URC are not generic to any cover size while our c-cover algorithm is. Although SRC achieves a constant query size, it is at the expense of high overhead, which could be the size of the whole database in the worst case. DP-cover is the most similar to our novel c-cover algorithm as they are both generic and allow for constant query sizes with acceptable overhead. Nonetheless, our c-cover algorithm offers a signifcantly lower runtime complexity which will make using covers on large ranges efficient practically. In practice, this means that for a range of 200000 and a cover size of 5, our novel algorithm runs around 800000000 times faster.

MME_π is often used as an underlying MME scheme to other more complex schemes in the literature. MME_π is also widely in real-world implementations such as in MongoDB, a popular

database management system. Although a security proof exists [2], it is considered inadequate due to its non-standard assumptions and its lack of extensibility for further security analysis. MME_π is also widely used in real implementations like mongo db and libraries. Its security result is used to justify the security of schemes built upon it. However, this result has yet to be formalized in any work.

2 Preliminaries

In this paper, $()$ denotes empty sequences, n denotes a positive integer, $\leftarrow$ assigns the output of a function or algorithm to a variable and $\leftarrow\!\!\$\ algorithm$ indicates that the $algorithm$ is non-deterministic while $\leftarrow\!\!\$\ \{...\}$ represents a variable being picked uniformly from random from a set. $\{0,1\}^n$ denotes the set of all bitstrings (e.g. 111001100010) of length n.

Table mappings Mappings are captured by lookup tables of the form $\mathbf{T}$. These map labels $l \in \{0,1\}^*$ to values $\mathbf{T}[l] \in \{0,1\}^* \cup \perp$.

Multimap data structure The multimap data type MMdt is a lookup table that maps all labels of a fixed bitlength $\mathsf{MMdt.lLen}$ to sequences of values of fixed bitlength $\mathsf{MMdt.vLenn}$, and all other labels to $()$. $|\mathsf{MMdt}| = \sum_{i=1}^{k}(\#(\mathsf{MMdt}[i]))$, the number of values in MMdt.

Symmetric Encryption Symmetric Encryption SE schemes like AES-CTR comprise of a set of encryption and decryption algorithms which encrypt arbitrary length messages (plaintext) and decrypt encrypted messages (ciphertext). It is defined in Appendix A.1

Function Families Function Families F schemes, also known as keyed cryptographic hash functions, have a deterministic evaluation algorithm that converts arbitrary length data (message) into a fixed-size string of characters (a hash) given a cryptographic key (key) and is defined in Appendix A.2. An example would be HMAC-SHA-256

Multimap Encryption Multimap Encryption MME schemes encrypt a MMdt with its structure preserved to form the Encrypted Data Structure (EDS) lookup table. The EDS is placed on a remote server so clients only have to send an encrypted token of a label for the server to use to search for their corresponding encrypted values which can then be decrypted. MME enables keyword search to be outsourced to untrusted servers by preserving privacy of the database and searches. The full definition of is found in Appendix A.3.

Range Searchable Encryption Range Searchable Encryption RSE schemes encrypt a database where every document is associated with an integer index to generate an EDS. Given a numerical range (a,b) with a and b included, RSE converts it to tokens which is used as an input to a search function that returns the correct documents within the range. Like MME, RSE allows for range search (eg. Age, Time, Date) to be privately outsourced to remote servers. The full definition of RSE is found in Appendix A.4.

3 RSE_Ω

RSE_Ω is an example of a RSE scheme. Figure 1 is an illustrative example of RSE_Ω's functionality and how RSE_Ω uses MME_π. Both are examples of RSE and MME schemes respectively. Full details of RSE_Ω and MME_π algorithms can be found in Appendix B.

In the example, we have a RSE database DB with medical documents of young children, which are represented by their names, stored with labels of their age. For example, "Bob" has the label of 1. This can be represented in a binary tree. The bottommost nodes store the medical documents while nodes further up in the tree map to those nodes. For example, the node (1,1) would map to

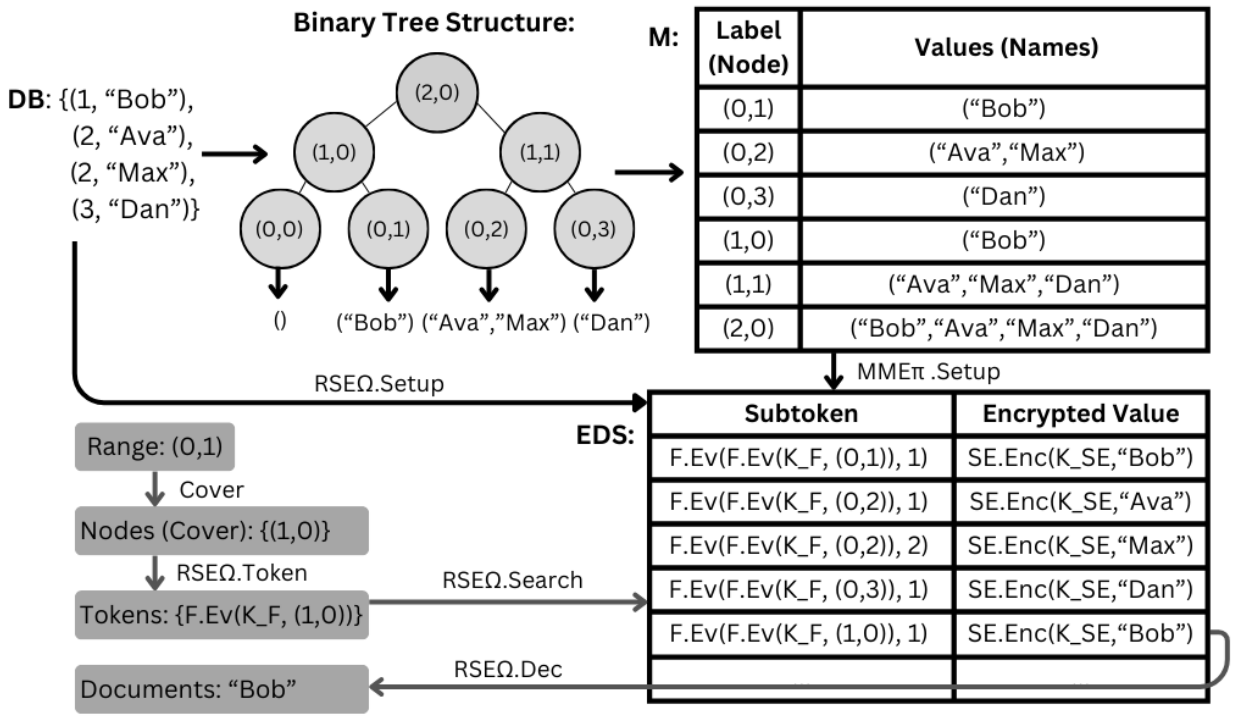

Fig. 1: Example of RSE_{Ω} being used with MME_{π} as its MME scheme in a hospital setting

(0,2) and (0,3) which store "Ava" and "Max". This mapping between nodes is encapsulated when converting the binary tree into a multimap $\mathbf{M}$.

$\mathbf{M}$ is then encrypted using $\mathsf{MME}_{\pi}.\mathsf{Setup}$ to produce an encrypted document structure EDS. For each label in $\mathbf{M}$, the label is tokenised by using a F scheme and a key ($\mathsf{F.Ev}(cryptographic\ key, label) \rightarrow token$). For each value in $\mathbf{M}$ associated to that label, subtokens are generated using the F scheme with the token used as the key and the index of the value being the message($\mathsf{F.Ev}(token, value\ index) \rightarrow subtoken$). Combined, it takes the form of $\mathsf{F.Ev}(\mathsf{F.Ev}(key, label), value\ index) \rightarrow subtoken$. Then, values are encrypted ($\mathsf{SE.Enc}(cryptographic\ key, value) \rightarrow encrypted\ value$). Finally, subtokens map to an encrypted value in EDS.

To query the encrypted database, a range of ages such as ages between 0 and 1 can be chosen. This is converted to a smaller set of nodes by a cover algorithm. Then, they are converted to tokens and used to generate subtokens. Using these subtokens, the server will return the correct encrypted documents from EDS. In this case, the token $\mathsf{F.Ev}(key, (1, 0))$ returns the encrypted document for Bob, which can then be decrypted to give the medical documents for Bob.

Notice that a server would only have access to the EDS and tokens sent. Hence this scheme ensures that the data in the database and any ranges searched remain confidential.

4 Provable Security

4.1 Security Games

Security games are a construction used to compare a cryptographic scheme to its idealised version. This idealised version has some security property that the real scheme tries to achieve. If they "look" the same, it can be said that the cryptographic scheme achieves the desired security properties.

The example below is a game with the security property of "Indistinguishability from random strings" (IND-$) which is a concept applied to symmetrical encryption SE schemes. It means that an ideal SE scheme should encrypt a plaintext such that its ciphertext is indistinguishable from a randomly generated string given the secrecy of a randomly generated key.

Game $\mathrm{G}^{\mathrm{ind}}_{\mathsf{SE}}(A)$	**Oracle** $\textsc{Enc}(M)$
$K_{\mathsf{SE}} \leftarrow\!\!\$\ \mathsf{SE.KS}$	$c_1 \leftarrow\!\!\$\ \mathsf{SE.Enc}(K_{\mathsf{SE}}, M)$
$b \leftarrow\!\!\$\ \{0,1\}$	$c_0 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{SE.cl}(\lvert M \rvert)}$
$b' \leftarrow A^{\textsc{Enc}}$	Return c_b
Return $b = b'$	

The game first randomizes the key (K_{SE}) and bit (b). An Adversary A is now given an oracle. Adversaries can be thought of as attackers of real schemes and them playing games is a way of modeling attackers interacting with real schemes. Oracle acts like a black box which Adversary A can input individual messages into and expect certain outputs. The oracle will always perform $\mathsf{SE.Enc}$ when $b = 1$ or generate a random string if $b = 0$. The adversary can call on the oracle as many times as it needs before it makes a guess on the value of b. We say the adversary wins if it can guess the value of b correctly and that the advantage of adversary A playing the ind game against a specific SE scheme is $\mathrm{Adv}^{\mathrm{ind}}_{\mathsf{SE}}(A)$.

A static game is a game with no oracle. The adversary only has one chance to run the real or ideal scheme before making a guess. A more detailed explanation of this is found in D.1.

Another game, played on F, describes an ideal F as being indistinguishable from a random function, hence their pseudo-random function game, $\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}$. We define this in D.2.

4.2 Quantifying Advantage

To demonstrate this concept, we refer again to the dynamic IND-$ game defined in the earlier section. We let $\Pr[\mathrm{win}]$ be the probability that the game returns True. This happens when $b = b'$, meaning that the adversary correctly guessed the bit. We define:

$$\begin{aligned}
\mathrm{Adv}^{\mathrm{ind}}_{\mathsf{SE}}(A) &= \Pr[\mathrm{win}] - \Pr[\mathrm{lose}] \\
&= \Pr[b' = b] - \Pr[b' \neq b] \\
&= \Pr[b = 1](\Pr[b' = 1 | b = 1] - \Pr[b' = 0 | b = 1]) + \Pr[b = 0](\Pr[b' = 0 | b = 0] - \Pr[b' = 1 | b = 0]) \\
&= 1/2(\Pr[b' = 1 | b = 1] + \Pr[b' = 0 | b = 0] - \Pr[b' = 0 | b = 1] - \Pr[b' = 1 | b = 0]) \\
&= 1/2(\Pr[b' = 1 | b = 1] + (1 - \Pr[b' = 1 | b = 0]) - (1 - \Pr[b' = 1 | b = 1]) - \Pr[b' = 1 | b = 0]) \\
&= \Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0]
\end{aligned}$$

Notice that the advantage is formulated this way to negate the 50% probability that the adversary can win by randomly guessing (if the adversary randomly guesses, $\Pr[\mathrm{win}] = 0.5$ and $\mathrm{Adv}^{\mathrm{ind}}_{\mathsf{SE}}(A) = 0$), and ensures that the advantage remains between 0 and 1.

5 Semantic Security

As mentioned earlier, the core of semantic security involves identifying leakage, which is the information about the data revealed to an outsider, and using it to simulate the scheme. This is done through generating the leakage profile of the scheme. Leakage profiles in terms of structured encryption, which MME and RSE fall under, include **access-pattern** (which query corresponds

to which data), **volume leakage** (the number of documents returned by each query and the total number in the database), and **query equality pattern** (which queries are repeat queries).

While this information is about the data, it is permitted since it offers no real insight into the contents of the database. A simulator simulates how the scheme would act ideally given the details in the leakage profile. Semantic Security games are played by identifying permissible leakage, and showing that given an randomly generated "ideal" database and queries with the same leakage profile, an attacker has no significant advantage in distinguishing it from a real database.

Simulator and Leakage Functions We define and explain the leakage function ($\mathcal{L}_{\mathsf{MME}_\pi}$)and simulator $\mathcal{S}_{\mathsf{MME}_\pi}$ for MME_π in Appendix D.3. In Appendix D.4, we define the leakage function and simulator for RSE_Ω, which can be built from any generic $\mathcal{L}_{\mathsf{MME}}$ and $\mathcal{S}_{\mathsf{MME}}$ respectively, $\mathcal{L}_\pi$ and $\mathcal{S}_\pi$ being one such example.

Leakage Games In Appendix D.5, we give games for generic MME and RSE, each with one example of a simulator and leakage functions that have been defined in the previous sections. The leakage functions and simulator are used to model how the scheme should work ideally when the bit b is 0 and real scheme is run normally when b is 1. Adversaries would then try to guess the value of b.

6 Security Proofs

6.1 MME_π Security

Theorem 1 : Given MME_π, $\mathcal{L}_\pi$ and $\mathcal{S}_\pi$ as defined and an adversary A, there exists adversaries B, C and D such that:

$$\mathrm{Adv}^{\mathrm{ss}}_{\mathsf{MME},\mathcal{L}_\pi,\mathcal{S}_\pi}(A) \leq \mathrm{Adv}^{\mathrm{ind}}_{\mathsf{SE}}(B) + \mathrm{Adv}^{\mathrm{prf}}_{\mathsf{F}}(C) + (m-x)\cdot \mathrm{Adv}^{\mathrm{prf}}_{\mathsf{F}}(D)$$

where m is the number of labels in $\mathbf{M}$ and $x = |\{l_1, ..., l_n\}|$ is the number of distinct labels sent by adversary A.

The full proof can be found in the D.6. Below outlines a proof sketch.

Proof Sketch : The goal of our proof is to bound the security of MME_π to SE and F which have examples of secure schemes (AES-CTR, HMAC-SHA-256). We do this by constructing adversaries B, C and D that can use any given adversary A. In a sense, these adversaries try to play games on each individual cryptographic component of the larger MME_π scheme (SE and F functions). They do so by modifying parts of the MME oracle and then passing the outputs on to Adversary A. This tests the ability of A on distinguish between real and ideal versions of each component hence reducing the security of MME_π into the security of its parts.

However, since the strategy they employ (using A to make guesses for them) is rudimentary, it may be possible for them to have higher chances of winning using other strategies which gives rise to the inequality. ∎

6.2 Practical Implication of Theorem 1

Corollary 1 : The implication is that the strongest possible adversary A can only ever have as much advantage as the sum of the advantages of the strongest adversaries B, C and D. The contrapositive implies that if we have secure underlying cryptographic parts, the sum of the advantages of B, C and D would be low (which has shown to be the case for AES-CTR and HMAC-SHA-256) and hence the advantage of A would be likewise low. This means the MME_π scheme is secure. MME_π is also a scheme used in many other applications and by real companies, hence a formalised proof of security is very important.

6.3 RSE_{Ω} Security

Theorem : Given RSE_{Ω}, $\mathcal{L}_{\Omega}$ and $\mathcal{S}_{\Omega}$ as defined, and an adversary A, there exists an adversary B such that:

$$\mathrm{Adv}^{\mathrm{ss}}_{\mathsf{RSE}_{\Omega},\mathcal{L}_{\Omega},\mathcal{S}_{\Omega}}(A) \leq \mathrm{Adv}^{\mathrm{ss}}_{\mathsf{MME},\mathcal{L}_{\mathsf{MME}},\mathcal{S}_{\mathsf{MME}}}(B)$$

where $\mathcal{L}_{\mathsf{MME}}$ and $\mathcal{S}_{\mathsf{MME}}$ are the leakage algorithm and simulator for MME.

This proof bounds the security of RSE_{Ω} in terms of the security of the MME used to build it. The idea is that we are able to construct a rudimentary adversary B that can use A to make its guess for it by converting between RSE and MME formats. Hence at minimum, B will always have the same probability of winning as A but a stronger strategy could give a better results, hence the inequality. We outline the full proof below.

Proof : We first define adversary B, which makes use of adversary A, as shown below. Note that Adversary $A = (A_1, A_2)$, and Adversary $B = (B_1, B_2)$.

Adversary B_1

$(St, \mathsf{DB}, (q_1, ..., q_n)) \leftarrow\!\!\$\ A_1$
$\{(a_1, D_1), ..., (a_n, D_n)\} \leftarrow \mathsf{DB}$
For $i = 0... \log_2(\mathsf{RSE}_{\Omega}.\mathsf{KS})$:
  For $j = 0...2^i - 1$:
    $\mathsf{up} \leftarrow 2^{\log_2(\mathsf{RSE}_{\Omega}.\mathsf{KS})-i} \cdot (j+1) - 1$
    $\mathsf{down} \leftarrow 2^{\log_2(\mathsf{RSE}_{\Omega}.\mathsf{KS})-i} \cdot j$
    $\mathsf{node} \leftarrow (\log_2(\mathsf{RSE}_{\Omega}.\mathsf{KS}) - i, j)$
    $\mathsf{docs} \leftarrow ((a_i, D_i) : i \in \{1...n\}, \mathsf{down} \leq a_i \leq \mathsf{up})$
    $\mathbf{M}[\mathsf{node}] \leftarrow \mathsf{docs}$
$L \leftarrow ()$
For $i = 1...n$:
  $S_i \leftarrow \mathsf{Cover}(q_i, c)$
  $L \leftarrow L \parallel (n : n \in S_i)$
$(l_1, ..., l_n) \leftarrow L$
Return $(St, \mathbf{M}, (l_1, ..., l_n))$

Adversary $B_2(St, \mathsf{EDS}, (tk_1, ..., tk_n))$

For $i = 1, 1 + c, ..., n - c + 1$:
  $T_i \leftarrow \{tk_x : i \leq x < i + c\}$
$b' \leftarrow\!\!\$\ A_2(St, \mathsf{EDS}, (T_1, T_{1+c}, ...T_{n-c+1}))$
Return b'

We now define a series of hybrids.

Game $\boxed{G_0}, \boxed{\boxed{G_1}}$

$K \leftarrow\$ \mathsf{RSE}_\Omega.\mathsf{KS}$
$(St, \mathsf{DB}, (q_1, ..., q_n)) \leftarrow\$ A_1$
$\boxed{\mathsf{EDS} \leftarrow\$ \mathsf{RSE}_\Omega.\mathsf{Setup}(K, \mathsf{DB})}$
$\boxed{\text{For } i = 1...n:}$
  $\boxed{T_i \leftarrow \mathsf{RSE}_\Omega.\mathsf{Token}(K, q_i, c)}$
$\boxed{\boxed{lk \leftarrow \mathcal{L}_\Omega(\mathsf{DB}, (q_1...q_n))}}$
$\boxed{\boxed{(\mathsf{EDS}, (T_1, ..., T_n)) \leftarrow \mathcal{S}_\Omega(lk)}}$
$b' \leftarrow\$ A_2(St, \mathsf{EDS}, (T_1, ..., T_n))$
Return $b = b'$

Game $\boxed{G_2}, \boxed{\boxed{G_3}}$

$(K_{\mathsf{SE}}, K_{\mathsf{F}}) \leftarrow\$ \mathsf{MME}.\mathsf{KS}$
$(St, \mathsf{DB}, (q_1, ...q_n)) \leftarrow\$ A_1$
$\{(a_1, D_1), ..., (a_n, D_n)\} \leftarrow \mathsf{DB}$
For $i = 0... \log_2(\mathsf{RSE}_\Omega.\mathsf{KS})$:
  For $j = 0...2^i - 1$:
    $\mathsf{up} \leftarrow 2^{\log_2(\mathsf{RSE}_\Omega.\mathsf{KS}) - i} \cdot (j+1) - 1$
    $\mathsf{down} \leftarrow 2^{\log_2(\mathsf{RSE}_\Omega.\mathsf{KS}) - i} \cdot j$
    $\mathsf{node} \leftarrow (\log_2(\mathsf{RSE}_\Omega.\mathsf{KS}) - i, j)$
    $\mathsf{docs} \leftarrow ((a_i, D_i) : i \in \{1...n\}, \mathsf{down} \leq a_i \leq \mathsf{up})$
    $\mathbf{M}[\mathsf{node}] \leftarrow \mathsf{docs}$
$L \leftarrow ()$
For $i = 1...n$:
  $S_i \leftarrow \mathsf{Cover}(q_i, c)$
  $\boxed{T_i \leftarrow \{\mathsf{MME}.\mathsf{Token}(K_{\mathsf{F}}, l) : l \in D_i\}}$
  $\boxed{\boxed{L \leftarrow L || (l : l \in S_i)}}$
$\boxed{\mathsf{EDS} \leftarrow\$ \mathsf{MME}.\mathsf{Setup}(K_{\mathsf{SE}}, \mathbf{M})}$
$\boxed{\boxed{lk \leftarrow \mathcal{L}_{\mathsf{MME}}(\mathbf{M}, L)}}$
$\boxed{\boxed{(\mathsf{EDS}, (tk_1, ...tk_{c*n})) \leftarrow \mathcal{S}_{\mathsf{MME}}(lk)}}$
$b' \leftarrow\$ A_2(St, \mathsf{EDS}, (T_1, ..., T_n))$
Return $b' = 1$

Using the above hybrids, we can show the following equations.

$$\mathrm{Adv}^{\mathrm{ss}}_{\mathsf{RSE}_\Omega, \mathcal{L}_\Omega, \mathcal{S}_\Omega}(A) = \Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1] \tag{1}$$

$$\mathrm{Adv}^{\mathrm{ss}}_{\mathsf{MME}, \mathcal{L}_{\mathsf{MME}}, \mathcal{S}_{\mathsf{MME}}}(B) = \Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1] \tag{2}$$

Combining equations (5) and (6), we can see that:

$$\begin{aligned}\mathrm{Adv}^{\mathrm{ss}}_{\mathsf{RSE}_\Omega, \mathcal{L}_\Omega, \mathcal{S}_\Omega}(A) &= \Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1] \\ &= \mathrm{Adv}^{\mathrm{ss}}_{\mathsf{MME}, \mathcal{L}_{\mathsf{MME}}, \mathcal{S}_{\mathsf{MME}}}(B)\end{aligned}$$

To show equation (5), first let b and b' be the random variables in $\mathrm{G}^{\mathrm{ss}}_{\mathsf{RSE}_\Omega, \mathcal{L}_\Omega, \mathcal{S}_\Omega}(A)$. Observe that G_0 and G_1 are $\mathrm{G}^{\mathrm{ss}}_{\mathsf{RSE}_\Omega, \mathcal{L}_\Omega, \mathcal{S}_\Omega}(A)$ when $b = 1$ and $b = 0$ respectively. Thus,

$$\begin{aligned}\mathrm{Adv}^{\mathrm{ss}}_{\mathsf{RSE}_\Omega, \mathcal{L}_\Omega, \mathcal{S}_\Omega}(A) &= \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{RSE}_\Omega, \mathcal{L}_\Omega, \mathcal{S}_\Omega}(A) : b' = 1 | b = 1] - \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{RSE}_\Omega, \mathcal{L}_\Omega, \mathcal{S}_\Omega}(A) : b' = 1 | b = 0] \\ &= \Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1]\end{aligned}$$

To show equation (6), let b and b' be the random variables in $\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME}, \mathcal{L}_{\mathsf{MME}}, \mathcal{S}_{\mathsf{MME}}}(B)$. Note that by definition, $\mathsf{MME}.\mathsf{KS} = \mathsf{RSE}_\Omega.\mathsf{KS}$. Notice that if we inline Adversary B into $\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME}, \mathcal{L}_{\mathsf{MME}}, \mathcal{S}_{\mathsf{MME}}}(B)$, we get G_2 when $b = 1$ and G_3 when $b = 0$. Next, notice that if we inline $\mathsf{RSE}_\Omega.\mathsf{Setup}$ and $\mathsf{RSE}_\Omega.\mathsf{Token}$ into G_0, we get G_2. Similarly, if we inline $\mathcal{L}_\Omega$ and $\mathcal{S}_\Omega$ into G_1, we get G_3. Thus,

$$\begin{aligned}\mathrm{Adv}^{\mathrm{ss}}_{\mathsf{MME}, \mathcal{L}_{\mathsf{MME}}, \mathcal{S}_{\mathsf{MME}}}(B) &= \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME}, \mathcal{L}_{\mathsf{MME}}, \mathcal{S}_{\mathsf{MME}}}(B) : b' = 1 | b = 1] - \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME}, \mathcal{L}_{\mathsf{MME}}, \mathcal{S}_{\mathsf{MME}}}(B) : b' = 1 | b = 0] \\ &= \Pr[\mathrm{G}_2] - \Pr[\mathrm{G}_3] \\ &= \Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1]\end{aligned}$$

Having shown equations (5) and (6), we thus conclude our proof of the theorem. ∎

Practical Security Combining these two proof together, we show that given the correct choice of SE and F, with secure examples such as AES-CTR and HMAC-SHA-256, we can construct a secure MME_π which can then be used to construct a secure RSE_Ω. Developers will now be able to use RSE_Ω and be assured of its security.

7 C-cover Algorithm

This section covers a contribution in a different area of RSE, particularly in one kind of algorithms used by RSE schemes in general. As mentioned in Section 3 the bottommost nodes of the binary tree store documents. Rather than sending the token for every node within a range, cover algorithms are used in RSE schemes to convert a large range to small set of nodes, known as a cover, that can also be used to retrieve the same documents.

Covers can be split into two types: **exact covers** which use the minimal number of nodes to cover a range exactly and **overcovers** which may cover nodes beyond the range. These extra nodes are known as **overhead** and the number of nodes in a cover are its **size**. An example is illustrated in in Figure 2.

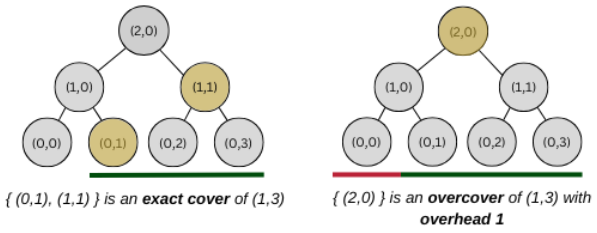

Fig. 2: Example of exact and over cover

C-cover **C-covers** are overcovers with a fixed size c. We say a c-cover is **optimal** if its overhead is minimised. In the previous example, (2,0) is the optimal 1-cover because there is no other 1 cover that has less overhead. A mathemtical defintion for c-cover can be found in Appendix C.

Advantages C-covers improve the **efficiency** of RSE as the query size is fixed to c and less tokens have to be looked up. C-covers also improve the **security** of RSE scheme as the false positives (overhead) reduce the information leaked to an adversarial server. Previously, optimal 1-cover and 3-cover algorithms have been presented. In our previous work, we designed an optimal but inefficient c-cover algorithm (DP-cover) with a runtime complexity of O(R^c) where R is the number of nodes in the range, making it unpractical to use on large databases.

Our contribution We now introduce a **novel efficient and optimal c-cover algorithm with runtime complexity of O(c^2)** and a proof of optimality.

Algorithm Intuition Notice that all optimal covers can be divided into a left and right side consisting of descending triangles as illustrated in Figure 3. We call nodes that that lie at the feet of the largest triangles in an optimal cover **hinge nodes**. The hinge node can be calculated with a formula.

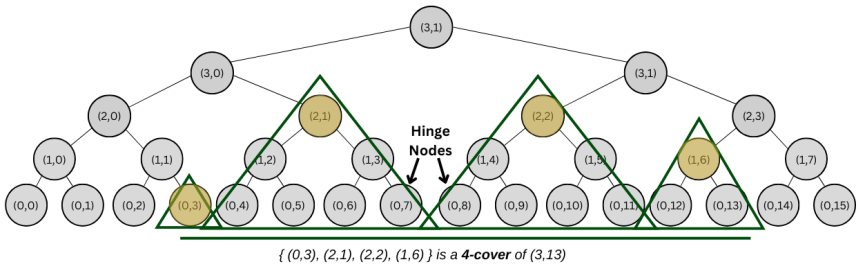

Fig. 3: An example showing the descending triangle structure of an optimal cover

Since there is only one side of overhead to consider in each side, it is trivial to find the optimal cover for the ranges divided by the hinge node. The algorithm then constructs all covers with different distributions of cover sizes on the left and right sides and picks the one with the least total overhead as the optimal c-cover. In Appendix C.2 , we give an optimal c-cover algorithm and prove its optimality.

8 Future Work

Security Comparisons Further work can be done to compare the security of RSE_{Ω} with other existing RSE schemes using game-playing provable security. For example, R-RSE which was outlined in our previous work. More can also be done to study the effects of different types of cover algorithms on the query leakage and security of the scheme. Namely, the effects of universal overcovers, explored in our previous paper, could be looked into.

9 Conclusion

Overcover Algorithm Our novel c-cover algorithm significantly advances prior work by reducing search bandwidth while maintaining the security advantages of overcovers. It also enhances efficiency by reducing search time from exponential to a near constant time in practical cases, making c-covers feasible for real RSE scheme implementations.
Security Proofs Through our proofs, we have shown that the security of MME_{π} can be reduced to the security of the SE and F, and that the security of our RSE_{Ω} scheme only depends on the security of the MME used to build it. Our MME_{π} proof allows for schemes built upon it to be easily analysed in future and for the security of current implementations to be more easily analysed. Our RSE_{Ω} proof allows for the assurance of practical security for RSE_{Ω} so developers can implement it in real-world cloud storage applications, thereby enhancing security and confidentiality for users.

10 Acknowledgements

We would like to thank our mentors, Dr Ruth Ng Ii-Yung and Mr John Khoo Teng Fong, for their continued support and guidance throughout this project.

References

[1] Mihir Bellare and Phillip Rogaway. *Code-Based Game-Playing Proofs and the Security of Triple Encryption*. Cryptology ePrint Archive, Paper 2004/331. 2004. URL: https://eprint.iacr.org/2004/331.

[2] David Cash et al. "Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries". In: *Advances in Cryptology – CRYPTO 2013*. Ed. by Ran Canetti and Juan A. Garay. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 353–373. ISBN: 978-3-642-40041-4.

[3] Ioannis Demertzis et al. "Practical Private Range Search in Depth". In: *ACM Trans. Database Syst.* 43.1 (Mar. 2018). ISSN: 0362-5915. DOI: 10.1145/3167971. URL: https://doi.org/10.1145/3167971.

[4] Ioannis Demertzis et al. "Practical Private Range Search Revisited". In: *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. Ed. by Fatma Özcan, Georgia Koutrika, and Sam Madden. ACM, 2016, pp. 185–198.

[5] Sky Faber et al. "Rich queries on encrypted data: Beyond exact matches". In: *European symposium on research in computer security*. Springer. 2015, pp. 123–145.

[6] Francesca Falzon et al. *Range Search over Encrypted Multi-Attribute Data*. Cryptology ePrint Archive, Paper 2022/1076. 2022. URL: https://eprint.iacr.org/2022/1076.

[7] Ruth Ng et al. *Structured Encryption for Indirect Addressing*. Cryptology ePrint Archive, Paper 2023/1146. 2023. URL: https://eprint.iacr.org/2023/1146.

[8] Richard Jun Quan Ong, Keer Guan, and Ruth Ii-Yung Ng. *New Optimal Algorithms for Computing Binary Tree Overcovers in Range Searchable Encryption*. 2024. URL: https://www.dsta.gov.sg/staticfile/ydsp/projects/files/posters/New_Optimal_Algorithms_for_Computing_Binary_Tree_Overcovers_in_Range_Searchable_Encryption.pdf.

[9] Cong Zuo et al. "Forward and Backward Private DSSE for Range Queries". In: *IEEE Transactions on Dependable and Secure Computing* 19.1 (2022), pp. 328–338.

A Definitions

A.1 Symmetric Encryption

A symmetric encryption scheme is used for encrypting and decrypting strings of arbitrary length. A symmetric encryption scheme SE defines:

- $\mathsf{SE.KS}$, the set of all possible keys K
- $\mathsf{SE.Enc} : K \times M \rightarrow C$, a deterministic encryption function that takes in $K \in \mathsf{SE.KS}$ and message $M \in \{0,1\}^*$ of arbitrary length and returns encrypted message $C \in \{0,1\}^*$ of arbitrary length
- $\mathsf{SE.Dec} : K \times C \rightarrow M$, a deterministic decryption function that takes K and C and returns M
- $\mathsf{SE.cl} : \mathbb{Z} \rightarrow \mathbb{Z}^+$, a ciphertext length function that maps plaintext length to ciphertext length.

We say that a symmetric encryption scheme is correct if for all K, M as above, $\mathsf{SE.Dec}(K, \mathsf{SE.Enc}(K, M)) = M$.

A.2 Function Families

A function family scheme is also known as a cryptographic hash function and maps arbitary length bitstrings to a constant length bitstring. A function family F defines:

- $\mathsf{F.KS}$, the set of all possible keys
- $\mathsf{F.IS}$, the set of all possible inputs
- $\mathsf{F.ol}$, a postive integer denoting fixed length of output
- $\mathsf{F.Ev} : K \times I \rightarrow O$, a deterministic evaluation algorithm that takes in $K \in \mathsf{F.KS}$, $I \in \mathsf{F.IS}$ and returns $O \in \{1,0\}^{\mathsf{F.ol}}$

A.3 MME Definition

Multimap data structure The multimap data type MMdt is a lookup table that maps all labels of some fixed bitlength $\mathsf{MMdt.lLen}$ to sequences of values of fixed bitlength $\mathsf{MMdt.vLenn}$, and all other labels to $()$. We say $|\mathsf{MMdt}| = \sum_{i=1}^{k}(\#(\mathsf{MMdt}[i]))$, the number of labels in MMdt.

A MME scheme MME defines:

- $\mathsf{MME.KS}$, the set of all possible key pairs $(K_{\mathsf{F}}, F_{\mathsf{SE}})$
- lLen, the positive integer that denotes label length
- vLen, the positive integer that denotes document length
- $\mathsf{MME.Eval} : \mathbf{M} \times l \rightarrow \mathbf{M}[l]$, which is a deterministic evaluation function that takes in multi-map $\mathbf{M}$ and $l \in \{1,0\}^{\mathsf{lLen}}$ and returns $\mathbf{M}[l]$
- $\mathsf{MME.Setup} : K \times \mathbf{M} \rightarrow \mathsf{EDS}$, a deterministic encryption function that takes in $\mathbf{M}$ and a key $K \in \mathsf{MME.KS}$ and returns an encrypted document set EDS
- $\mathsf{MME.Token} : K \times l \rightarrow tk$, a token function that takes in K and l and returns tk, a token.
- $\mathsf{MME.Search} : l \times \mathsf{EDS} \rightarrow \{C_1, ..., C_n\}$, a search function that takes in l and EDS and returns a set of encrypted documents, $\{C_1, ..., C_n\}$
- $\mathsf{MME.Dec} : K \times C \rightarrow D$, a decryption function that takes in K, and an encrypted document and returns a document.

We require that for any $K \in \mathsf{MME.KS}$, $\mathbf{M}$ and l that if $\mathsf{EDS} \leftarrow \mathsf{MME.Setup}(K, \mathbf{M})$ and $tk \leftarrow \mathsf{MME.Token}(K, l)$, then $\mathsf{MME.Dec}(K, \mathsf{MME.Search}(tk, \mathsf{EDS})) = \mathsf{MME.Eval}(\mathbf{M}, l)$

A.4 RSE Definition

A RSE scheme RSE defines:

- RSE.KS, the set of all possible keys
- RSE.max, the largest positive integer index possible for a document
- RSE.dl, the positive integer that denotes document length
- RSE.Setup : RSE.KS $\times$ DB $\rightarrow$ EDS, an algorithm which takes a Key $K \in$ RSE.KS, database DB and returns EDS, Encrypted Data Structure. We require that $\mathsf{DB} = \{(a_1, D_1), ..., (a_x, D_x)\}$ for some positive integer x, such that for all $i \in \{1...x\}$,
  - $1 \leq a_i \leq$ RSE.max
  - $D_i \in \{0,1\}^{\mathsf{RSE.dl}}$
  - For all i, j, if $i \neq j$ then $D_i \neq D_j$
  - We say that allDB is the set of all possible databases that meet the above requirements.
- RSE.Token : $K \times q \rightarrow tk$, an algorithm which takes a Key $K \in$ RSE.KS and q, a query consisting of tuple of positive integers, (a, b) where $1 \leq a \leq b \leq$ RSE.max and returns a token tk, bit-string of fixed length.
- RSE.Search : $tk \times$ EDS $\rightarrow \{(a_1, C_1), ..., (a_n, C_n)\}$, an algorithm which takes a bit-string of arbitrary length from the output of RSE.Token and EDS, and a set of encrypted documents $\{(a_1, C_1), ..., (a_n, C_n)\}$.
- RSE.Dec : $K \times (a_i, C_i) \rightarrow (a_i, D_i)$, an algorithm which takes a Key $K \in$ RSE.KS and (a_i, C_i) and returns $(a_i, D_i) \in$ DB

We require that for any $K \in$ RSE.KS , DB and q that if EDS $\leftarrow$ RSE.Setup(K, DB) and $tk \leftarrow$ RSE.Token(K, q), then RSE.Dec$(K,$ RSE.Search$(tk,$ EDS$))$ = RSE.Eval(DB, l)

B Algorithms

B.1 MME algorithms

Alg $\mathsf{MME}_\pi.\mathsf{Setup}(K, \mathbf{M})$
$(K_{\mathsf{SE}}, K_{\mathsf{F}}) \leftarrow K$
For all $\mathbf{M}[l]$ in $\mathbf{M}$:
  $tk \leftarrow \mathsf{F.Ev}(K_{\mathsf{F}}, l)$
  $(v_1, ..., v_n) \leftarrow \mathbf{M}[l]$
  For i in $1, ..., n$:
    $\mathsf{EDS}[\mathsf{F.Ev}(tk, i)] \leftarrow\!\!\$\ \mathsf{SE.Enc}(K_{\mathsf{SE}}, v_i)$
Return EDS

Alg $\mathsf{MME}_\pi.\mathsf{Search}(tk, \mathsf{EDS})$
$i = 1$
While $exist =$ True:
  $tk_i \leftarrow \mathsf{F.Ev}(tk, i)$
  if $\mathsf{EDS}[tk_i] = \perp$:
    $exist =$ False
  else:
    $c_i \leftarrow \mathsf{EDS}[tk_i]$
    $i = i + 1$
Return $(c_1, ..., c_n)$

Alg $\mathsf{MME}_\pi.\mathsf{Token}(K_{\mathsf{F}}, l)$
$tk \leftarrow \mathsf{F.Ev}(K_{\mathsf{F}}, l)$
Return tk
Alg $\mathsf{MME}_\pi.\mathsf{Dec}(K_{\mathsf{SE}}, (c_1, ..., c_n))$
For i in $1, ..., n$:
  $v_i \leftarrow \mathsf{SE.Dec}(K_{\mathsf{SE}}, c_i)$
Return $(v_1, ..., v_n)$
Alg $\mathsf{MME}_\pi.\mathsf{Eval}(l, \mathbf{M})$
$(v_1, ..., v_n) \leftarrow \mathbf{M}[l]$
Return $(v_1, ..., v_n)$

Fig. 4: Algorithm definitions for MME_π

MME_π **Algorithms** Figure 4 defines MME_π, an example of a MME scheme. MME_π.Setup takes in K, the keyset and $\mathbf{M}$ which is the multimap to be encrypted. For each label in $\mathbf{M}$, a token is generated with the keyed cryptographic hash function. Then a sub-token is generated by using a

token as a key and the integer corresponding to the index of a value mapped to by that label as an input to the cryptographic hash function.

The lookup table EDS is the encrypted data set maps subtokens to its respective value encrypted by symmetric encryption. MME_π.Search takes in a token and uses it as the key to the cryptographic hash function to generate subtokens by hashing integers starting from 1 and incrementing each round. It then uses the subtoken to lookup the correct encrypted value until it receives an error after which it returns all encrypted values in a sequence.

B.2 RSE algorithms

Alg$\mathsf{RSE}_\Omega.\mathsf{Setup}(K, \{(a_1, D_1), ..., (a_x, D_x)\})$
For $i = 0... \log_2(\mathsf{RSE}_\Omega.\mathsf{KS})$ do:
  For $j = 0...2^i - 1$ do:
    $\mathsf{right} \leftarrow 2^{\log_2(\mathsf{RSE}_\Omega.\mathsf{KS})-i} \cdot (j+1) - 1$
    $\mathsf{left} \leftarrow 2^{\log_2(\mathsf{RSE}_\Omega.\mathsf{KS})-i} \cdot j$
    $\mathsf{node} \leftarrow (\log_2(\mathsf{RSE}_\Omega.\mathsf{KS}) - i, j)$
    $\mathsf{docs} \leftarrow ((a_i, D_i) : i \in \{1...x\}, \mathsf{left} \leq a_i \leq \mathsf{right})$
    $\mathbf{M}[\mathsf{node}] \leftarrow \mathsf{docs}$
$\mathsf{EDS} \leftarrow \mathsf{MME.Setup}(K, \mathbf{M})$
Return EDS

Alg$\mathsf{RSE}_\Omega.\mathsf{Token}(K, (a, b), c)$
$S \leftarrow \mathsf{Cover}((a, b), c)$
$T \leftarrow \varnothing$
For $(i, j) \in S$ do:
  $T \leftarrow T \cup \{\mathsf{MME.Token}(K, (i, j))\}$
Return T

Alg$\mathsf{RSE}_\Omega.\mathsf{Search}(T, \mathsf{EDS})$
$C \leftarrow \varnothing$
For $tk \in T$ do:
  $C \leftarrow C \cup \{\mathsf{MME.Search}(tk, \mathsf{EDS})\}$
Return C

Alg$\mathsf{RSE}_\Omega.\mathsf{Dec}(K, (a, b), C)$
$D \leftarrow \varnothing$
For $c \in C$ do:
  $((a_1, D_1), ..., (a_x, D_x)) \leftarrow \mathsf{MME.Dec}(K, c)$
  For $i = 1...x$ do:
    If $a \leq a_i \leq b$ then:
      $D \leftarrow D \cup \{D_i\}$
Return D

Alg$\mathsf{RSE}_\Omega.\mathsf{Eval}((a, b), \{(a_1, D_1), ..., (a_x, D_x)\})$
$D \leftarrow \varnothing$
For $i = 1...x$ do:
  If $a \leq a_i \leq b$ then:
    $D \leftarrow D \cup \{D_i\}$
Return D

Fig. 5: Algorithm definitions for RSE_Ω

RSE_Ω **Algorithms** RSE_Ω is an example of a RSE scheme and is built upon any MME scheme and defined in Figure 5. RSE_Ω.Setup takes in a secret key K and database DB. It then for all nodes in the binary tree computes their leftmost and rightmost descendant node. These nodes are then all stored under a multi-map the node index as their label. The multi-map is then encrypted using MME.Setup. The other functions likewise use their MME counterparts and converts them into sets.

C C-cover Algorithm and Proof

C.1 Definitions

Our algorithms are defined in terms of infinite binary trees. Figure 6 shows a subsection of this tree. **Nodes** are referred to in the notation **(h, p)** where h is the **height** of the nodes which is defined as the layer the node is on, counted from the bottom of the tree and p is the **position** of the node in its layer, counted from the left.

Let $\mathscr{S} := \{(i, j) : i, j \in \mathbb{N}\}$ represent the set of all nodes in the tree. A **leaf** is defined as a node with no nodes connected below it (ie. At the bottom most layer of the tree) and has a height of 0. Let $\mathscr{L} = \{(0, j) : j \in \mathbb{N}\}$ represent the set of all leaves.

We refer to node x as a **descendant** of node y and y as a **predecessor** of x if y is on the shortest path between x and the root node. We define function $desc(n)$ s.t. $\forall n \in \mathscr{S}$:

$$desc(n) = \{n' \in \mathscr{L} : n' \text{ is a descendant of } n\}$$

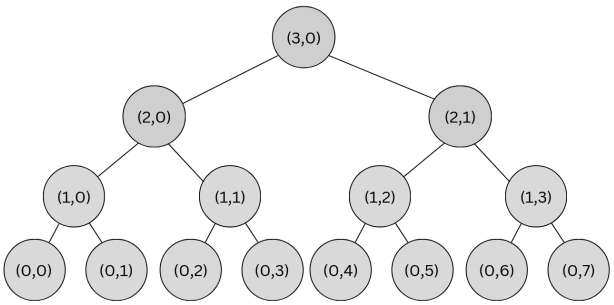

Fig. 6: Infinite Binary Tree used for C-cover algorithm

Then we say that $C \subseteq \mathscr{S}$ is a cover of (a, b) where $a, b \in \mathbb{N}$ and $a \leq b$ if :

$$\bigcup_{n \in C} desc(n) \supseteq \{(0, a), (0, a + 1), \ldots, (0, b)\}$$

We say that C is an optimal c-cover of (a, b) if :

- C covers (a, b)
- $|C| \leq c$
- $\forall n, n' \in C$ and $n \neq n', desc(n) \bigcap desc(n') = \emptyset$ (non-overlapping)
- for all other covers C' of (a, b)

$$\left| \bigcup_{n \in C'} desc(n) \right| \geq \left| \bigcup_{n \in C} desc(n) \right|$$

Note that for some a, b, c there can exist more than 1 optimal c-cover.

We say that **Alg** is a Non-universal Algorithm if on any input (a, b, c) it always returns some optimal c-cover of (a, b).

Given a range (a, b), let $C = \mathsf{Cover}(a, b, c)$. We define the hinge node l as a leaf in cover C such that such that C is made up of 2 sub-covers which are covers of ranges $(k, l - 1)$ and (l, m), with nodes of increasing and decreasing heights respectively. Note that $k \leq a$ and $m \geq b$.

C.2 C-cover Algorithm and Proof

The Non-universal Algorithm is based on finding the hinge node, $(1, 2^s \cdot i)$, a leaf node which should be able to separate the cover into 2 sides with nodes of descending heights as they cover moves further from the hinge node. In the following section, we will define our algorithm, and at the same time prove its optimality.

A hinge value can be found using the algorithm below. Intuitively, we first find the total number of leaves in the range (a, b), which we express as the size, s. We then find the leaf in the range (a, b) which has the largest power of 2. The algorithms for the calculation of the hinge node are shown

below.

For a range (a, b) where $r = b - a + 1$, $s = \lfloor \log_2 r \rfloor$. We then find x s.t $a \leq 2^s \cdot i \leq b$. Additionally, if there are i, i' satisfying this condition, i is the even one. The hinge node is $(0, 2^s \cdot i)$ and $x = 2^s \cdot i$ is the hinge value.

We can calculate the hinge node using $\mathsf{HingeValue}(a, b)$

Alg$\mathsf{HingeValue}(a, b)$
$s \leftarrow \lfloor \log_2(b - a + 1) \rfloor$
$S \leftarrow \{j \in \mathbb{N} : a \leq j \cdot 2^s \leq b\}$
If $S = \{x\}$:
  Return $x \cdot 2^s$
Else:
  Let $y \in S$ be the even integer
Return $y \cdot 2^s$

We can show that for all ranges (a, b), there exists an optimal cover with this hinge node "at the base" of the node of greatest height.

Having calculated the hinge value, we use 2 algorithms to compute the left and right covers. We generate the left and right subcovers using LSubCover and RSubCover respectively, where LSubCover returns the c-cover for $[a, x)$ and RSubCover returns the c-cover for $[x, b]$ provided x is hinge value.

Alg$\mathsf{LSubCover}(a, x, c)$
$s \leftarrow \lfloor \log_2(x - 1 - a + 1) \rfloor$
If $2^s = x - a$:
  $C \leftarrow \{(s, \frac{x}{2^s} - 1)\}$
Else if $c = 1$:
  $C \leftarrow \{(s + 1, \frac{x}{2^{s+1}} - 1)\}$
Else:
  $C \leftarrow \mathsf{LSubCover}(a, x - 2^s, c - 1) \cup \{(s, \frac{x}{2^s} - 1)\}$
Return C

Alg$\mathsf{RSubCover}(x, b, c)$
$s \leftarrow \lfloor \log_2(b - x + 1) \rfloor$
If $2^s = b - x + 1$:
  $C \leftarrow \{s, \frac{x}{2^s}\}$
Else if $c = 1$:
  $C \leftarrow \{(s + 1, \frac{x}{2^{s+1}})\}$
Else:
  $C \leftarrow \{(s, \frac{x}{2^s})\} \cup \mathsf{RSubCover}(x + 2^s, b, c - 1)$
Return C

Now, we combine the above algorithms with a final cover algorithm Cover. We do this by cycling through all possible distributions of c (for $c \geq 2$) between the left and right subcovers, and finding the distribution that gives the most optimal cover.

Alg$\mathsf{Cover}(a, b, c)$
$n \leftarrow \mathsf{HingeValue}(a, b)$
If $n = a$:
  $C \leftarrow \mathsf{RSubCover}(a, b, c)$
Else:
  For $i = 1 \ldots c - 1$ do:
    $C_i \leftarrow \mathsf{LSubCover}(a, n, i) \cup \mathsf{RSubCover}(n, b, c - i)$
Let $j = \underset{i=1\ldots c-1}{\arg\min} \sum_{(x,y) \in C_i} 2^x$
$C \leftarrow C_j$
Return C

Lemma 1: Given a range (a, b) where $a < b$, there exists an optimal c-cover C where either for some $n \in C$, the leftmost leaf descendant of n is the hinge node of (a, b).

Proof: We can show this via a proof by contradiction. Suppose there exists an optimal cover C_1 where the hinge node and the 2 next to it are covered by the same cover. Based on our definition of a hinge node, it is the foot of the largest triangle that can fit in a range (a, b). Thus, any cover that covers the hinge node and the 2 next to it will always overcover the range (a, b). Let the amount of overcover (the number of extra leaves) be e.

Using our algorithm and the notion of a hinge node, we can always generate a c-cover C_2 for all $c \geq 2$ with overcover e', such that $e' \leq e$. This can be shown by first considering the case where $c = 1$, where our algorithm is redundant and the hinge node definitely needs to be covered. For all other cases, our algorithm returns a cover C_2 that is minimally as good as C_1. Thus, C_1 cannot possibly be the only optimal cover, as our algorithm can return a C_2 which is as good as, or better than, C_1.∎

Lemma 2: Given a range $(2^s \cdot i, b)$ and cover size 1, let R_1 be the output of $\mathsf{RSubCover}(2^s \cdot i, b, 1)$, then R_1 is an right-optimal 1-cover of the range $(2^s \cdot i, b)$.

Proof: Our algorithm will return the smallest cover that covers the range. Notice that if it is any smaller, it will not cover the range fully and will not meet the requirements of a cover. ∎

We can show that algorithms RSubCover and LSubCover algorithms are optimal via induction. We will show optimality for the RSubCover algorithm and the proof for the LSubCover algorithm is analogous. We say a subcover is right-optimal if $(2^s \cdot i, b)$ is the leftmost leaf descendant, and the overhead on the right is minimal.

Lemma 3: Given a range $(2^s \cdot i, b)$ and cover size c, let R be the output of $\mathsf{RSubCover}(2^s \cdot i, b, c)$, then R is an right-optimal 1-cover of the range $(2^s \cdot i, b)$.

Proof: Consider the case where $c = 2$. Notice that in order to minimise overcover, the first cover should always cover the largest range that does not exceed (x, b). Let this cover cover the range (x, z), where $z = 2^t \cdot j$. Then notice that the range we have left to cover is (z, b), where $b - x + 1 < 2^t$. This is very similar to our original range (x, b), where $b - x + 1 < 2^s$, and we can thus find a right-optimal 2-cover. This same process can be repeated to find a right-optimal n-cover given a right-optimal (n-1)-cover. ∎

Theorem: Given a range (a, b) and a cover size c, let $C = \mathsf{Cover}(a, b, c)$, then C is an optimal c-cover of the range (a, b).

Proof: Using Lemma 3, we have now proven that RSubCover returns a right-optimal c-cover for right side of the hinge node. Similarly, we can prove that LSubCover returns a left-optimal c-cover for the left side of the hinge node.

If the hinge node of (a, b) is $(0, a)$, then our algorithm is optimal since RSubCover is optimal. A similar argument can be made for if the hinge node is $(0, b)$, since LSubCover is optimal as well.

If the hinge node lies outside the range (a, b), this implies that all nodes are in strictly descending order in just one direction. Regardless of whether the hinge node lies to the left of right of the range (a, b), our algorithm is optimal since both LSubCover and RSubCover are optimal.

Else, there exists an optimal c-cover C where for some $n \leftarrow C$, the leftmost leaf descendant of n is the hinge node of (a, b) (Lemma 1). Notice that lines 5 and 6 of our algorithm cycles through all possible distributions of c between the left and right subcovers, and line 7 chooses the most optimal one. We have thus proven that C is an optimal c-cover of the range (a, b). ∎

D Provable Security

D.1 Static SE game

Game $\mathrm{G}^{\mathrm{ind}}_{\mathsf{SE}}(A)$
$K_{\mathsf{SE}} \leftarrow\$ \mathsf{SE.KS}$
$b \leftarrow\$ \{0,1\}$
$(St, (M_1, ..., M_n)) \leftarrow A$
If $b = 1$:
  For i in $1, ..., n$:
    $C_i \leftarrow\$ \mathsf{SE.Enc}(K_{\mathsf{SE}}, M_i)$
Else:
  For i in $1, ..., n$:
    $C_i \leftarrow\$ \{0,1\}^{|\mathsf{SE.cl}(M)|}$
$b \leftarrow A(St, C_1, ..., C_n)$
Return $b = b'$

Here we have a static IND-$ game. The game randomly generates a key and randomly selects bit 1 or 0. Adversary A then inputs however many messages M it wishes into the game. It also gives state, St, which it uses to store any relevant information. The game then performs SE.Enc or generates a random string for each message respectively. This is returned to the adversary who guesses the "world" the game is in.

D.2 Dynamic PRF game

We now proceed to define a dynamic indistinguishability game for F as shown below.

Game $\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(A)$
$K_{\mathsf{F}} \leftarrow\$ \mathsf{F.KS}$
$b \leftarrow\$ \{0,1\}$
$b' \leftarrow\$ A^{\mathrm{F_N}}$
Return $b = b'$

Oracle $\mathrm{F_N}(x)$
If $b = 1$:
  Return $\mathsf{F.Ev}(K_{\mathsf{F}}, x)$
Else:
  If $\mathbf{T}[x] = \perp$:
    $\mathbf{T}[x] \leftarrow\$ \mathsf{F.IS}$
  Return $\mathbf{T}[x]$

In the PRF game, the real world using $\mathsf{F.Ev}$ to hash a given message and the output is deterministic based on the inputs. The ideal world is simulates this by using a pseudo-random function. For every input x to the oracle, an output is selected from the input set $\mathsf{F.IS}$ at random. The oracle stores the previous values of x and their corresponding outputs, so that if an identical input x is re-submitted, the same output is returned. The advantage of adversary A playing the above game is $\mathrm{Adv}^{\mathrm{prf}}_{\mathsf{F}}(A)$.

D.3 MME_π Leakage and Simulator Functions

We define MME_π leakage and simulator functions as below.

Alg $\mathcal{L}_\pi(\mathbf{M}, (l_1, ..., l_n))$
$total \leftarrow \sum_{i=1}^{k}(\#(\mathbf{M}[i]))$
For $i = 1...n$ do:
  $vol_i \leftarrow \#(\mathbf{M}[l_i])$
  $ep_i \leftarrow min(\{j : 1 \leq j \leq i, l_i = l_j\})$
Return $(total, (vol_1, ..., vol_n), (ep_1, ..., ep_n))$

Alg $\mathcal{S}_\pi(total, (vol_1, ..., vol_n), (ep_1, ..., ep_n))$
For $i = 1...n$:
  If $ep_i = i$:
    $tk_i \leftarrow\$ \{0,1\}^{\mathsf{F.ol}}$
    For $j = 1...vol_i$:
      $\mathsf{EDS}[\mathsf{F.Ev}(tk_i, j)] \leftarrow\$ \{0,1\}^{\mathsf{vLen}}$
  Else:
    $tk_i \leftarrow tk_{ep_i}$
While $|\mathsf{EDS}| < total$:
  $l \leftarrow\$ \{0,1\}^{\mathsf{F.ol}}$
  $\mathsf{EDS}[l] \leftarrow\$ \{0,1\}^{\mathsf{vLen}}$
Return $(\mathsf{EDS}, (tk_1, ..., tk_n))$

$\mathcal{L}_\pi$ returns the leakage, which comprises of 3 main components. $total$, the total number of documents in the database, which is calculated by counting the number of documents in each database entry. $(vol_1...vol_n)$ is the query volume, the number of documents each query returns, which is counted for each query. $(ep_1...ep_n)$ is the equality pattern, which notes down any repeated queries by associating them with the previous most query that was equal.

In $\mathcal{S}_\pi$, equality pattern of queries, $(ep_1...ep_n)$, is used to randomly generate tokens of length $\mathsf{F.ol}$ that follow a pseudo-random function. The documents returned for each query is randomly generated for each unique token to the length of vLen. At the same time, the algorithm counts how much of the total number of documents, $total$, has been used up, and randomly generates the needed tokens and documents until $|\mathsf{EDS}| = total$. The simulator returns EDS and a tuple of tokens $(tk_1...tk_n)$, that follow the equality patterns, query volume and total number of documents.

D.4 RSE_Ω Leakage and Simulator Functions

We define RSE_Ω leakage and simulator functions as below.

Alg $\mathcal{L}_\Omega(\mathsf{DB}, (q_1, ..., q_k))$
$\mathbf{l} \leftarrow (l : l \in \mathsf{Cover}(q_i, c))$
$\{(a_1, D_1), ..., (a_x, D_x)\} \leftarrow \mathsf{DB}$
For $i = 0... \log_2(\mathsf{RSE}_\Omega.\mathsf{KS})$ do:
  For $j = 0...2^i - 1$ do:
    $\mathsf{up} \leftarrow 2^{\log_2(\mathsf{RSE}_\Omega.\mathsf{KS})-i} \cdot (j+1) - 1$
    $\mathsf{down} \leftarrow 2^{\log_2(\mathsf{RSE}_\Omega.\mathsf{KS})-i} \cdot j$
    $\mathsf{node} \leftarrow (\log_2(\mathsf{RSE}_\Omega.\mathsf{KS}) - i, j)$
    $\mathsf{docs} \leftarrow ((a_i, D_i) : i \in \{1...x\}, \mathsf{down} \leq a_i \leq \mathsf{up})$
    $\mathbf{M}[\mathsf{node}] \leftarrow \mathsf{docs}$
$(total, (vol_1...vol_n), (ep_1...ep_n)) \leftarrow \mathcal{L}_{\mathsf{MME}}(\mathbf{M}, \mathbf{l})$
Return lk

Alg $\mathcal{S}_\Omega(total, (vol_1...vol_n), (ep_1...ep_n))$
$(\mathsf{EDS}, (T_1...T_n)) \leftarrow \mathcal{S}_{\mathsf{MME}}(total, (vol_1...vol_n), (ep_1...ep_n))$
Return $(\mathsf{EDS}, (T_1...T_n))$

$\mathcal{L}_\Omega$ uses the Cover algorithm to generate labelled sub-queries for all $q \in (q_1...q_y)$. The label of each query is a tuple in the form of (a, b), where the integer value a refers to which query the sub-query was generated from, and value of b refers to in which order the sub-queries of a select query was generated. In addition, DB is converted to a table $\mathbf{M}$, to imitate the data structure used in MME. The sub-queries and $\mathbf{M}$ are fed into $\mathcal{L}_\Omega$ as they are no different from the expected input.

$\mathcal{S}_\Omega$ is similar to $\mathcal{S}_\Omega$, except it returns a set of tuples, $(T_1...T_n)$, with T_i containing the tokens corresponding to the query q_i.

D.5 Leakage Games

We define the following leakage games for MME_π and RSE_Ω

Game $\mathrm{G}^{\mathrm{ss}}_{\mathsf{RSE},\mathcal{L},\mathcal{S}}(A)$	**Game** $\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME},\mathcal{L}_{\mathsf{MME}},\mathcal{S}_{\mathsf{MME}}}(B)$
$K \leftarrow\$ \mathsf{RSE}_\Omega.\mathsf{KS}$	$K \leftarrow\$ \mathsf{MME}.\mathsf{KS}$
$b \leftarrow\$ \{0,1\}$	$b \leftarrow\$ \{0,1\}$
$(St, \mathsf{DB}, (q_1, ..., q_n)) \leftarrow\$ A_1$	$(St, \mathbf{M}, (l_1, ..., l_n)) \leftarrow\$ B_1$
If $\mathsf{DB} \notin \mathsf{allDB}$:	If $M \notin \mathsf{allM}$:
Return false	Return false
If $b = 1$:	If $b = 1$:
$\mathsf{EDS} \leftarrow\$ \mathsf{RSE}_\Omega.\mathsf{Setup}(K, \mathsf{DB})$	$\mathsf{EDS} \leftarrow\$ \mathsf{MME}.\mathsf{Setup}(K, \mathbf{M})$
For i in $1...n$ do:	For i in $1...n$ do:
$T_i \leftarrow \mathsf{RSE}_\Omega.\mathsf{Token}(K, q_i, c)$	$tk_i \leftarrow \mathsf{MME}.\mathsf{Token}(K, l_i)$
Else:	Else:
$lk \leftarrow \mathcal{L}_\Omega(\mathsf{DB}, (q_1, ..., q_n))$	$lk \leftarrow \mathcal{L}_{\mathsf{MME}}(\mathbf{M}, (l_1, ..., l_n))$
$(\mathsf{EDS}, (T_1, ..., T_n)) \leftarrow \mathcal{S}_\Omega(lk)$	$(\mathsf{EDS}, (tk_1...tk_n)) \leftarrow \mathcal{S}_{\mathsf{MME}}(lk)$
$b' \leftarrow\$ A_2(St, \mathsf{EDS}, (T_1, ..., T_n))$	$b' \leftarrow\$ B_2(St, \mathsf{EDS}, (tk_1, ..., tk_n))$
Return $b = b'$	Return $b = b'$

In the above games, the adversary provides a St for it to store information that it needs to decide if it is in the "real" or "ideal" world. The adversary also provides a database, together with n queries.

We will explain how the MME game works and this intuition can be used for RSE. In the "real" world, EDS generated by MME.Setup and the respective tokens are generated for all the queries using MME.Token. In the "ideal" world, an ideal EDS and corresponding tokens are generated by a simulator $\mathcal{S}_{\mathsf{MME}}$ which takes lk as defined by $\mathcal{L}_{\mathsf{MME}}(\mathsf{DB}, (q_1...q_n))$. The security of the scheme requires that only certain properties of the database and queries are leaked, as defined by lk. As such, if the adversary can distinguish between an EDS and tokens generated by the simulator and ones generated in the "real" world, there is some other leakage that is excluded in the leakage function. More intuitively, if the adversary cannot distinguish between them, it can be concluded that all the leakage is quantified by the leakage function.

The advantage of Adversary A is $\mathrm{Adv}^{\mathrm{ss}}_{\mathsf{MME},\mathcal{L}_{\mathsf{MME}},\mathcal{S}_{\mathsf{MME}}} = 2\Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME},\mathcal{L}_{\mathsf{MME}},\mathcal{S}_{\mathsf{MME}}}(A)] - 1$. Intuitively, the purpose of $\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME},\mathcal{L}_{\mathsf{MME}},\mathcal{S}_{\mathsf{MME}}}$ is to show whether or not the leakage functions used cover all possible leakage of EDS and all tk. If so, $\mathrm{Adv}^{\mathrm{ss}}_{\mathsf{MME},\mathcal{L}_{\mathsf{MME}},\mathcal{S}_{\mathsf{MME}}} = 0$. The advantage of Adversary B is $\mathrm{Adv}^{\mathrm{ss}}_{\mathsf{RSE},\mathcal{L}_{\mathsf{RSE}},\mathcal{S}_{\mathsf{RSE}}} = 2\Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{RSE},\mathcal{L},\mathcal{S}}(B)] - 1$, and a similar logic applies.

D.6 Proof of MME_π Security

For reference, these are the games played by B, C and D respectively:

Game $\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME},\mathcal{L}_\pi,\mathcal{S}_\pi}(A)$
$(A_1, A_2) \leftarrow A$
$K \leftarrow\$ \mathsf{MME.KS}$
$b \leftarrow\$ \{0,1\}$
$(St, \mathbf{M}, (l_1, ..., l_n)) \leftarrow\$ A_1$
If $\mathbf{M} \notin \mathsf{allM}$:
  Return false
If $b = 1$:
  $\mathsf{EDS} \leftarrow\$ \mathsf{MME.Setup}(K, \mathbf{M})$
  For i in $1...n$:
    $tk_i \leftarrow \mathsf{MME.Token}(K, l_i)$
Else:
  $lk \leftarrow \mathcal{L}_{\mathsf{MME}}(\mathbf{M}, (l_1, ..., l_n))$
  $(\mathsf{EDS}, (tk_1...tk_n)) \leftarrow \mathcal{S}_{\mathsf{MME}}(lk)$
$b' \leftarrow\$ A_2(St, \mathsf{EDS}, (tk_1, ..., tk_n))$

Game $\mathrm{G}^{\mathrm{ind}}_{\mathsf{SE}}(B)$
$K_{\mathsf{SE}} \leftarrow\$ \mathsf{SE.KS}$
$b \leftarrow\$ \{0,1\}$
$b' \leftarrow A^{\mathrm{Enc}}$
Return $b = b'$

Oracle $\mathrm{Enc}(x)$
$c_1 \leftarrow\$ \mathsf{SE.Enc}(K_{\mathsf{SE}}, x)$
$c_0 \leftarrow\$ \{0,1\}^{\mathsf{SE.cl}(|x|)}$
Return c_b

Game $\boxed{\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(C)}$ $\boxed{\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(D)}$
$K_{\mathsf{F}} \leftarrow\$ \mathsf{F.KS}$
$b \leftarrow\$ \{0,1\}$
$b' \leftarrow\$ \boxed{C^{\mathrm{Fn}}}\ \boxed{D^{\mathrm{Fn}}}$
Return $b = b'$

Oracle $\mathrm{Fn}(x)$
If $b = 1$:
  Return $\mathsf{F.Ev}(K_{\mathsf{F}}, x)$
Else:
  If $\mathbf{T}[x] = \perp$:
    $\mathbf{T}[x] \leftarrow\$ \{0,1\}^{\mathsf{F.ol}}$
  Return $\mathbf{T}[x]$

Proof : We first define adversaries B, C and D, which are built from adversary A, as shown below. Adversaries B, C and D, try to win their games by using adversary A to make guesses for them. Hence, they need to "hijack" Adversary A. Note that $A = (A_1, A_2)$.

Adversary B^{Enc}
$(St, \mathbf{M}, (l_1, ..., l_n)) \leftarrow\$ A_1$
$K_{\mathsf{F}} \leftarrow\$ \mathsf{F.KS}$
If $\mathbf{M} \notin \mathsf{allM}$:
  Return false
For all $\mathbf{M}[l]$ in $\mathbf{M}$:
  $tk \leftarrow \mathsf{F.Ev}(K_{\mathsf{F}}, l)$
  $(v_1, ..., v_a) \leftarrow \mathbf{M}[l]$
  For $j = 1...a$:
    $\mathsf{EDS}[\mathsf{F.Ev}(tk, j)] \leftarrow \mathrm{Enc}(v_j)$
For $i = 1...n$:
  $tk_i \leftarrow \mathsf{F.Ev}(K_{\mathsf{F}}, l_i)$
$b' \leftarrow\$ A_2(St, \mathsf{EDS}, (tk_1, ..., tk_n))$
Return b'

Adversary C^{Fn}
$(St, \mathbf{M}, (l_1, ..., l_n)) \leftarrow\$ A_1$
If $\mathbf{M} \notin \mathsf{allM}$:
  Return false
For all $\mathbf{M}[l]$ in $\mathbf{M}$:
  $tk \leftarrow\$ \mathrm{Fn}(l)$
  $(v_1, ..., v_a) \leftarrow \mathbf{M}[l]$
  For j =$1...a$:
    $\mathsf{EDS}[\mathsf{F.Ev}(tk, j)] \leftarrow\$ \{1,0\}^{\mathsf{vLen}}$
For $i = 1...n$:
  $tk_i \leftarrow \mathrm{Fn}(l_i)$
$b' \leftarrow\$ A_2(St, \mathsf{EDS}, (tk_1, ..., tk_n))$
Return b'

In the above table, B essentially runs MME_π.Setup and token generation is real. However, since it is trying to distinguish between real and fake encryption, it uses the Enc oracle from $\mathrm{G}^{\mathrm{ind}}_{\mathsf{SE}}(B)$ to perform encryption on values. B then feeds EDS and tokens to A_2 and uses their guess. C does something similiar, but now performs a modified version of $\mathcal{L}_\pi$. Encryption on values is always faked, while tokens are generated using the Fn oracle. Both adversaries convert the input $\mathbf{M}$ and ls into EDS and tks, before feeding them back into A and taking A's guess.

Adversary D^{FN}

$(St, \mathbf{M}, (l_1, ..., l_n)) \leftarrow\$ A_1$
$x \leftarrow |\{l_1, ..., l_n\}|$
$p \leftarrow\$ \{1, ..., m - x\}$
$counter \leftarrow 1$
For all $\mathbf{M}[l]$ in $\mathbf{M}$:
  $(v_1, ..., v_a) \leftarrow \mathbf{M}[l]$
  $tk \leftarrow\$ \{0,1\}^{\mathsf{lLen}}$
  For all $i = 1...n$ where $l = l_i$:
    $tk_i \leftarrow tk$
  For $j = 1...a$:
    If $l \in \{l_1, ..., l_n\}$ or $counter > p$:
      $\mathsf{EDS}[\mathsf{F.Ev}(tk, j)] \leftarrow\$ \{0,1\}^{\mathsf{vLen}}$
    Else if $counter = p$:
      $\mathsf{EDS}[\mathrm{FN}(j)] \leftarrow\$ \{0,1\}^{\mathsf{vLen}}$
      $counter \leftarrow counter + 1$
    Else:
      $\mathsf{EDS}[\{1,0\}^{\mathsf{lLen}}] \leftarrow\$ \{0,1\}^{\mathsf{vLen}}$
      $counter \leftarrow counter + 1$
$b' \leftarrow\$ A_2(St, \mathsf{EDS}, (tk_1, ..., tk_n))$
Return b'

We now define a series of hybrids which we will explain and use later on.

Game G_0

$K \leftarrow\$ \mathsf{MME.KS}$
$(St, \mathbf{M}, (l_1, ..., l_n)) \leftarrow\$ A_1$
If $\mathbf{M} \notin \mathsf{allM}$ or :
  Return false
$total \leftarrow \sum_{i=1}^{k}(\#(\mathbf{M}[i]))$
For $i = 1...n$ do:
  $vol_i \leftarrow \#(\mathbf{M}[l_i])$
  $ep_i \leftarrow min(\{j : 1 \leq j \leq i, l_i = l_j\})$
For $i = 1...n$:
  If $ep_i = i$:
    $tk_i \leftarrow\$ \{0,1\}^{\mathsf{F.ol}}$
    For $j = 1...vol_i$:
      $\mathsf{EDS}[\mathsf{F.Ev}(tk_i, j)] \leftarrow\$ \{0,1\}^{\mathsf{vLen}}$
  Else:
    $tk_i \leftarrow tk_{ep_i}$
While $|\mathsf{EDS}| < total$:
  $l \leftarrow\$ \{0,1\}^{\mathsf{F.ol}}$
  $\mathsf{EDS}[l] \leftarrow\$ \{0,1\}^{\mathsf{vLen}}$
$b' \leftarrow\$ A_2(St, \mathsf{EDS}, (tk_1, ..., tk_n))$
Return $b = b'$

Game G_1

$(K_{\mathsf{SE}}, K_{\mathsf{F}}) \leftarrow\$ \mathsf{MME.KS}$
$(St, \mathbf{M}, (l_1, ..., l_n)) \leftarrow\$ A_1$
If $\mathbf{M} \notin \mathsf{allM}$:
  Return false
For all $\mathbf{M}[l]$ in $\mathbf{M}$:
  $tk \leftarrow \mathsf{F.Ev}(K_{\mathsf{F}}, l)$
  For all $i = 1...n$ where $l_i = l$:
    $tk_i \leftarrow tk$
  $(v_1, ..., v_a) \leftarrow \mathbf{M}[l]$
  For $j = 1...a$:
    $\mathsf{EDS}[\mathsf{F.Ev}(tk, j)] \leftarrow \mathsf{SE.Enc}(K_{\mathsf{SE}}, v_j)$
$b' \leftarrow\$ A_2(St, \mathsf{EDS}, (tk_1, ..., tk_n))$
Return $b' = 1$

Game $\boxed{G_2}$, $\boxed{G_3}$

$\boxed{K_{\mathsf{F}} \leftarrow\$ \mathsf{F.KS}}$
$(St, \mathbf{M}, (l_1, ..., l_n)) \leftarrow\$ A_1$
If $\mathbf{M} \notin \mathsf{allM}$:
  Return false
For all $\mathbf{M}[l]$ in $\mathbf{M}$:
  $\boxed{tk \leftarrow \mathsf{F.Ev}(K_{\mathsf{F}}, l)}$ $\boxed{tk \leftarrow\$ \{1,0\}^{\mathsf{lLen}}}$
  For all $i = 1...n$ where $l_i = l$:
    $tk_i \leftarrow tk$
  $(v_1, ..., v_a) \leftarrow \mathbf{M}[l]$
  For j =1...a do:
    $\mathsf{EDS}[\mathsf{F.Ev}(tk, j)] \leftarrow\$ \{1,0\}^{\mathsf{vLen}}$
$b' \leftarrow\$ A_2(St, \mathsf{EDS}, (tk_1, ..., tk_n))$
Return $b' = 1$

Game G_4

$(St, \mathbf{M}, (l_1, ..., l_n)) \leftarrow\$ A_1$
If $\mathbf{M} \notin \mathsf{allM}$:
  Return false
For all $\mathbf{M}[l]$ in $\mathbf{M}$:
  $tk \leftarrow\$ \{1,0\}^{\mathsf{lLen}}$
  For all $i = 1...n$ where $l_i = l$:
    $tk_i \leftarrow tk$
  $(v_1, ..., v_a) \leftarrow \mathbf{M}[l]$
  For $j = 1...a$:
    If $l \in \{l_1, ..., l_n\}$:
      $\mathsf{EDS}[\mathsf{F.Ev}(tk, j)] \leftarrow\$ \{1,0\}^{\mathsf{vLen}}$
    Else: $\mathsf{EDS}[\{1,0\}^{\mathsf{lLen}}] \leftarrow\$ \{1,0\}^{\mathsf{vLen}}$
$b' \leftarrow\$ A_2(St, \mathsf{EDS}, (tk_1, ..., tk_n))$
Return $b' = 1$

Game H_p

$(St, \mathbf{M}, (l_1, ..., l_n)) \leftarrow\$ A_1$
If $\mathbf{M} \notin \mathsf{allM}$:
  Return false
$counter \leftarrow 1$
For all $\mathbf{M}[l]$ in $\mathbf{M}$:
  $(v_1, ..., v_a) \leftarrow \mathbf{M}[l]$
  $tk \leftarrow\$ \{0,1\}^{\mathsf{lLen}}$
  For all $i = 1...n$ where $l = l_i$:
    $tk_i \leftarrow tk$
  For $j = 1...a$:
    If $l \in \{l_1, ..., l_n\}$ or $counter > p$:
      $\mathsf{EDS}[\mathsf{F.Ev}(tk, j)] \leftarrow\$ \{1,0\}^{\mathsf{vLen}}$
    Else:
      $\mathsf{EDS}[\{1,0\}^{\mathsf{lLen}}] \leftarrow\$ \{1,0\}^{\mathsf{vLen}}$
      $counter \leftarrow counter + 1$
$b' \leftarrow\$ A_2(St, \mathsf{EDS}, (tk_1, ..., tk_n))$
Return $b' = 1$

Using the above hybrids, we can show the following equations.

$$\mathrm{Adv}^{\mathrm{ss}}_{\mathsf{MME},\mathcal{L}_\pi,\mathcal{S}_\pi}(A) = \Pr[\mathrm{G}_1] - \Pr[\mathrm{G}_0] \tag{3}$$

$$\mathrm{Adv}^{\mathrm{ind}}_{\mathsf{SE}}(B) = \Pr[\mathrm{G}_1] - \Pr[\mathrm{G}_2] \tag{4}$$

$$\mathrm{Adv}^{\mathrm{prf}}_{\mathsf{F}}(C) = \Pr[\mathrm{G}_2] - \Pr[\mathrm{G}_3] \tag{5}$$

$$(m - x) \cdot \mathrm{Adv}^{\mathrm{prf}}_{\mathsf{F}}(D) = \Pr[\mathrm{G}_3] - \Pr[\mathrm{G}_0] \tag{6}$$

Combining equations (1), (2), (3) and (4) we can see that:

$$\begin{aligned}\mathrm{Adv}^{\mathrm{ss}}_{\mathsf{MME},\mathcal{L}_\pi,\mathcal{S}_\pi}(A) &= \Pr[\mathrm{G}_1] - \Pr[\mathrm{G}_0] \\ &= \Pr[\mathrm{G}_1] - \Pr[\mathrm{G}_2] + \Pr[\mathrm{G}_2] - \Pr[\mathrm{G}_3] + \Pr[\mathrm{G}_3] - \Pr[\mathrm{G}_0] \\ &= \mathrm{Adv}^{\mathrm{ind}}_{\mathsf{SE}}(B) + \mathrm{Adv}^{\mathrm{prf}}_{\mathsf{F}}(C) + (m - x) \cdot \mathrm{Adv}^{\mathrm{prf}}_{\mathsf{F}}(D)\end{aligned}$$

To show equation (1), first let b and b' be the random variables in $\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME},\mathcal{L}_\pi,\mathcal{S}_\pi}(A)$. Notice that when $b = 1$, the game runs MME_π.Setup and MME_π.Token to generate EDS and $tk_1, ..., tk_n$ respectively. Inlining these algorithms into $\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME},\mathcal{L}_\pi,\mathcal{S}_\pi}$ gives us G_1. When $b = 0$, $\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME},\mathcal{L}_\pi,\mathcal{S}_\pi}(A)$ runs $\mathcal{L}_\pi$ and

$\mathcal{S}_\pi$ to fake EDS and $tk_1, ..., tk_n$ and this gives G_0.

$$\begin{aligned}\mathrm{Adv}^{\mathrm{ss}}_{\mathsf{MME},\mathcal{L}_\pi,\mathcal{S}_\pi}(A) &= \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME},\mathcal{L}_\pi,\mathcal{S}_\pi}(A) : b' = 1 | b = 1] - \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME},\mathcal{L}_\pi,\mathcal{S}_\pi}(A) : b' = 1 | b = 0] \\ &= \Pr[\mathrm{G}_1] - \Pr[\mathrm{G}_0]\end{aligned}$$

To show equation (2), first let b and b' be the random variables in $\mathrm{G}^{\mathrm{ind}}_{\mathsf{SE}}(B)$. Observe that G_1 and G_2 are $\mathrm{G}^{\mathrm{ind}}_{\mathsf{SE}}(B)$ when $b = 1$ and $b = 0$ respectively. Thus,

$$\begin{aligned}\mathrm{Adv}^{\mathrm{ind}}_{\mathsf{SE}}(B) &= \Pr[\mathrm{G}^{\mathrm{ind}}_{\mathsf{SE}}(B) : b' = 1 | b = 1] - \Pr[\mathrm{G}^{\mathrm{ind}}_{\mathsf{SE}}(B) : b' = 1 | b = 0] \\ &= \Pr[\mathrm{G}_1] - \Pr[\mathrm{G}_2]\end{aligned}$$

To show equation (3), first let b and b' be the random variables in $\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(C)$. Observe that G_2 and G_3 are $\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(C)$ when $b = 1$ and $b = 0$ respectively. Thus,

$$\begin{aligned}\mathrm{Adv}^{\mathrm{prf}}_{\mathsf{F}}(C) &= \Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(C) : b' = 1 | b = 1] - \Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(C) : b' = 1 | b = 0] \\ &= \Pr[\mathrm{G}_2] - \Pr[\mathrm{G}_3]\end{aligned}$$

Notice that G_3 and G_0 fake the EDS labels differently. Hence, adversary D will "bridge" this gap.

To show equation (4), let b and b' be the random variables in $\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(D)$. Since D picks a p to play the $\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}$ game with equal probability from $\{1, ..., m - x\}$,

$$\Pr[p = 1] + \Pr[p = 2] + ... + \Pr[p = m - x] = 1$$

$$\Pr[p = i] = \frac{1}{m - x}$$

It thus follows that,

$$\Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(D)] = \sum_{i=1}^{m-x} \Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(D) | p = i] \Pr[p = i]$$

Hence using this identity,

$$\begin{aligned}\mathrm{Adv}^{\mathrm{prf}}_{\mathsf{F}}(D) &= \Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(D) : b' = 1 | b = 1] - \Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(D) : b' = 1 | b = 0] \\ &= \sum_{i=1}^{m-x} (\Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(D) : b' = 1 | b = 1, p = i] \cdot \Pr[p = i]) \\ &\quad - \sum_{i=1}^{m-x} (\Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(D) : b' = 1 | b = 0, p = i] \cdot \Pr[p = i]) \\ &= \frac{1}{m - x} (\sum_{i=1}^{m-x} (\Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(D) : b' = 1 | b = 1, p = i]) \\ &\quad - \sum_{i=1}^{m-x} (\Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(D) : b' = 1 | b = 0, p = i])) \\ (m - x) \cdot \mathrm{Adv}^{\mathrm{prf}}_{\mathsf{F}}(D) &= \sum_{i=1}^{m-x} (\Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(D) : b' = 1 | b = 1, p = i] - \Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(D) : b' = 1 | b = 0, p = i])\end{aligned}$$

$\mathrm{G}_{\mathsf{F}}^{\mathrm{prf}}(D)$ must ensure that all tokens for queried labels are constructed such that they can still access encrypted values in EDS. Hence for x unique queries, after randomly initializing a token, it must perform $\mathsf{F.Ev}$. For the other $m - x$ labels in $\mathbf{M}$ however, their "sub-tokens" (the keys to EDS) can be generated either via random initialization or in a similiar fashion to queried labels. For these $m - x$ labels in $\mathbf{M}$, $\mathrm{G}_{\mathsf{F}}^{\mathrm{prf}}(D)$ picks a value $p \in \{1, .., m - x\}$ to decide how many labels it will randomly initialize or perform $\mathsf{F.Ev}$.

Let us consider the case where $p = i$. On i labels, "sub-tokens" will be generated randomly using $\{0, 1\}^{\mathsf{lLen}}$, while $\mathsf{F.Ev}$ will be performed on $m - x - i - 1$ labels. "Sub-tokens" will be generated for the last label using $\mathsf{F.Ev}$ when $b = 1$, and randomly when $b = 0$. Thus notice that when $b = 1$, $\mathsf{F.Ev}$ is run $m - i$ times (including queried labels) and randomly initialization for "sub-tokens" is performed for i labels. When $b = 0$, this is $m - x - i - 1$ and $i + 1$ respectively.
Now, consider out hybrids H_0 to H_{m-x}. Notice that in H_0, since $counter$ is always > 0, all labels are generated using $\mathsf{F.Ev}$. Conversely, in H_{m-x}, counter is always $\leq (m - x)$, thus all unqueried labels are generated randomly. Following this logic, deduce that for any hybrid H_n, n unqueried labels are generated randomly.

To link our hybrids to various versions of $\mathrm{G}_{\mathsf{F}}^{\mathrm{prf}}(D)$ when different values of p are chosen, we will consider the number of unqueried labels that are generated randomly. Recall that this can be expressed as p when $b = 0$ and $p + 1$ when $b = 1$. Thus, it can be observed that

$$\Pr[\mathrm{G}_{\mathsf{F}}^{\mathrm{prf}}(D) : b' = 1 | b = 1, p = i] = \Pr[H_{i-1}]$$
$$\Pr[\mathrm{G}_{\mathsf{F}}^{\mathrm{prf}}(D) : b' = 1 | b = 0, p = i] = \Pr[H_i]$$

Consider again H_0 and H_{m-x}. Notice that these extreme cases are equivalent to G_3 and G_4 respectively.
Hence,

$$\Pr[H_0] = \Pr[G_3]$$
$$\Pr[H_{m-x}] = \Pr[G_4]$$

Now, compare hybrids G_0 and G_4. Notice that G_4 finds the volume vol_i of each label l_i, corresponding to the number of values stored under each label. It then randomly generates tokens tk, hashes them with their respective value and assigns a randomly generates encrypted document. Functionally, G_0 does the same thing by running the same algorithm for all $(v_1, ..., v_a)$ under each label l. Next, observe that G_4 ensures that EDS satisfies the equality pattern ep_i, which ensures that a repeated label queried will return the same token. Functionally, G_0 does this by checking for all $i = 1...n$ where $l_i = l$, and assigning the same value of tk. Thus, we conclude that G_4 and G_0 are functionally equivalent.
Hence,

$$\Pr[G_4] = \Pr[G_0]$$

Hence,

$$
\begin{aligned}
(m-x)\cdot \mathrm{Adv}^{\mathrm{prf}}_{\mathsf{F}}(D) &= \sum_{i=1}^{m-x}(\Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(D):b'=1|b=1,p=i]-\Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(D):b'=1|b=0,p=i]) \\
&= \sum_{i=1}^{m-x}(\Pr[H_{i-1}]-\Pr[H_i]) \\
&= \Pr[H_0]-\Pr[H_1]+\Pr[H_1]-\Pr[H_2]...+\Pr[H_{m-x-1}]-\Pr[H_{m-x}] \\
&= \Pr[H_0]-\Pr[H_{m-x}] \\
&= \Pr[G_3]-\Pr[G_4] \\
&= \Pr[G_3]-\Pr[G_0]
\end{aligned}
$$

Having shown equations (1), (2), (3) and (4), we thus conclude our proof of the theorem. ∎