Structured Encryption for Indirect Addressing

Ruth Ng[1], Alexander Hoover[2], David Cash[2], and Eileen Ee[1]

[1] DSO National Laboratories
[2] University of Chicago

Abstract. The Structured Encryption (StE) framework can be used to capture the encryption and querying of complex data structures on an honest-but-curious server. In this work, we introduce a new type of StE called *indirectly addressed multimap encryption (IA-MME)*. We propose two IA-MME schemes: the *the layered multimaps approach* which extends and generalizes the existing "multimap chaining" approach, and a novel technique called the *single multimap approach* which has comparable efficiency and strictly better security. We demonstrate that our formalisms simplify and modularize StE solutions for real-world use cases in searchable encryption and SQL databases, and provide simulations demonstrating that our IA-MME constructions lead to tangible efficiency and security gains on realistic data.

1 Introduction

Computing on encrypted data has tremendous potential to mitigate the risk of placing data in the hands of cloud services. Amongst many approaches, *Structured Encryption* (StE) [14] has emerged as a promising tool for efficiently outsourcing encrypted data and query computation. StE allows one to encrypt a data structure and then delegate the ability to run queries via query-specific tokens. The StE definition is sufficiently general that it can be used to capture the functionality that would be desired in many real-world encrypted databases, including keyword-search (e.g. over documents in SSE), relational databases (e.g. SQL), and web graphs (e.g. social networks).

While many techniques can fit into the definitional framework of StE, much research has been on simple, efficient constructions from basic symmetric encryption with few rounds of interaction. This efficiency is enabled by allowing for some controlled leakage to the server such as the size and access pattern to the database. As such, the security of an StE scheme can be directly quantified through its *leakage profile*.

Chase and Kamara introduced "multimap chaining" as a technique to build more advanced StE from simpler StE primitives. The technique involves using multiple multimaps – a mapping $\mathbf{M}$ from labels ℓ to tuples of values $\mathbf{M}[\ell]$ – to index complicated data structures in such a way that the tokens for accessing one multimap were put in a second. These are then encrypted using multimap encryption (MME), a well-studied StE primitive. Multimap chaining has since been leveraged to support large and complex subsets of SQL queries on SQL databases [23,26,12].

Our work extends and generalizes multimap chaining to *indirect addressing* (IA), where a single query may trigger an arbitrary number and pattern of accesses within the data structure. More concretely, IA can be used in a multimap $\mathbf{M}$ where there are labels $\ell, \ell_1, \ldots, \ell_n$ such that $\mathbf{M}[\ell] = (\ell_1, \ldots, \ell_n)$ and for each i $\mathbf{M}[\ell_i] = D_i$ for some payload D_i. For example, IA has been used implicitly in searchable encryption (SSE) for keyword-based document retrieval, where ℓ is a keyword, the ℓ_i are document identifiers and the D_i are the document contents.

Our work develops IA-MME generically from MME primitives in two ways, by extending multimap chaining to the *layered multimaps approach* (LMM) and by a novel technique we call the *single multimap approach* (SMM). Of these, SMM has strictly better security, and is likely to be simpler to implement and more efficient in practice. Through this study, we identify subtle proof issues, design MME primitives, and propose security conditions to enable the IA-MME constructions to be generically functional and secure. We believe these formalisms are of independent interest beyond IA-MME.

The IA-MME abstraction handles the complexity of IA, simplifying the expression of StE schemes. This can be put to use in real-world use-cases, making the schemes easy to understand and customize. We demonstrate this by building several SSE and SQL StE schemes using IA-MME, and running simulations of these schemes on realistic data.

Our contributions. In this work:

1. We formalize *IA-MME as a new StE primitive*, encompassing and modularizing a large class of StE.
2. We generalize the multimap chaining technique from the literature to the *layered multimap approach (LMM)*, which we use to build IA-MME generically from standard MME primitives. In doing so, we identify a proof issue with the generic reduction which we resolve via a sufficient condition on some of the MME primitives called *content obliviousness*. We note that the proof issue is also endemic in multimap chaining proofs from prior work, and our condition recovers them.
3. We propose the *single multimap approach (SMM)*, an IA-MME which has strictly better security than LMM without sacrificing efficiency. By amalgamating multiple multimaps into a monolithic index, setup leakage is confined to the total size of the encrypted data structures (instead of the size of the constituent multimap layers). To build SMM, we introduce a new type of MME primitive called *response-flexible MME* and a new notion of *security in the presence of token-values*.
4. We use IA-MME as a primitive to design schemes for *real-world use cases in SSE and SQL StE*. The modularity of IA-MME allows us to reframe existing schemes and explore new approaches to these use cases simply. We conclude with some *simulations on realistic datasets*, demonstrating that IA-MME can bring huge storage savings (over naïve MME solutions) and that the leakage reduction of SMM over LMM is significant.

1.1 Related Work

StE was first introduced by CK [14] as a generalization of searchable symmetric encryption which was first introduced by SWP and formalized by CGKO [35,15]. The StE framework can and has been used to capture many real-world use cases including encrypting SQL data [23,26,12] and supporting rich keyword queries in document storage systems [10,11,39,17].

Added functionality and security has been studied for specific forms of StE, including support for dynamic data structures [28,27], volume hiding queries [24,32,31], models for adaptive compromise [21], costs of minimizing leakage [33,25] and many more [19,5,13,36,7,2,8,22,1,3,16].

StE has been subject to so-called *leakage-abuse attacks* which can sometimes recover damaging information about queries and encrypted data [20,30,9,34,40]. The attacks work against proven-secure constructions by exploiting the permitted leakage, so they are independent of possible gaps in proofs due to composition. However, reducing leakage in order to limit leakage abuse has been a common goal.

2 Preliminaries

Given positive integer n, let $[n] = \{1, \ldots, n\}$. We denote the cardinality of a tuple $\mathbf{t}$ with $\#(\mathbf{t})$, the empty tuple with $()$ and the bitlength of a string s with $|s|$. In pseudocode, we will assume that all integers are intialized to 1 and tuples to $()$.

Table mappings. We capture mappings as lookup tables of the form $\mathbf{T}$. These map labels $\ell \in \{0,1\}^*$ to values $\mathbf{T}[\ell] \in \{0,1\}^* \cup \bot$. In pseudocode, uninitialized tables are assumed to map all labels to $\bot$.

Games. We use the code-based game-playing framework of BR [6]. Given oracle O and adversary $\mathcal{A}$, we write $x \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{O}}(x_1, \ldots, x_m)$ to denote that $\mathcal{A}$, a possibly randomized algorithm, is run with inputs $x_1, \ldots, x_m$ and its output is x. It has black-box access to O and can make as many queries as it likes. Given game G we write $\Pr[\mathrm{G}(\mathcal{A})]$ to denote the probability that $\mathcal{A}$ plays G and the latter returns true.

Function families, PRF security. A function family F defines a key set $\mathsf{F.KS}$ and output length $\mathsf{F.ol}$. It defines an evaluation algorithm $\mathsf{F.Ev} : \mathsf{F.KS} \times \{0,1\}^* \to \{0,1\}^{\mathsf{F.ol}}$. We define PRF security for function family F via the game $\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}$ depicted in Fig. 1. Given adversary $\mathcal{A}$, let $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A}) = 2\Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A})] - 1$ be its PRF advantage.

We give a stronger game and advantage definition in Appendix E Fig. 17 for PRF security in an idealized model with adaptive compromise, taken from [21]. It is only needed for Theorem 1, so we omit it from the main text.

Symmetric Encryption and IND$-security. A symmetric encryption scheme SE defines key set $\mathsf{SE.KS}$, encryption algorithm $\mathsf{SE.Enc}$ and decryption algorithm $\mathsf{SE.Dec}$. and ciphertext length function $\mathsf{SE.cl}$. We require that if $C \leftarrow\!\!\$\ \mathsf{SE.Enc}(K, M)$ then $|C| = \mathsf{SE.cl}(|M|)$ and $\mathsf{SE.Dec}(K, C) = M$.

Game $\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A})$	**Game** $\mathrm{G}^{\mathrm{ind\$}}_{\mathsf{SE}}(\mathcal{A})$
$b \leftarrow\$ \{0,1\}$; $K \leftarrow\$ \mathsf{F.KS}$	$b \leftarrow\$ \{0,1\}$; $K \leftarrow\$ \mathsf{SE.KS}$
$b' \leftarrow\$ \mathcal{A}^{\mathrm{FN}}$; Return $b = b'$	$b' \leftarrow\$ \mathcal{A}^{\mathrm{ENC}}$; Return $b = b'$
Oracle $\mathrm{FN}(X)$	**Oracle** $\mathrm{ENC}(m)$
If $\mathbf{T}[X] = \perp$ then $\mathbf{T}[X] \leftarrow\$ \{0,1\}^{\mathsf{F.ol}}$	$c_1 \leftarrow\$ \mathsf{SE.Enc}(K, m)$
$c_1 \leftarrow\$ \mathsf{F.Ev}(K, X)$; $c_0 \leftarrow \mathbf{T}[X]$; Return c_b	$c_0 \leftarrow\$ \{0,1\}^{\lvert c_1 \rvert}$; Return c_b

Fig. 1. Games used in defining PRF security of function family F (left) and IND$-security of symmetric encryption scheme SE (right) against adversary $\mathcal{A}$.

We say SE is IND$-secure if its ciphertexts are indistinguishability from random strings. This is captured in the game $\mathrm{G}^{\mathrm{ind\$}}_{\mathsf{SE}}$ in Fig. 1. Given adversary $\mathcal{A}$, let $\mathbf{Adv}^{\mathrm{ind\$}}_{\mathsf{SE}}(A) = 2[\Pr[\mathrm{G}^{\mathrm{ind\$}}_{\mathsf{SE}}(\mathcal{A})] - 1$ be its IND$ advantage. We give an additional game and advantage definition in Appendix E Fig. 17 for KP security which captures an idealized model with adaptive compromise, taken from [21]. This definition is only needed for Theorem 1, so we omit it from the main text.

2.1 Structured Encryption

The following definitions follow CK's formalism [14].

Data Types, Structured Encryption. A data type DT defines domain set $\mathsf{DT.Dom}$, query set $\mathsf{DT.Qrys}$, and a deterministic evaluation function $\mathsf{DT.Eval} : \mathsf{DT.Dom} \times \mathsf{DT.Qrys} \to \{0,1\}^* \cup \{\perp\}$.

A structured encryption scheme StE for DT defines a non-empty key set $\mathsf{StE.KS}$ and the following algorithms:

- Randomized encryption algorithm $\mathsf{StE.Enc}$ which takes as input a data structure $DS \in \mathsf{DT.Dom}$ and a key $K \in \mathsf{StE.KS}$. It returns an encrypted data structure $ED \in \{0,1\}^*$.
- Possibly randomized token generation algorithm $\mathsf{StE.Tok}$ which takes as input a key and a query $q \in \mathsf{DT.Qrys}$, and it returns a fixed-length token $tk \in \{0,1\}^{\mathsf{StE.tl}}$.
- Deterministic evaluation algorithm $\mathsf{StE.Eval}$ which takes as input a token and an encrypted data structure, and returns a ciphertext $c \in \{0,1\}^*$ or $\perp$.
- Deterministic decryption algorithm $\mathsf{StE.Dec}$ which takes a key and a ciphertext, and returns a query output $s \in \{0,1\}^*$ or $\perp$.

The correctness condition is that $\Pr[\mathsf{StE.Dec}(K, c) = \mathsf{DT.Eval}(DS, q)] = 1$ where the probability is taken over all $K \in \mathsf{StE.KS}$, $DS \in \mathsf{DT.Dom}$ and $q \in \mathsf{DT.Qrys}$ and the random variables are defined via $ED \leftarrow\$ \mathsf{StE.Enc}(K, DS)$, $tk \leftarrow\$ \mathsf{StE.Tok}(K, q)$, and $c \leftarrow \mathsf{StE.Eval}(tk, ED)$.

We often construct StE for complex data types from StE primitives which reveal query results to the server. We refer to this class of StE as *response revealing* (RR) StE. More precisely, an RR StE for DT is such that for all $DS \in \mathsf{DT.Dom}$ and $q \in \mathsf{DT.Qrys}$ we have that $\mathsf{DT.Eval}(DS, q) = c = \mathsf{StE.Dec}(K, c)$ (where the random variables are defined as in the correctness condition).

While we allow $\mathsf{StE.Eval}$ and $\mathsf{StE.Dec}$ to return $\perp$, this is to handle malformed input. In this work, we leave implicit the handling of such in pseudocode and assume no queries will trigger these behaviors.

Semantic security. CK defines adaptive semantic security for StE using game $\mathrm{G}^{\mathrm{ss}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}$ where $\mathcal{L}, \mathcal{S}$ are the leakage algorithm and simulator respectively. In $\mathrm{G}^{\mathrm{ss}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}(\mathcal{A})$, all three algorithms (i.e. $\mathcal{A}, \mathcal{L}, \mathcal{S}$) can be run in "setup" or "query" mode, as indicated using the first argument (i.e. $\mathtt{s}$ or $\mathtt{q}$).

The game proceeds in two phases, with the adversary providing a data structure in the "setup phase" then making queries to that data structure in the "query phase". We refer to the output of the leakage algorithm in the first (resp. second) phase as the "setup leakage" (resp. "query leakage"). The adversary's goal is to distinguish the encrypted data structure and tokens from those generated by a simulator using the leakage. The details of $\mathrm{G}^{\mathrm{ss}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}$ are given in Fig. 2. The advantage of adversary$\mathcal{A}$ is $\mathbf{Adv}^{\mathrm{ss}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}(\mathcal{A}) = 2\Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}(\mathcal{A})] - 1$.

Notice that with an RR scheme, in order for $\mathcal{S}$ to provide a realistic simulation of of a token for query q, the query leakage must include $\mathsf{DT.Eval}(q, DS)$ (the query response) because this must be returned should

Game $\mathrm{G}^{\mathrm{ss}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}(\mathcal{A})$	**Oracle** $\textsc{Tok}(q)$
$K \leftarrow\!\!\$\ \mathsf{StE.KS}$; $b \leftarrow\!\!\$\ \{0,1\}$	If $q \notin \mathsf{DT.Qrys}$
$(DS, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathtt{s})$	Return false
If $DS \notin \mathsf{DT.Dom}$ then return false	If $b = 1$
If $b = 1$	$tk \leftarrow\!\!\$\ \mathsf{StE.Tok}(K, q)$
$ED \leftarrow\!\!\$\ \mathsf{StE.Enc}(K, DS)$	Else
Else	$(lk, St) \leftarrow\!\!\$\ \mathcal{L}(\mathtt{q}, q, St)$
$(lk, St) \leftarrow\!\!\$\ \mathcal{L}(\mathtt{s}, DS)$	$(tk, St') \leftarrow\!\!\$\ \mathcal{S}(\mathtt{q}, lk, St')$
$(ED, St') \leftarrow\!\!\$\ \mathcal{S}(\mathtt{s}, lk)$	Return tk
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Tok}}(\mathtt{q}, ED, St_{\mathrm{a}})$	
Return $b = b'$	

Fig. 2. Game used in defining adaptive semantic security of structured encryption scheme StE for data type DT with respect to leakage algorithm $\mathcal{L}$ and simulator $\mathcal{S}$.

the adversary run $\mathsf{StE.Eval}(tk, ED)$. Therefore, we will assume that all leakage algorithm associated to RR StE schemes are such that if $(lk, St) \leftarrow\!\!\$\ \mathcal{L}(\mathsf{q}, q, St)$ then $lk = (\mathsf{DT.Eval}(q, DS), lk')$ for some lk'.

MULTIMAP DATA STRUCTURE. The multimap data type MMdt captures mappings from labels to tuples of values. In particular, its domain elements are table mappings (as defined above) which map all labels of some fixed bitlength $\mathsf{MMdt.lLen}$ to tuples of values of fixed bitlength $\mathsf{MMdt.vLen}$, and maps all other labels to $()$. We refer to these as *multimaps*. The multimap query set is the label set while evaluation simply retrieves the mapped tuple associated to a label. This means that $\mathsf{MMdt.Qrys} = \{0,1\}^{\mathsf{MMdt.lLen}}$ and $\mathsf{MMdt.Eval}(\mathbf{M}, \ell) = \mathbf{M}[\ell]$.

In this paper we will assume all multimaps come from the MMdt with appropriate length values, which we abbreviate to $\mathsf{lLen}, \mathsf{vLen}$ and sometimes leave implicit. In pseudocode, uninitialized multimaps are assumed to map all $\ell \in \{0,1\}^{\mathsf{lLen}}$ to $()$. While other definitions may not have required fixed-length labels or values, ours loses no generality because values may be padded or broken up into equal length blocks while labels may be hashed to a common length.

MULTIMAP ENCRYPTION. StE for MMdt is a key primitive used to build more complex StE schemes. We will refer to these as multimap encryption (MME) schemes. As above, each MMdt has an implicit value of $\mathsf{lLen}, \mathsf{vLen}$ associated to it. Therefore, each MME scheme need only support a constant implied label length and value length.

As discussed in Section 1.1, a large number of MME primitives have been proposed in the searchable encryption, secure indexing, encrypted databases and StE literature. While none of our schemes require specific MME primitives nor leakage profiles, it is still useful to contextualize the security of schemes using the leakage profile of state-of-the-art schemes from the literature. Intuitively, the setup leakage of this profile reveals the total number of values in the multimap. The query leakage are the query and access patterns. The former is the equality pattern of the queries made thus far. The latter is the query response for RR schemes, and the number of values returned otherwise. In Appendix A we give a full description of these leakage profiles and examples of MME schemes which achieves them. These are derived from the popular SSE scheme $\prod_{\mathrm{bas}}$ (2Lev in the Clusion library) by CJJ+ [10,29].

3 Indirectly Addressed Encrypted Multimaps

We now show how multimaps with indirect addressing can be captured as a data type. We illustrate some of the issues of constructing StE for IA, which we call "indirectly addressed multimap encryption schemes" (IA-MMEs), via two strawman constructions. In addition to generalizing the "multimap chaining" technique of AC [14], our IA-MME syntax allows us to elegantly and simply capture many desired StE functionalities. We discuss some such schemes in Section 6 in the area of searchable encryption and SQL StE.

We then demonstrate the utility of such a primitive by capturing Searchable Encryption (as defined by CGKO [15]) and "chained multimaps" (as defined by AC [14]) as special cases of it.

Alg $\mathsf{IA.Eval}(\mathbf{M}, \ell)$
$(b_1\|x_1, \ldots, b_n\|x_n) \leftarrow \mathbf{M}[\ell]$
For $i \in [n]$ do
  If $b_i = 0$ then $v_i \leftarrow x_i$ else $v_i \leftarrow \mathsf{IA.Eval}(\mathbf{M}, x_i)$
Return $(v_1, \ldots, v_n)$

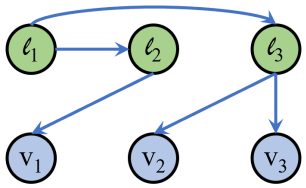

Fig. 3. Evaluation algorithm for IA data type (left), and graphical visualization of $\mathbf{M} \in \mathsf{IA.Dom}$ with $\mathbf{M}[\ell_1] = (1\|\ell_2, 1\|\ell_3)$, $\mathbf{M}[\ell_2] = (0\|v_1)$ and $\mathbf{M}[\ell_3] = (0\|v_2, 0\|v_3)$ (right).

INDIRECT ADDRESSING DATA TYPE IA. Much like MMdt, IA also involves fixed-length labels and values, but now we require they all be of some common length len. For the same reasons as those presented in Section 2, this simplifying assumption does not hurt the generality of our formalism.

Intuitively, indirectly addressed multimaps (IA-MMs) can be seen as directed acyclic graphs (DAGs) with nodes for each label and value. Edges may point from labels either to either values or other labels. When a label is queried, the graph is traversed starting from that node and returns all descendant leaf (i.e. value) nodes. Note that the acyclic requirement is to ensure that this evaluation terminates.

More formally, $\mathsf{IA.Dom}$ contains multimaps with $\mathsf{lLen} = \mathsf{len}$ and $\mathsf{vLen} = \mathsf{len} + 1$ while $\mathsf{IA.Qrys} = \{0,1\}^{\mathsf{len}}$. When $\mathsf{IA.Eval}(\mathbf{M}, \ell)$ is called, it recursively accesses the multimap, interpreting the values with leading bit 0 as values (which can be returned as is) and those with leading bit 1 as labels (which prompts the recursive access). In Fig. 3, we depict (part of) an example IA-MM as a DAG (specifically $\mathbf{M}$ where $\mathbf{M}[\ell_1] = (1\|\ell_2, 1\|\ell_3)$, $\mathbf{M}[\ell_2] = (0\|v_1)$ and $\mathbf{M}[\ell_3] = (0\|v_2, 0\|v_3)$) and provide the full pseudocode for $\mathsf{IA.Eval}$. Notice that in the example, $\mathsf{IA.Eval}(\mathbf{M}, \ell_1) = (v_1, (v_2, v_3))$.

DEPTH. The number of "layers" of indirect addressing in an IA-MM is measured using a function depth. It can be used to measure the depth of some query ℓ in $\mathbf{M} \in \mathsf{IA.Dom}$, via $\mathsf{depth}(\mathbf{M}, \ell)$. In the DAG, this corresponds to the maximum length path beginning at ℓ. Sometimes, it is useful to describe the maximum depth of all labels in the IA-MM. We call this the "depth of $\mathbf{M}$" and expand our earlier notation to denote this with $\mathsf{depth}(\mathbf{M})$. In the DAG, this corresponds to the maximum length path.

The precise definition of these algorithms are:

Alg $\mathsf{depth}(\mathbf{M}, \ell)$
$(b_1\|x_1, \ldots, b_n\|x_n) \leftarrow \mathbf{M}[\ell]$
If $b_1 = \cdots = b_n = 0$ then return 1
else return $\left(1 + \max_{i \in [n], b_i = 1} \mathsf{depth}(\mathbf{M}, x_i)\right)$

Alg $\mathsf{depth}(\mathbf{M})$
Return $\max_{\ell \in \{0,1\}^{\mathsf{len}}} \mathsf{depth}(\mathbf{M}, \ell)$

Note that in the example visualized in Fig. 3 we have $\mathsf{depth}(\mathbf{M}, \ell_1) = 2$, $\mathsf{depth}(\mathbf{M}, \ell_2) = 1$, and $\mathsf{depth}(\mathbf{M}) = 2$.

In our IA-MME schemes we will assume that the $\mathbf{M}$ to be encrypted have some predetermined and finite maximum depth $\mathsf{IA.dp}$. In other words, we require that all $\mathbf{M} \in \mathsf{IA.Dom}$ have $\mathsf{depth}(\mathbf{M}) \leq \mathsf{IA.dp}$. To avoid degeneracy (to MMdt), we assume that $\mathsf{IA.dp} \geq 2$.

UNIFORMITY. We say that $\mathbf{M} \in \mathsf{IA.Dom}$ is *uniform* if all the values associated to each label have the same depth. In other words, for all $\ell \in \{0,1\}^{\mathsf{len}}$ where $\mathbf{M}[\ell] = (b_1\|x_1, \ldots, b_n\|x_n)$, we expect that either $b_1 = \cdots = b_n = 0$ (i.e. they are all values) or $b_1 = \cdots = b_n = 1$ and $\mathsf{depth}(\mathbf{M}, x_1) = \cdots = \mathsf{depth}(\mathbf{M}, x_n)$. For example, multimaps from the multimap data type are all uniform, depth-1 IA-MMs while $\mathbf{M}$ depicted in Fig. 3 is a uniform, depth-2 IA-MM.

As we show later, some IA-MME techniques only work with uniform IA-MMs. Since this may suffice for some applications, we capture this as a data type UIA so that we can define StE for it. UIA is identical to IA except $\mathsf{UIA.Dom}$ contains only the uniform IA-MMs.

STRAWMAN 1: NAÏVE ENCRYPTION. A natural first instinct toward constructing IA-MMEs (StE for IA) would be to simply encrypt $\mathbf{M} \in \mathsf{IA.Dom}$ using an MME scheme MME. This achieves some intuitive notion of security since MME is secure and $\mathbf{M}$ is itself a multimap. Notice that we would want to use non-RR MME here, to avoid leaking the ℓ_i and v_i to the server.

A problem crops up, however, when one tries to query this encrypted data structure. Consider what happens when computing $\mathsf{IA.Eval}(\mathbf{M}, \ell_1)$ in the example depicted in Fig. 3. If the token $\mathsf{MME.Tok}(K, \ell_1)$ is

presented to the server, the best it can do is return the client a ciphertext which decrypts to $(1\|\ell_2, 1\|\ell_3)$. The client would need to then generate the tokens for ℓ_2, ℓ_3 and query the server with those to compute $\mathbf{M}[\ell_1]$. More generally, this solution would require $\mathsf{depth}(\mathbf{M}, \ell)$ rounds of communication between the client and server to retrieve $\mathsf{IA.Eval}(\mathbf{M}, \ell_1)$. This added communication can increase bandwidth and latency in practice and, as we demonstrate later, is unnecessary.

STRAWMAN 2: INLINED PAYLOADS. A second natural approach would be to do away with the indirect addressing entirely and inline the values within $\mathbf{M}$. In our example, this would mean transforming $\mathbf{M}$ into the multimap $\mathbf{M}'$ where $\mathbf{M}'[\ell_1] = (v_1, (v_2, v_3))$, $\mathbf{M}'[\ell_2] = (v_1)$, and $\mathbf{M}'[\ell_3] = (v_2, v_3)$. These values should be padded so vLen is equal to $|(v_2, v_3)|$, before $\mathbf{M}'$ is encrypted with an MME scheme. This can be queried in a single round using the MME.

This approach is both correct and secure but we lose any storage efficiencies that indirect addressing afforded us. In existing applications making use of such data structures (e.g. those discussed below), many depth-1 labels are associated with a long tuple of values, and are "pointed to" by multiple labels of higher depth. For example, in SSE, this would entail to storing one copy of each document for each keyword that it is associated to. In the worst case, inlining a depth-n IA-MM could lead to a power-of-n blowup in storage. As such, this solution is also unsatisfying.

4 Layered-Multimap Approach

We extend the "multimap chaining" technique from the literature to an IA-MME technique that we call "layered multimaps" (LMM). In doing so, we identify an issue that prevents the security of LMM schemes from being proven in full generality and provide a sufficient condition on the MME primitives – *content oblivious leakage algorithms* – to recover this proof approach. Multimap chaining schemes in the literature (e.g. $\mathsf{LabGraph}$, SPX, and OPX) do not address this issue, so their proofs are technically incorrect and can be restored with the addition of our condition.

In Section 5, we present an IA-MME technique with strictly better leakage. So we focus on the intuition of LMM and the sufficient condition that restores proofs from prior work. A full discussion on LMM can be found in Appendix B.

LMM APPROACH FOR UNIFORM, DEPTH-2 IA-MMS. The multimap chaining approach of prior work is equivalent to the handling of depth-2, uniform IA-MMs. We will capture this as LMM_u, an StE scheme for UIA where $\mathsf{UIA.dp} = 2$.

The scheme LMM_u will index a depth-2, uniform IA-MM $\mathbf{M}$ using two multimaps $\mathbf{M}_1, \mathbf{M}_2$. $\mathbf{M}_1$ is a copy of all the depth-1 mappings in $\mathbf{M}$ (i.e. if $\mathsf{depth}(\mathbf{M}, \ell) = 1$ then $\mathbf{M}_1[\ell] = \mathbf{M}[\ell]$). This will be encrypted using MME scheme MME_1. $\mathbf{M}_2$ contains all other (i.e. depth-2) mappings in $\mathbf{M}$. However, instead of mapping these to other (depth-1) labels, they will be mapped to tokens for accessing that label using MME_1. $\mathbf{M}_2$ will be encrypted using RR MME scheme MME_2.

When a query ℓ is made, the token generation algorithm will return the tokens to access ℓ with both MME primitives (since the token generation algorithm need not "know" the depth of ℓ). If $\mathsf{depth}(\mathbf{M}, \ell) = 1$, a ciphertext can be retrieved from the encrypted $\mathbf{M}_1$. If $\mathsf{depth}(\mathbf{M}, \ell) = 2$, the encrypted $\mathbf{M}_2$ will return a tuple of tokens (since it was encrypted with an RR scheme) which can then be used to query the encrypted $\mathbf{M}_1$ to retrieve ciphertexts. In either case, $\mathsf{MME}_1.\mathsf{Dec}$ can be used to retrieve $\mathsf{UIA.Eval}(\mathbf{M}, \ell)$.

EXTENDING LMM_u TO LMM. We can extend the LMM technique from uniform, depth-2 IA-MMs to arbitrary IA-MMs. The result of this is LMM, StE scheme for IA with no restrictions on depth.

Extending LMM_u, LMM will index an IA-MM $\mathbf{M}$ with $\mathsf{depth}(\mathbf{M}) = D$ multimaps. Each $\mathbf{M}_i$ will index the queries of depth i and be encrypted with MME scheme MME_i, where $\mathsf{MME}_2, \ldots, \mathsf{MME}_D$ are RR. For $i \geq 2$, the labels (i.e. values where $b_i = 1$) in $\mathbf{M}_i$ will be replaced with tokens prior to encryption. We derive keys for all MME primitives using a function family in the standard way.

We embellish LMM_u in two ways in this extension. First, alongside each token in the $D-1$ RR multimaps, we include a depth-indicator for the location to points to. This improves efficiency by "telling" the server which multimap to subsequently access with that token. Additionally, while LMM_u tokens were made out of two MME tokens, we can do better than sending D such tokens in LMM. Instead, we use an additional multimap $\mathbf{M}_0$ to index which token to use and which multimap to access for each possible client query. The client can then just send the one token needed to access $\mathbf{M}_0$.

Game $\mathrm{G}^{\mathrm{obliv}}_{\mathcal{L}}(\mathcal{A})$

$\big(\mathbf{M}_1, \mathbf{M}_2, (\ell_1, \ldots, \ell_n)\big) \leftarrow\!\!\$\ \mathcal{A}$

For $i = 1, 2$ do

  $(lk_i^0, St_i) \leftarrow\!\!\$\ \mathcal{L}(\mathtt{s}, \mathbf{M}_i)$

  For $j \in [n]$ do $\big((\mathbf{v}, lk_i^j), St_i\big) \leftarrow\!\!\$\ \mathcal{L}(\mathtt{q}, \ell_j, St_i)$

Return $\big((lk_1^0 \neq lk_2^0) \vee \cdots \vee (lk_1^n \neq lk_2^n)\big)$

Fig. 4. Game used to define content obliviousness of leakage algorithm $\mathcal{L}$. Here, we assume that the adversary provides $\mathbf{M}, \mathbf{M}'$ which are homomorphic, and $\ell_1, \ldots, \ell_n \in \mathsf{DT.Qrys}$.

We provide the full pseudocode and security analysis of LMM in Appendix B.

Technicality in LMM proof. Unfortunately, the LMM construction cannot be proven secure in full generality. Prior work's approach to the proof of $\mathsf{LMM}_{\mathrm{u}}$'s security involves composing the leakage algorithms (resp. simulators) for $\mathsf{MME}_1, \mathsf{MME}_2$ to a single algorithm $\mathcal{L}$ (resp. $\mathcal{S}$), then "proved" $\mathsf{LMM}_{\mathrm{u}}$ secure under $\mathcal{L}, \mathcal{S}$. The problem arises in the definition of $\mathcal{S}$, where the two constituent simulators need to "work together" at query time to return tokens to depth-2 labels which must be able to recursively access the simulated data structures. (If such an access cannot be done, the adversary can trivially distinguish this from the genuine StE algorithms). The constituent simulator's lack of shared state make them unlikely to present consistent output. So if these simulators were instantiated pathologically, there will be no way to build a secure $\mathsf{LMM}_{\mathrm{u}}$ even with secure primitives. In Appendix B, we present this proof approach in full and discuss how a pathological primitive would scuttle the proof.

Homomorphic multimaps, content oblivious leakage. In order to resolve the above proof issue for schemes with the LMM approach, we make a sufficient assumption on the leakage algorithm associated to the RR MME schenes which we call *content obliviousness*. Since our condition has to do with how the leakage algorithm treats multimaps of the "same shape", we start by giving a formal definition of this.

Specifically, we say that two multimaps $\mathbf{M}_1, \mathbf{M}_2 \in \mathsf{MMdt}$ are *homomorphic* if the tuples mapped to by each label are of the same length (i.e. $\#(\mathbf{M}_1[\ell]) = \#(\mathbf{M}_2[\ell])$ for all $\ell \in \{0,1\}^{\mathsf{lLen}}$). In other words, if $\mathbf{M}_1[\ell] = (v_1, \ldots, v_n)$ then $\mathbf{M}_2[\ell]$ must also be a tuple of n (not necessarily identical) values.

Let MME be an RR MME scheme used in any LMM scheme (i.e. $\mathsf{LMM}_{\mathrm{u}}$ or LMM), with leakage algorithm $\mathcal{L}$. Then, $\mathcal{L}$ is *content oblivious* if homormorphic multimaps have the same leakage (modulo query responses). We capture this precisely in the game $\mathrm{G}^{\mathrm{obliv}}_{\mathcal{L}}$ in Fig. 4, where $\mathcal{L}$ is a leakage algorithm. The game also assumes that the adversary provides homomorphic $\mathbf{M}, \mathbf{M}'$ and $\ell_1, \ldots, \ell_n \in \mathsf{DT.Qrys}$. We say that $\mathcal{L}$ is content oblivious if $\Pr[\mathrm{G}^{\mathrm{obliv}}_{\mathcal{L}}(\mathcal{A})] = 0$ for all adversaries $\mathcal{A}$. Note that our notion assumes a leakage algorithm consistent with an RR MME scheme (in particular, that query leakage contains $\mathbf{v} = \mathbf{M}[\ell_j]$). We believe that analogous definitions for other MME are of independent interest, but in our work we only require this of RR MME schemes.

Since the "standard" leakage profile (discussed in Section 2.1) has content oblivious leakage, so do all state-of-the-art RR MME schemes (including $\mathsf{MME}^{\mathrm{r}}_{\pi}$ in Appendix A). This condition is sufficient to prove the security of $\mathsf{LMM}_{\mathrm{u}}$ and LMM. For completeness, we provide these proofs in Appendix B.

It is worth noting that content obliviousness is a *sufficient* condition to resolving the proof issue. Specifically, conditions could be placed on the simulator for the proof to go through even for some non-content oblivious RR MME, by requiring the simulator "behaves well" when the leakage input is substituted in the proof. Nevertheless, we believe our content obliviousness assumption is simpler to use in practice because it only concerns the leakage profile which can often be deduced by seeing the scheme in action. A condition based on the simulator is more opaque to someone who does not work through the proof details. Additionally, all state-of-the-art schemes are already content oblivious. For these reasons, we leave open the problem of finding the necessary leakage properties and other simulator properties that allow this proof technique to go through. We see it as a problem of theoretical rather than practical interest.

Implications for prior work. We identified three schemes in the literature where the above "proof bug" occurs, $\mathsf{LabGraph}$, SPX and OPX. We discuss the specific issues in their proofs, and the insufficient assumptions made therein in Appendix C. These results can be restored by assuming that the RR MME primitives are content oblivious. This also means that existing systems using these results are still "secure" so long as they used state-of-the-art primitives.

Discussion. Before moving on, we draw attention to a couple of strengths and weaknesses of the LMM approach. On the one hand, it can efficiently and securely realized using existing MME primitives. In particular, it avoids the pitfalls of the two naïve techniques proposed in Section 3.

However, the use of multiple data structures in the LMM approach is unsatisfying for a number of reasons. First, the adversary learns about the relative size of the layers and deduce information about the "structure" of the IA-MM. Second, for non-uniform IA-MMs, the $\mathbf{M}_0$ index and depth indicators in the RR multimaps have to be used to "tell the server" which data structure it should be searching on. Both of these add to the complexity and overhead of the scheme.

In this, we see a common thread: if we are able to index and encrypt $\mathbf{M}$ as a monolithic data structure instead of splitting it into several multimaps, we would avoid the above weaknesses. The challenge is doing so using simple primitives and without added overhead or complexity. This is the focus of our next section.

5 Single-Multimap Approach

In Section 4, we suggested that the LMM could be improved by indexing the entire IA-MM using a single monolithic encrypted multimap. To realize this intuition, we require a multimap encryption primitive with expanded functionality (response-flexible MME) and stronger security (TV-security). We explore these properties in detail, since they have nuanced definitions, interesting leakage implications and are non-trivial to construct. We then use such MME to construct SMM, the IA-MME employing the single-multimap (SMM) approach.

5.1 Response-Flexible MME

Recall that in the LMM approach, we used different MME schemes to encrypt tokens and values so that the tokens could be revealed to the server at query time. If we want to collapse all this indexing into a single multimap, we need an MME scheme which reveals the intermediate tokens to the server but not the final values. To achieve this, we start by defining a new type of MME which we call *response-flexible* (RF) MME.

Syntax. For precision, we need to modify the multimap data type slightly so that one bit of each value can be used to indicate the desired response type. This creates the RF data type RFdt which has the same domain and query set as MMdt but its evaluation omits the indicator bit. In other words,

$$\mathbf{M}[\ell] = (b_1\|v_1, \ldots, b_n\|v_n) \implies \mathsf{RFdt.Eval}(\mathbf{M}, \ell) = (v_1, \ldots, v_n).$$

For consistency, we will let lLen refer to the length of v_i (not $b_i\|v_i$).

Now suppose $\mathsf{MME_f}$ is an StE for RFdt. We say that $\mathsf{MME_f}$ is RF if its evaluation reveals the indicator bit of all values, and the remaining bits of those with $b_i = 1$. In other words, for $\mathbf{M}[\ell]$ as above, if $ED \leftarrow\!\!\$\ \mathsf{MME_f.Enc}(K, \mathbf{M})$ and $tk \leftarrow\!\!\$\ \mathsf{MME_f.Tok}(K, \ell)$ we have:

$$\mathsf{MME_f.Eval}(tk, ED) = (b'_1\|u_1, \ldots, b'_n\|u_n)$$
$$\text{where } \begin{cases} b'_i\|u_i = b_i\|v_i & \text{if } b_i = 1, \\ b'_i = b_i & \text{if } b_i = 0. \end{cases}$$

The final requirement we make is on the decryption algorithm. Since the evaluation output is a tuple which may contain some unencrypted values, we will assume the existence of a decryption algorithm that works on a per-value basis. More precisely, we need that $\mathsf{MME_f}$ defines algorithm $\mathsf{MME_f.Dec1}$ where

$$\mathsf{MME_f.Dec}\big(K, (b_1\|c_1, \ldots, b_n\|c_n)\big) = (u_1, \ldots, u_n)$$
$$\text{where } \begin{cases} u_i = c_i & \text{if } b_i = 1 \\ u_i = \mathsf{MME_f.Dec1}(K, c_i) & \text{if } b_i = 0. \end{cases}$$

RF MME Constructions. Before we start building RF MME, we give some intuition as to the leakage profile that one should expect of such a scheme. Notice that if all values in $\mathbf{M}$ have $b_i = 1$, the scheme is

functionally equivalent to an RR MME. Therefore, one may expect that state-of-the-art RF leakage profiles are equivalent to the analogous RR profile, except in the query leakage. In particular, if the query output is $(b_1\|v_1, \ldots, b_n\|v_n)$, the RR scheme would have returned $(v_1, \ldots, v_n)$ as part of the leakage and the RF scheme would return only the v_i where $b_i = 1$.

This intuition hints at our approach to building RF MME – from RR MME. Many RR MME schemes in the literature will encrypt the values associated with each label using a differentiated key. This key is used by the server to recover said values at query time. These schemes can be adapted to achieve response flexibility by only encrypting the RR values with the differentiated key, and encrypting the other values using a key which is not given to the server. This is exactly our approach to extending our RR variant of the $\prod_{\text{bas}}$ SSE scheme to the RF variant $\mathsf{MME}^{\text{r}}_{\pi}$ in Appendix A. In Section 5.2, we will show that $\mathsf{MME}^{\text{f}}_{\pi}$ is secure under the stronger TV-security notion. In particular, Theorem 1 subsumes the proof of $\mathsf{MME}^{\text{f}}_{\pi}$'s semantic security.

One might hope for a more generic transform that converts RR to RF MME schemes while maintaining semantic security. This is indeed possible, as we demonstrate with our transform **RfT** in Appendix D which takes an RR MME scheme MME_{r} and symmetric encryption scheme SE and returns RF MME MME_{f}. Intuitively, MME_{f} will encrypt the values which need to be hidden under a key that is kept secret from the server, before it proceeds with encryption under MME_{r}. At query time, $\mathsf{MME}_{\text{r}}.\mathsf{Eval}$ will peel back its own layer of encryption but it will only see these encrypted values, which can be decrypted during $\mathsf{MME}_{\text{f}}.\mathsf{Dec}$. As one would expect, a reduction can be given from MME_{r} and SE's security to MME_{f}'s under a leakage profile something like the one described above. The proof of this is a bit involved, so we defer a full discussion on **RfT** to Appendix D, where we provide the full pseudocode, leakage profiles and a proof in the stronger TV-security model (which we will define in Section 5.2).

We also briefly mention that using **RfT** with non-pathological RR MEE schemes will usually lead to data being wrapped in two layers of encryption. In practice, it would be better to replace the former layer with the latter (as we did in $\mathsf{MME}^{\text{f}}_{\pi}$).

Discussion. We believe the idea of response flexibility is of independent interest, so we highlight some alternate syntax that we considered which future work may look into. Intuitively, the goal of response flexibility is to allow a data structure to support both revealing and non-revealing query responses. In the approach above, we assume that each value in the multimap is annotated with the desired response type. An alternate approach would be to annotate labels instead of values, thereby requiring that the client decide, at setup time, whether each tuple in the multimap will be revealed (in entirety) at query time or not. This approach loses some of the fine-grain control over response types that the former approach had, but may allow for constructions with less leakage. Future work could look into this to improve the security of applications which do not need such fine-grained control (e.g. uniform IA-MMs).

Both of the above approaches rely on the intrinsic structure of a multimap and are difficult to extend to StE in general. However, we believe an RF analog of RR StE would enrich the entire StE framework. One approach to doing this would be to have a syntax where the client indicates during token generation whether the response should be revealed or not. This has the added advantage that the client can delay the decision of whether responses should be revealed till query time, unlike the above approaches. This added flexibility may be useful in some applications.

While these alternate approaches have their merits, we went with the fine-grained variant because it allowed for the most storage efficient and straightforward SMM construction. Intuitively, only the fine-grained approach allows each entry (i.e. mapping of label to values) in the IA-MM to be captured as a single entry in the $\mathbf{M}_i$ indexes computed during the setup phase. Future work could also explore leakage reduction in this fine-grained setting. In particular, one might extend techniques used to hide the volume and ordering (with respect to the revealed values) of the non-RR values.

5.2 MME Security with Token Values

A challenge in adopting the SMM approach is that a multimap will contain tokens generated under the same key that it will be encrypted with. This is so that the server can perform the necessary recursive lookups using a single data structure. Proving the security of SMM based on a semantically secure MME scheme runs seems out of reach, because there are pathological MMEs that misbehave when they encrypt key-dependent tokens. As a convenience in the proof, we define a stronger assumption on MMEs which we call adaptive security in the presence of token-values (or TV-security).

Game $\mathrm{G}^{\mathrm{tv}}_{\mathsf{MME},\mathcal{L},\mathcal{S}}(\mathcal{A})$

$K \leftarrow\!\!\$\ \mathsf{MME.KS}$; $b \leftarrow\!\!\$\ \{0,1\}$
$(\mathbf{M}, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{s})$
If $b = 1$ then
  For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
    If $\mathsf{Search}(\mathbf{M}[\ell], \{\ell\})$ then return false
    $(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell]$
    For $i \in [n]$ do
      If $b_i = 1$ then $v_i \leftarrow\!\!\$\ \mathsf{MME.Tok}(K, v_i)$
    $\mathbf{M}_1[\ell] \leftarrow (b_1 \| v_1, \ldots, b_n \| v_n)$
  $ED \leftarrow\!\!\$\ \mathsf{MME.Enc}(K, \mathbf{M}_1)$
Else
  $(lk, St) \leftarrow\!\!\$\ \mathcal{L}(\mathsf{s}, \mathbf{M})$; $(ED, St') \leftarrow\!\!\$\ \mathcal{S}(\mathsf{s}, lk)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Tok}}(\mathsf{q}, ED, St_{\mathrm{a}})$; Return $b = b'$

Oracle $\mathrm{Tok}(\ell)$

If $b = 1$ then
  $tk \leftarrow\!\!\$\ \mathsf{MME.Tok}(K, \ell)$
Else
  $(lk, St) \leftarrow\!\!\$\ \mathcal{L}(\mathsf{q}, \ell, St)$
  $(tk, St') \leftarrow\!\!\$\ \mathcal{S}(\mathsf{q}, lk, St')$
Return tk

Alg $\mathsf{Search}((b_1 \| v_1, \ldots, b_n \| v_n), S)$

For $i \in [n]$ do
  If $b_i = 1$ then
    If $v_i \in S$ then return true
    If $\mathsf{Search}(\mathbf{M}[v_i], S \cup v_i)$
      then return true
Return false

Fig. 5. Game used in defining adaptive TV-security of multimap encryption scheme MME with respect to leakage algorithm $\mathcal{L}$ and simulator $\mathcal{S}$.

TV-SECURITY. This notion extends the semantic security definition given in Section 2.1 but applies only to MME (not to all of StE, nor specifically to IA-MM). The security game for defining TV-security for MME is $\mathrm{G}^{\mathrm{tv}}_{\mathsf{MME}}$, depicted in Fig. 5. Intuitively, the game is similar to $\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME}}$ but when the adversary provides a multimap $\mathbf{M}$ they may request that some of the values therein be tokenized before encryption. We require that the values to be tokenized form no cycles when the multimap is viewed as a directed graph.[3] In the "real world" this tokenization is done using MME.Tok prior to encryption with MME.Enc. In the "ideal world" the leakage algorithm gets $\mathbf{M}$ and simulator must construct something comparable to this encryption output. The adversary indicates whether a value should be tokenized using the first bit in each multimap value. The algorithm Search is called recursively in the game to check that the graph visualization of $\mathbf{M}$ is acyclic. Note that in the game we assume $\mathsf{lLen} = \mathsf{MME.tl}$ and $\mathsf{vLen} = \mathsf{MME.tl} + 1$ for $\mathbf{M}$ given by $\mathcal{A}$. The advantage of adversary $\mathcal{A}$ is $\mathbf{Adv}^{\mathrm{tv}}_{\mathsf{MME},\mathcal{L},\mathcal{S}}(\mathcal{A}) = 2\Pr[\mathrm{G}^{\mathrm{tv}}_{\mathsf{MME},\mathcal{L},\mathcal{S}}(\mathcal{A})] - 1$.

Recall that the values in RF MME have a similar indicator bit in their data structure (thereby making it a slightly different data type). In our notion of TV-security for RF MME, we use the *same indicator bit* to indicate response type (for response flexibility) and tokenization (for TV-security). Intuitively, this restricts our study of TV-secure response-flexible scheme to those where values are response revealing if and only if they are tokens. This notion is a perfect fit for our SMM construction because the TV-security game essentially builds an IA-MM. While a more general definition of TV-secure RF MME can be given with separate indicator bits, this introduces too many confusing details in defining and proving TV-security (e.g. the ordering of tokenization and encryption, if both are needed), so we leave the general treatment for future work to explore.

TV-SECURE MME LEAKAGE. Since our TV-security game is still a simulation-based one, we now give some intuition about what kind of leakage a TV-secure MME scheme should aim for. In particular, given the leakage profile of state-of-the-art scheme MME (in the standard model), what analogous leakage algorithm should we hope to prove it TV-secure under?

One might notice that the multimaps encrypted in the standard and TV-security games (i.e. $\mathbf{M}, \mathbf{M}_1$ respectively) are homomorphic. So one might hope that, using state-of-the-art schemes with content oblivious leakage, the profiles would be equivalent. However, the TV-leakage profile needs to simulate the tokens requested by the adversary as well as the tokens values (in the multimap) should the adversary choose to evaluate the token he gets on the encrypted data structure. Because the query leakage associated to token values should be in the TV-security profile but not in the standard profile, one should not expect the same scheme to achieve the same leakage in both games.

Along this line of thought, one might expect that the leakage associated to all token values must be included in the TV-security profile so that the simulator can generate ED. However, this is also an erroneous

[3] This restriction is sufficient for IA and gets around a technical issue in the simulation sketched in Appendix E.

intuition because this is equivalent to revealing the query leakage associated to all these tokens during the setup phase which makes no sense because no queries have been made.

These observations lead us to the correct intuition regarding the TV-security leakage profile. Its setup leakage should be comparable to the standard case. When a query is made, the query leakage of that query and all its descendants (in the graph visualization) should be revealed. This is more than the standard case which would only leak the initial query. We give a concrete example of such a profile in Appendix E, for the RF variant of the $\prod_{\text{bas}}$ scheme.

CONSTRUCTING TV-SECURE RF MME. Many MME schemes from the literature satisfy this stronger notion of security, with the appropriate modifications to their leakage algorithm and simulator to accommodate the tokenization. This includes the schemes in Appendix A. In particular, the proofs of $\prod_{\text{bas}}$'s (standard) semantic security given by CJJ+ and JT [10,21] can be extended to one of TV-security for our response flexible variant $\mathsf{MME}^{\mathrm{f}}_{\pi}$ under similar conditions (namely, that the proof is in the random oracle model and that SE is instantiated with a "one-time pad form" scheme such as CTR mode). In particular:

Theorem 1. *Let $\mathsf{MME}^{\mathrm{f}}_{\pi}$ and $\mathcal{L}_{\mathrm{f}}$ be the scheme and leakage respectively described in Figure 18 using PRF F, symmetric encryption scheme SE, and ideal primitives P_1 and P_2. Then, given adversary $\mathcal{A}$ and simulators $\mathsf{S}_{\mathsf{prf}}, \mathsf{S}_{\mathsf{kp}}$ one can define $\mathcal{S}_{\mathrm{f}}, \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ such that:*

$$\begin{aligned}\mathbf{Adv}^{\mathrm{tv}}_{\mathsf{MME}^{\mathrm{f}}_{\pi},\mathcal{L}_{\mathrm{f}},\mathcal{S}_{\mathrm{f}},\mathsf{P}_1,\mathsf{P}_2}(\mathcal{A}) \leq\ & \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F},\mathsf{P}_1}(\mathcal{A}_1) \\ & + \mathbf{Adv}^{\text{sim-ac-prf}}_{\mathsf{F},\mathsf{S}_{\mathsf{prf}},\mathsf{P}_1}(\mathcal{A}_2) \\ & + \mathbf{Adv}^{\text{sim-ac-kp}}_{\mathsf{SE},\mathsf{S}_{\mathsf{kp}},\mathsf{P}_2}(\mathcal{A}_3).\end{aligned}$$

The security notions used in this reduction, SIM-AC-PRF and SIM-AC-KP, refer to security under adaptive compromise. Both are is are extensions of the PRF and key private (KP) games into this setting, adapted from the definitions givin[21].

In Appendix E, we give the full definitions for the theorem above and show how the proof of Theorem 1 is obtained by modifying the proof of Theorem D.1, in [21], with additional leakage $\mathcal{L}^{\mathrm{f}}_{\pi}$, detailed in Fig. 18.

Other MME techniques from the literature also suffice to construct TV-secure MME. In particular, if one wishes for security outside the RO model, one could use pseudorandomly generated plaintext masks in place of encryption (in the style of CK's scheme Matrix [14]) or generate one token per value (the extension suggested by CJJ+ to MME_{π}).

While a more general result (say, constructing TV-secure schemes from those secure under standard assumptions) would be desirable, we found this tricky to obtain. When moving from the standard security to the TV games, the behavior of the simulators quickly becomes undefined. For example, in the standard security game, the simulator is never asked for tokens before it generates a multimap. So, attempting to give a generic reduction, from standard to TV security seems to require at least one assumption about on the simulator. And, unfortunately, this assumption is not enough to allow a proof to go through, there are many ways a degenerate simulator can break a generic proof.

After some effort, it seems the assumptions one has to make on the simulator (and consequently the scheme) for a generic reduction make the proof almost trivial (i.e. nearly assuming you have a simulator for the TV game). So, instead of presenting a generic reduction, we focus on a specific scheme and leave the target of TV-security open for particular schemes.

5.3 Indirectly Addressed MME SMM

SMM DETAILS. Now we are ready to present the details of the SMM approach. Intuitively, this approach is similar to the LMM one except that we now generate all the recursively accessed tokens using RF MME scheme $\mathsf{MME}_{\mathrm{f}}$ and store the contents of all the $\mathbf{M}_i$ (in LMM) in a single multimap $\mathbf{M}'$ which will also be encrypted with $\mathsf{MME}_{\mathrm{f}}$.

This gives us IA-MME scheme SMM whose algorithms are depicted in Fig. 6. The pseudocode is given in Fig. 6, where $\mathsf{SMM.KS} = \mathsf{MME}_{\mathrm{f}}.\mathsf{KS}$, and $\mathsf{lLen} = \mathsf{len}, \mathsf{vLen} = \mathsf{len} + 1 = \mathsf{MME}_{\mathrm{f}}.\mathsf{tl} + 1$ for $\mathbf{M}_1$.

SMM'S SECURITY. One can notice that on the same multimap and queries, the SMM algorithms and "real world" of the TV-security game generate ED, tk in the exact same way. Additionally, the leakage algorithm

Alg $\mathsf{SMM.Enc}(K, \mathbf{M})$

For $\ell \in \{0,1\}^{\mathsf{len}}$ where $\mathbf{M}[\ell] \neq ()$ do
  $(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell]$
  For $i \in [n]$ do
    If $b_i = 1$ then
      $v_i \leftarrow\!\!\$\ \mathsf{MME_f.Tok}(K, v_i)$
  $\mathbf{M}'[\ell] \leftarrow (b_1 \| v_1, \ldots, b_n \| v_n)$
$ED \leftarrow\!\!\$\ \mathsf{MME_f.Enc}(K, \mathbf{M}')$
Return ED

Alg $\mathsf{SMM.Tok}(K, \ell)$

$tk \leftarrow\!\!\$\ \mathsf{MME_f.Tok}(K, \ell)$; Return tk

Alg $\mathsf{SMM.Eval}(tk, ED)$

$(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathsf{MME_f.Eval}(tk, ED)$
For $i \in [n]$ do
  If $b_i = 1$ then $t_i \leftarrow \mathsf{SMM.Eval}(v_i, ED)$
    else $t_i \leftarrow v_i$
Return $\big((b_1, t_1), \ldots, (b_n, t_n)\big)$

Alg $\mathsf{SMM.Dec}\big(K, ((b_1, t_1), \ldots, (b_n, t_n))\big)$

For $i \in [n]$ do
  If $b_i = 0$ then $u_i \leftarrow \mathsf{MME_f.Dec1}(K, t_i)$
    else $u_i \leftarrow \mathsf{SMM.Dec}(K, t_i)$
Return $(u_1, \ldots, u_n)$

Fig. 6. Algorithms for IA-MME scheme SMM (i.e. StE for IA) using the SMM technique. Here, $\mathsf{MME_f}$ is a response-flexible MME scheme as defined in Section 5.

and simulator in the TV-security game for $\mathsf{MME_f}$ and the semantic security game of SMM are also essentially doing the same this. As such, the following result follows directly from the respective security definitions and a proof is omitted for brevity.

Theorem 2. *Let* SMM *be the IA-MME scheme for* IA *defined in Fig. 6 which uses RF MME scheme* $\mathsf{MME_f}$ *as a primitive. Then, given adversary* $\mathcal{A}$ *we have that:*

$$\mathbf{Adv}^{\mathrm{ss}}_{\mathsf{SMM},\mathcal{L},\mathcal{S}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{tv}}_{\mathsf{MME_f},\mathcal{L},\mathcal{S}}(\mathcal{A}).$$

Discussion. In practice, we expect SMM to be superior to LMM.

In terms of leakage, the only case where the SMM approach is not strictly superior is a degenerate one (e.g. $\mathsf{depth}(\mathbf{M}) = 1$) or a pathological one (with intentionally leaky primitives). Using "standard" MME primitives (e.g. those in Appendix A), the SMM approach avoids leaking the size of each layer and instead just leaks their sum (i.e. the size of its monolithic index). Concretely, given $\mathbf{M}$ as input, the setup leakage (i.e. before any queries are made) in the SMM approach is $\sum_{\ell \in \{0,1\}^{\mathsf{len}}} \#(\mathbf{M}[\ell])$, the number of values in $\mathbf{M}$. However, the LMM leakage is $(n_1, \ldots, n_{\mathsf{depth}(\mathbf{M})})$ where $n_i = \sum_{\ell \in \{0,1\}^{\mathsf{len}}} \#(\mathbf{M}_i[\ell])$ is the number of values in $\mathbf{M}_i$, the multimap indexing all labels of depth i.

We believe that this difference in leakage is significant in a real-world use case. For example, in LabGraph (CK's scheme for searching over web graphs [14]), the difference in leakage means an adversary can distinguish a dense web graph with few vertices from a sparser web graph with more vertices before any queries are made. In Section 6, we demonstrate this significance further in real-world use cases via simulations on realistic data.

SMM is simpler, since the work to prepare $\mathbf{M}$ for encryption is drastically reduced (as evidenced in a much shorter pseudocode of SMM (Fig. 6) compared to LMM (in Appendix B)). There are fewer data structures on the server and differentiated keys on the client side to manage at query time. The one-time cost of performing the security analysis of specific MME schemes is worth the permanent complexity, security and efficiency savings that comes with the SMM approach. And if the analysis it too much, one could choose to use $\mathsf{MME}^{\mathsf{f}}_{\pi}$, which we have already analyzed in Appendix D.

Finally, we expect SMM to be more storage efficient in practice. In LMM, non-uniform constructions need to include depth indicators, a complication that SMM can avoid. In Section 6, we explore in our simulations how this overhead is significant even in relatively low-depth IA-MMEs.

6 Applications and Simulations of IA-MME

The power of the indirect addressing abstraction is its ability to capture and simplify complex StE schemes. In this section, we describe how indirect addressing can be used in a wide range of real-world applications. We then perform simulations on realistic datasets to concretize the efficiency and security gains of using IA-MME, and in particular, the SMM construction.

Searchable Encryption. CGKO first defined Searchable Symmetric Encryption (SSE) as a document storage scheme[15]. Each document is associated with a long payload and a set of keywords. When a keyword is queried, the payloads for all documents associated to that keyword should be returned.

A first insinct might be to use a single multimap associating each keyword to the payloads it should return. This means that SSE is realizable using just an MME scheme. However, this is akin to using the "inlined-payloads" strawman solution presented in Section 3. A better solution uses uniform depth-2 IA-MMEs, with keywords associated to document identifiers at depth-2 and document identifiers associated to payloads at depth-1. We formalize an SSE datatype and the two schemes in Appendix F as $\mathsf{SEdt}, \mathsf{SE1}, \mathsf{SE2}$ respectively. In doing so, we can see the simplicity that comes with the IA-MME primitive since $\mathsf{SE2}$ (which uses indirect addressing) can be expressed with comparable ease to $\mathsf{SE1}$ (which does not).

SQL StE schemes. Various recent works have used the StE framework to capture certain classes of SQL queries, resulting in schemes such as $\mathsf{SPX}, \mathsf{OPX}, \mathsf{FpSj}$ amd PpSj [23,26,12]. The IA-MME primitive greatly simplifies and generalizes the description of such schemes.

To demonstrate this, we define a data type SQLdt which supports relation retrievals and joins over SQL databases. Relation retrievals are queries of the form "**select** * **from** [table]". They let the client retrieve a single table from a database made out of many tables. Joins are queries of the form "**select** * **from** [table1] **join** [table2] **on** [predicate]". These let the client retrieve pairs of rows from the cross product between two tables an accordance with some predicate.

We define three StE schemes for SQLdt. In all three, depth-1 labels associate a unique identifier to the contents of each row in the database. The schemes then use the upper layers in different ways to index which rows (identified by their unique identifier) should be returned in response to each possible query.

Our first two schemes $\mathsf{FP2}, \mathsf{PP2}$ capture the fully and partially precomputed join indexing techniques of CNR, respectively [12]. They both make use of uniform depth-2 IA-MMs in their indexing, with the depth-2 labels associating a query to which rows should be returned. $\mathsf{FP2}$ is essentially equivalent to the schemes $\mathsf{SPX}, \mathsf{OPX}, \mathsf{FpSj}$ when restricted to the same query support, while $\mathsf{PP2}$ is analogously equivalent to PpSj. The key difference between the two schemes is how they index join queries, with the latter indexing rows from the two input tables separately and doing the join computation on the client-side to reduce leakage, bandwidth and server storage. The full details of both schemes can be found in Appendix G. We note that by abstracting out IA-MME, we make the pseudocode of such schemes significantly simpler (compared to prior work). It also allows for better modularity since one can instantiate the scheme using SMM, LMM or any other IA-MME (unlike past work which strictly used multimap chaining).

Our IA-MME abstraction also inspired a new join indexing technique which has not appeared in prior work, which we present now as $\mathsf{PP3}$. The key observation is that real-world joins often make use of every row in one or both of the input tables. In that case, one could "reuse" a relation retrieval query index to index that half of the partially precomputed join thereby saving server storage. This makes the data structure non-uniform and depth-3, demonstrating the need for a generalized IA-MME beyond uniform and depth-2 use-cases.

Simulation Setups. To get an idea for the security and efficiency savings that come with using IA-MME, we run some simulations for the schemes above. For our full simulation methodology and results, see Appendix H.

We gathered two SSE document collections using the papers submitted to the IACR Cryptology ePrint Archive in 2020 and 2021 [18]. For each paper, the PDF submitted to ePrint served as the document payload, while the SSE keyword(s) were those provided by the authors at submission time. The two collections had 3188/3329 distinct keywords, and 1584/1677 document payloads respectively.

For the SQL StE use-case, we generated 1 GB and 10MB databases using the TPC-H benchmark [38]. This database's schema indicates ten relationships between columns from eight relations which a client may wish to perform equijoins on. So we define the StE query class to be exactly these eight relation retrievals and ten joins.

Indirect Addressing saves space. We use our simulations of $\mathsf{SE1}, \mathsf{SE2}$ to emphasize that indirect addressing is much more storage efficient than the "inlined payloads" technique.

In our simulations, we assume that the client pre-processes all the documents such that $\mathsf{len} = 128$. We then measure the number of values in the data-structure (multimap or IA-MM) prior to encryption with $\mathsf{SE1}, \mathsf{SE2}$. With the 2021 documents, $\mathsf{SE1}$ will store $387,841,372$ values in its multimap while $\mathsf{SE2}$ will store 93,888,191 in its IA-MM, a decrease of 75.79%. Likewise, for the 2021 documents, there is a decrease of 76.13% (from 378,100,870 to 90,236,426).

Data	Scheme	LMM				SMM
		$\mathbf{M}_0$	$\mathbf{M}_1$	$\mathbf{M}_2$	$\mathbf{M}_3$	$\mathbf{M}$
2021 ePrint	SE2	–	9.388e7	6.542e3	–	9.389e7
TPC-H (1GB)	FP2	–	7.165e7	1.155e9	–	1.227e9
TPC-H (1GB)	PP2	–	7.165e7	3.349e7	–	1.051e8
TPC-H (1GB)	PP3	18	7.165e7	8.761e6	20	8.041e7

Fig. 7. Selected simulation results computing the sizes of unencrypted data structures when using LMM and SMM. Sizes are computed in blocks of 128-bits (black) or 130-bits (in blue). These demonstrate that SMM leaks less and is more storage efficient.

Assuming encryption changes these sizes only negligibly (true using standard primitives) we see that indirect addressing is useful in a real-world application over naïve solutions thereby justifying our IA-MME formalisms.

SMM's advantages over LMM. We also simulated the size of server-side data structures for the schemes that use IA-MME, to compare the LMM and SMM approaches. Some of our results are in Fig. 7.

We can use this to compare the difference in setup leakage of each scheme under LMM and SMM. Using standard MME primitives (e.g. those in Appendix A), LMM would leak all the sizes of each of its $\mathbf{M}_i$ while SMM leaks just that of $\mathbf{M}$ (i.e. the sum of the $\mathbf{M}_i$ for $i > 0$). Our simulations show that this difference in leakage can be significant in a realistic use-case. For example, with PP3 instantiated with LMM, the adversary learns during setup time how many distinct queries there are (from $\mathbf{M}_0$) and the number of complete joins (from $\mathbf{M}_3$,$\mathbf{M}_0$). The difference in leakage also allows an adversary to easily make inferences about the data, for example, they may be able to deduce the average number of keywords per document or rows per join with just a small amount of auxiliary data. With SMM, the monolithic data structure will hide some of this frequency information making such deductions hard or even impossible.

As mentioned in Section 4, for non-uniform IA-MME, each token stored in the LMM multimaps needs to be accompanied with a depth-indicator. Our simulations show that this is not an insignificant overhead. Even though PP3 only uses depth-3 schemes, this incurs an addition 17522482 bits of storage on top of any additional metadata to support the multiple data structures.

Finally, we note the superiority of PP3 on realistic datasets. This demonstrates that our generalized IA-MME abstraction (beyond uniform depth-2 indirect addressing) allows us to easily tweak StE schemes so they perform better in practice.

References

1. G. Amjad, S. Kamara, and T. Moataz. Breach-resistant structured encryption. Cryptology ePrint Archive, Report 2018/195, 2018. https://eprint.iacr.org/2018/195.
2. G. Asharov, M. Naor, G. Segev, and I. Shahaf. Searchable symmetric encryption: optimal locality in linear space via two-dimensional balanced allocations. In D. Wichs and Y. Mansour, editors, *48th ACM STOC*, pages 1101–1114. ACM Press, June 2016.
3. G. Asharov, G. Segev, and I. Shahaf. Tight tradeoffs in searchable symmetric encryption. Cryptology ePrint Archive, Report 2018/507, 2018. https://eprint.iacr.org/2018/507.
4. A. Authors. Ia-mme simulations. https://github.com/IA-MME-StE/IA-MME-Simulations, 2022.
5. M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. Cryptology ePrint Archive, Report 2006/186, 2006. http://eprint.iacr.org/2006/186.
6. M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006.
7. R. Bost. Sophos - forward secure searchable encryption. Cryptology ePrint Archive, Report 2016/728, 2016. http://eprint.iacr.org/2016/728.
8. R. Bost, B. Minaud, and O. Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *ACM CCS 2017*, pages 1465–1482. ACM Press, Oct. / Nov. 2017.
9. D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 2015*, pages 668–679. ACM Press, Oct. 2015.

10. D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS 2014*. The Internet Society, Feb. 2014.
11. D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for Boolean queries. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 353–373. Springer, Heidelberg, Aug. 2013.
12. D. Cash, R. Ng, and A. Rivkin. Improved structured encryption for sql databases via hybrid indexing. In *International Conference on Applied Cryptography and Network Security*, pages 480–510. Springer, 2021.
13. D. Cash and S. Tessaro. The locality of searchable symmetric encryption. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 351–368. Springer, Heidelberg, May 2014.
14. M. Chase and S. Kamara. Structured encryption and controlled disclosure. In M. Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 577–594. Springer, Heidelberg, Dec. 2010.
15. R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In A. Juels, R. N. Wright, and S. De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 79–88. ACM Press, Oct. / Nov. 2006.
16. I. Demertzis, D. Papadopoulos, and C. Papamanthou. Searchable encryption with optimal locality: Achieving sublogarithmic read efficiency. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 371–406. Springer, Heidelberg, Aug. 2018.
17. S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner. Rich queries on encrypted data: Beyond exact matches. In *European symposium on research in computer security*, pages 123–145. Springer, 2015.
18. I. A. for Cryptologic Research. Cryptology ePrint archive. `https://eprint.iacr.org/`, 2022.
19. E.-J. Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. `http://eprint.iacr.org/2003/216`.
20. M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS 2012*. The Internet Society, Feb. 2012.
21. J. Jaeger and N. Tyagi. Handling adaptive compromise for practical encryption schemes. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 3–32. Springer, Heidelberg, Aug. 2020.
22. S. Kamara and T. Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In J.-S. Coron and J. B. Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 94–124. Springer, Heidelberg, Apr. / May 2017.
23. S. Kamara and T. Moataz. SQL on structurally-encrypted databases. In T. Peyrin and S. Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 149–180. Springer, Heidelberg, Dec. 2018.
24. S. Kamara and T. Moataz. Computationally volume-hiding structured encryption. In Y. Ishai and V. Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 183–213. Springer, Heidelberg, May 2019.
25. S. Kamara, T. Moataz, and O. Ohrimenko. Structured encryption and leakage suppression. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 339–370. Springer, Heidelberg, Aug. 2018.
26. S. Kamara, T. Moataz, S. Zdonik, and Z. Zhao. An optimal relational database encryption scheme. Cryptology ePrint Archive, Report 2020/274, 2020. `https://eprint.iacr.org/2020/274`.
27. S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *International conference on financial cryptography and data security*, pages 258–274. Springer, 2013.
28. S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In T. Yu, G. Danezis, and V. D. Gligor, editors, *ACM CCS 2012*, pages 965–976. ACM Press, Oct. 2012.
29. E. S. Lab. The clusion library. `https://github.com/encryptedsystems/Clusion`, 2020.
30. M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 2015*, pages 644–655. ACM Press, Oct. 2015.
31. M. Naveed, M. Prabhakaran, and C. A. Gunter. Dynamic searchable encryption via blind storage. In *2014 IEEE Symposium on Security and Privacy*, pages 639–654. IEEE Computer Society Press, May 2014.
32. S. Patel, G. Persiano, and K. Yeo. Private stateful information retrieval. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 2018*, pages 1002–1019. ACM Press, Oct. 2018.
33. S. Patel, G. Persiano, and K. Yeo. Lower bounds for encrypted multi-maps and searchable encryption in the leakage cell probe model. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 433–463. Springer, Heidelberg, Aug. 2020.
34. D. Pouliot and C. V. Wright. The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 2016*, pages 1341–1352. ACM Press, Oct. 2016.
35. D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55. IEEE Computer Society Press, May 2000.

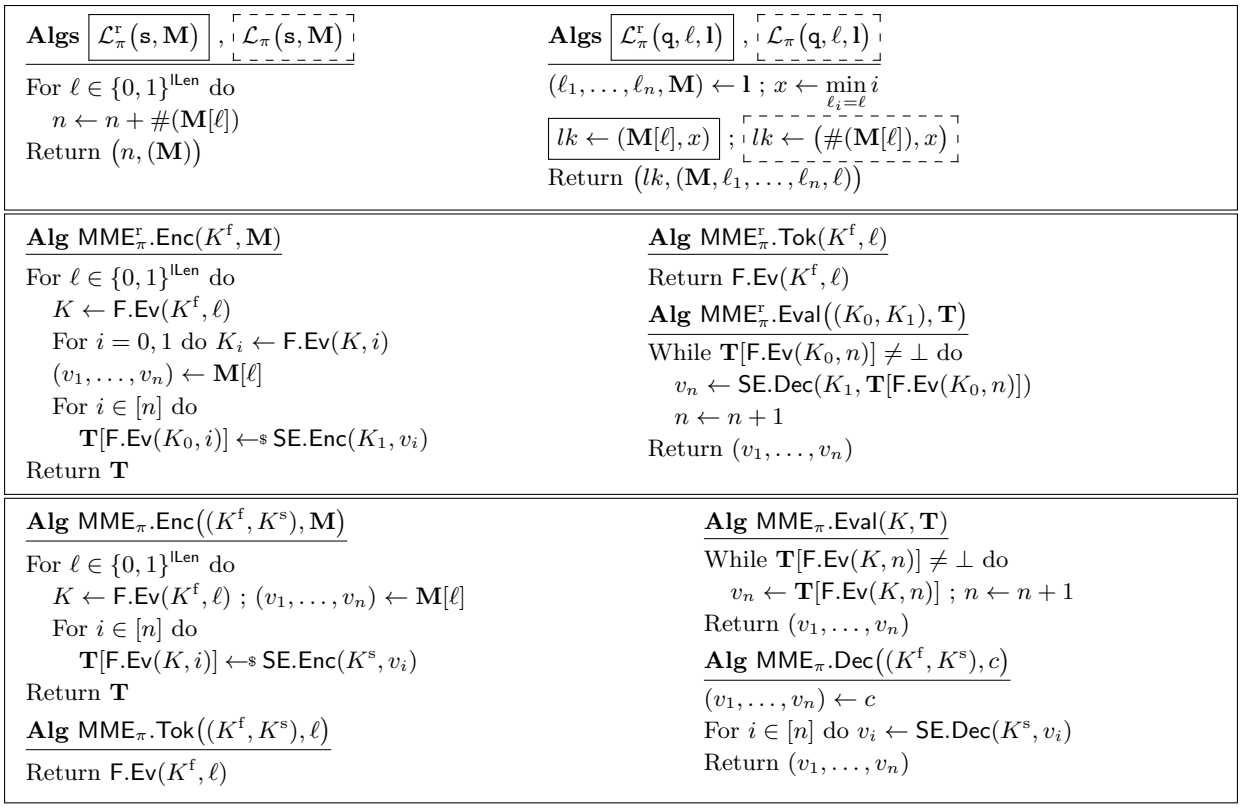

Fig. 8. "Standard" leakage for MME schemes that are $\boxed{\text{RR}}$ and $\boxed{\text{not RR}}$ (top), an example of each such scheme (middle), and an RF MME scheme with analogous leakage profile to $\mathsf{MME}^{\mathrm{r}}_{\pi}$.

36. E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *NDSS 2014*. The Internet Society, Feb. 2014.
37. Transaction Processing Performance Council. Tpc benchmarktm h (decision support) standard specification revision 3.0.1. `https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v3.0.1.pdf`, 2022.
38. Transaction Processing Performance Council. Tpc download current specs/source. `https://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp`, 2023.
39. C. Wang, N. Cao, J. Li, K. Ren, and W. Lou. Secure ranked keyword search over encrypted cloud data. In *2010 IEEE 30th international conference on distributed computing systems*, pages 253–262. IEEE, 2010.
40. Y. Zhang, J. Katz, and C. Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In T. Holz and S. Savage, editors, *USENIX Security 2016*, pages 707–720. USENIX Association, Aug. 2016.

A Standard MME Leakages and Example MME Schemes.

In Section 2.1 we discuss "standard" MME schemes and their leakage profiles. For RR MME, this leakage profile is $\mathcal{L}^{\mathrm{r}}_{\pi}$ and $\mathsf{MME}^{\mathrm{r}}_{\pi}$ achieves it. For non-RR MME, this leakage profile is $\mathcal{L}_{\pi}$ and MME_{π} below achieves it. For RF MME, $\mathsf{MME}^{\mathrm{f}}_{\pi}$ achieves the the RF leakage analog (see Section 5.1) of $\mathsf{MME}^{\mathrm{r}}_{\pi}$. The above MME and leakage algorithms are given in Fig. 8. All of these schemes were inspired by $\prod_{\mathrm{bas}}$ from [10], with minor modifications to allow for the different response types and keep tokens compact. Each of these schemes achieve (adaptive) semantic security with respect to their leakage algorithms in the RO model (as was done in [10,21]). The only caveat to this being that SE in the RR and RF variants are of the "one-time pad style" (e.g. CTR mode, or using a PRG mask).

The primitives used are symmetric encryption scheme SE and function family F. We require that $\mathsf{SE.KS} = \{0,1\}^{\mathsf{F.ol}} = \mathsf{F.KS}$ in $\mathsf{MME}^{\mathrm{r}}_{\pi}$. Note that $\mathsf{MME}_{\pi}.\mathsf{KS} = \mathsf{SE.KS} \times \mathsf{F.KS}$ and $\mathsf{MME}^{\mathrm{r}}_{\pi}.\mathsf{KS} = \mathsf{F.KS}$.

Note that $\mathsf{MME}^{\mathrm{r}}_{\pi}$ has content oblivious leakage (as defined in Section 4). The setup leakage is the number of values in $\mathbf{T}$ which is constant for homomorphic multimaps because SE's ciphertext length function is

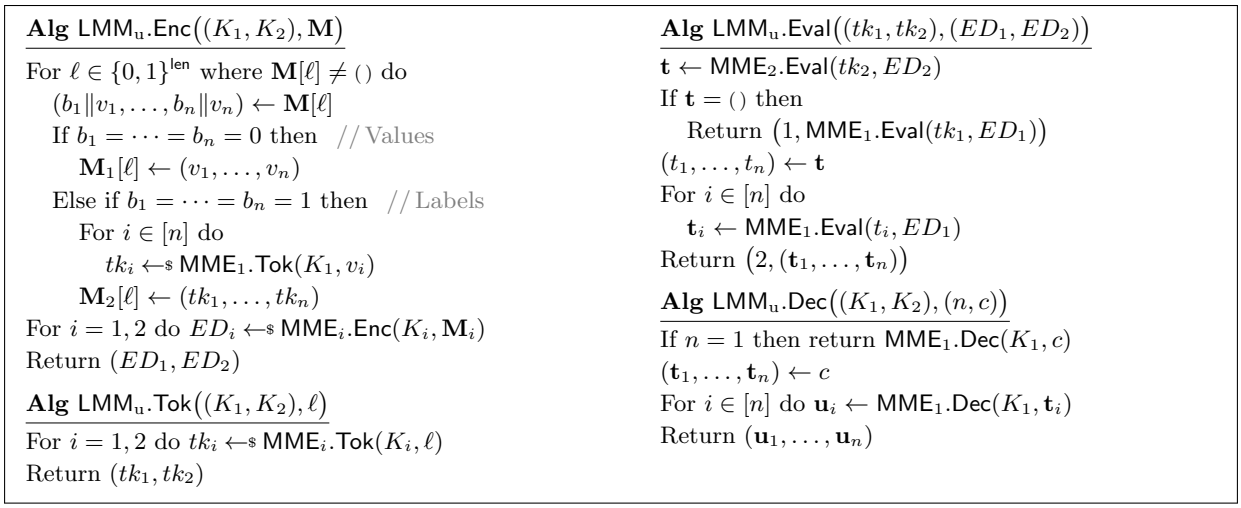

Fig. 9. Algorithms for IA-MME scheme $\mathsf{LMM_u}$ for uniform depth-2 IA-MMs (i.e. StE for UIA where $\mathsf{UIA.dp} = \mathsf{depth}(\mathbf{M}) = 2$) using the LMM technique. Here, $\mathsf{MME_1}, \mathsf{MME_2}$ are MME schemes and $\mathsf{MME_2}$ is RR. See Appendix B for a more general variant of $\mathsf{LMM_u}$ which supports non-uniform, arbitrary depth $\mathbf{M}$.

message independent. The query leakage (apart from the query response) is the query equality pattern (which is independent of $\mathbf{M}$).

B LMM details

LMM approach for uniform, depth-2 IA-MMs. The full pseudocode for $\mathsf{LMM_u}$ is given in Fig. 9. Note that $\mathsf{LMM_u.KS} = \mathsf{MME_1.KS} \times \mathsf{MME_2.KS}$. We additionally assume that $\mathsf{lLen} = \mathsf{vLen} = \mathsf{len} = \mathsf{MME_1.tl}$ (for $\mathbf{M}_1, \mathbf{M}_2$), and that $\mathsf{depth}(\mathbf{M}) = 2$ to avoid degeneracy.

Proving security of the LMM approach. Proving the security of the above IA-MME schemes is more tricky than one might initially think. We demonstrate this by walking through the seemingly straightforward security proof for $\mathsf{LMM_u}$ then pointing out a issue therein. Note that this applies to LMM too since $\mathsf{LMM_u}$ is a special case of it.

Intuitively, this proof would involve reducing the security of $\mathsf{LMM_u}$ to that of $\mathsf{MME_1}, \mathsf{MME_2}$. Since these are all StE schemes, our proof would construct $\mathcal{L}_{\mathrm{lm}}, \mathcal{S}_{\mathrm{lm}}$, the leakage algorithm and simulator for $\mathsf{LMM_u}$, from those associated to $\mathsf{MME_1}, \mathsf{MME_2}$ (i.e. $\mathcal{L}_1, \mathcal{S}_1, \mathcal{L}_2, \mathcal{S}_2$). This intuitive proof works when $\mathcal{L}_1, \mathcal{L}_2$ are the"standard" leakage profile (as discussed in Section 2.1 and Appendix A) but falls through under some pathological leakage algorithms and simulators.

In Fig. 10 we give the intuitive $\mathcal{L}_{\mathrm{lm}}, \mathcal{S}_{\mathrm{lm}}$ inspired by those used in prior work. Intuitively, $\mathcal{L}_{\mathrm{lm}}(\mathsf{s}, \mathbf{M})$ will construct $\mathbf{M}_1, \mathbf{M}_2$ in the same way as $\mathsf{LMM_u.Enc}$ (using a random $K_1 \in \mathsf{MME_1.KS}$)[4] then return their setup leakages under the respective schemes that encrypt them. The query leakage for ℓ includes the leakage incurred for querying ℓ under both schemes, and also the leakage associated with querying $\mathsf{MME_1}$ with any ℓ_i that would be returned by $\mathsf{MME_2}$ (in the case of a depth-2 query).

Meanwhile, $\mathcal{S}_{\mathrm{lm}}$'s algorithms channel their inputs into the respective simulators and compose their outputs in the natural way. During the setup phase we have no problems simulating data structures ED_1, ED_2. However, consider what happens in the query phase when a depth-2 query is made. Recall that $\mathsf{MME_2}$ is response-revealing and so lk_2 takes the form $(\mathbf{M}_2[\ell], lk)$ for some lk where $\mathbf{M}_2$ is as it was constructed in $\mathcal{L}_{\mathrm{lm}}$. Before this leakage is passed to $\mathcal{S}_2$, the first argument is rewritten with s – the tokens returned by $\mathcal{S}_1$. We call this the "leakage rewriting" trick. It is done so that $\mathsf{MME_2.Eval}(tk, ED_2)$ will return tokens which $\mathsf{MME_1.Eval}$ can evaluate. This switch is necessary because the tokens in $\mathbf{M}_2[\ell]$ are generated with K_2 selected

[4] An alternative approach to $\mathcal{L}_{\mathrm{lm}}$ is to generate the tokens tk_i in $\mathbf{M}_2$ using $\mathcal{S}_1$. However, this leads to the same type of issues. If we instantiate a new instance of $\mathcal{S}_1$ outside of the leakage function, it is possible that the simulator is randomized and the tokens it generates are different than the instance inside $\mathcal{L}_{\mathrm{lm}}$, so we will once again need to replace the tokens and employ content obliviousness to get the proof to go through.

Alg $\mathcal{L}_{\mathrm{lm}}(\mathtt{s}, \mathbf{M})$

$K_1 \leftarrow\!\!\$\ \mathsf{MME}_1.\mathsf{KS}$
For $\ell \in \{0,1\}^{\mathsf{len}}$ where $\mathbf{M}[\ell] \neq ()$ do
  $(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell]$
  If $b_1 = \cdots = b_n = 0$ then $\mathbf{M}_1[\ell] \leftarrow (v_1, \ldots, v_n)$
  Else if $b_1 = \cdots = b_n = 1$ then
    For $i \in [n]$ do $tk_i \leftarrow\!\!\$\ \mathsf{MME}_1.\mathsf{Tok}(K_1, v_i)$
    $\mathbf{M}_2[\ell] \leftarrow (tk_1, \ldots, tk_n)$
$(\mathcal{L}_1, St_1) \leftarrow\!\!\$\ \mathcal{L}_1(\mathtt{s}, \mathbf{M}_1)$
$(\mathcal{L}_2, St_2) \leftarrow\!\!\$\ \mathcal{L}_2(\mathtt{s}, \mathbf{M}_2)$
Return $\big((lk_1, lk_2), (St_1, St_2, \mathbf{M})\big)$

Alg $\mathcal{L}_{\mathrm{lm}}(\mathtt{q}, \ell, (St_1, St_2, \mathbf{M}))$

$(lk_1, St_1) \leftarrow\!\!\$\ \mathcal{L}_1(\mathtt{q}, \ell, St_1)$
$(lk_2, St_2) \leftarrow\!\!\$\ \mathcal{L}_2(\mathtt{q}, \ell, St_2)$
If $\mathbf{M}[\ell] = (1\|\ell_1, \ldots, 1\|\ell_n)$ then
  For $i \in [n]$ do
    $(lk_i, St_1) \leftarrow\!\!\$\ \mathcal{L}_1(\mathtt{q}, \ell_i, St_1)$
  $\mathbf{lk} \leftarrow (lk_1, \ldots, lk_n)$
Else $\mathbf{lk} \leftarrow ()$
Return $\big((lk_1, lk_2, \mathbf{lk}), (St_1, St_2, \mathbf{M})\big)$

Alg $\mathcal{S}_{\mathrm{lm}}(\mathtt{s}, (lk_1, lk_2))$

$(ED_1, St_1') \leftarrow\!\!\$\ \mathcal{S}_1(\mathtt{s}, lk_1)$
$(ED_2, St_2') \leftarrow\!\!\$\ \mathcal{S}_2(\mathtt{s}, lk_2)$
$(ED_1, ED_2) \leftarrow ED$
Return $\big(ED, (St_1', St_2')\big)$

Alg $\mathcal{S}_{\mathrm{lm}}(\mathtt{q}, (lk_1, lk_2, \mathbf{lk}), (St_1', St_2'))$

If $\mathbf{lk} = (lk_1', \ldots, lk_n')$ then
  For $i \in [n]$ do $(tk_i, St_1') \leftarrow\!\!\$\ \mathcal{S}_1(\mathtt{q}, lk_i', St_1')$
  $s \leftarrow (tk_1, \ldots, tk_n)$; $(s', lk) \leftarrow lk_2$; $lk_2 \leftarrow (s, lk)$
$(tk_1, St_1') \leftarrow\!\!\$\ \mathcal{S}_1(\mathtt{q}, lk_1, St_1')$
$(tk_2, St_2') \leftarrow\!\!\$\ \mathcal{S}_2(\mathtt{q}, lk_2, St_2')$
Return $\big((tk_1, tk_2), (St_1', St_2')\big)$

Fig. 10. Leakage algorithm (left) and simulator (right) for $\mathsf{LMM}_{\mathrm{u}}$, the IA-MME scheme for uniform depth-2 IA-MMs. Here, $\mathsf{MME}_1, \mathsf{MME}_2$ are MME schemes and MME_2 is RR. The leakage algorithm and simulator of MME_i is $\mathcal{L}_{\mathrm{i}}, \mathcal{S}_{\mathrm{i}}$ respectively.

in the leakage algorithm. $\mathcal{S}_1$ has no knowledge of this key so we can expect that these tokens are unlikely to "work" with the simulated ED_2. However, this switch also means that the behavior of $\mathcal{S}_2$ in $\mathcal{S}_{\mathrm{lm}}$ is no longer well defined because MME_2's semantic security only promises that $\mathcal{S}_2(\mathtt{q}, (\mathbf{M}_2[\ell], lk), St_2')$ returns a token, not $\mathcal{S}_2(\mathtt{q}, (s, lk), St_2')$.

Proof of Theorem 3. We can now state and prove $\mathsf{LMM}_{\mathrm{u}}$'s security under the content obliviousness assumption:

Theorem 3. *Let* $\mathsf{LMM}_{\mathrm{u}}$ *be the IA-MME scheme for* UIA *defined in Fig. 9 using MME primitives* $\mathsf{MME}_1, \mathsf{MME}_2$. *Let* $\mathcal{L}_{\mathrm{lm}}, \mathcal{S}_{\mathrm{lm}}$ *be as defined in Fig. 10, where* $\mathcal{L}_{\mathrm{i}}, \mathcal{S}_{\mathrm{i}}$ *are the leakage algorithm and simulator for* MME_i *(and* $\mathcal{L}_2$ *is content oblivious). Then given adversary* $\mathcal{A}$ *one can define* $\mathcal{A}_1, \mathcal{A}_2$ *such that:*

$$\mathbf{Adv}^{\mathrm{ss}}_{\mathsf{LMM}_{\mathrm{u}}, \mathcal{L}_{\mathrm{lm}}, \mathcal{S}_{\mathrm{lm}}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{ss}}_{\mathsf{MME}_1, \mathcal{L}_1, \mathcal{S}_1}(\mathcal{A}_1) + \mathbf{Adv}^{\mathrm{ss}}_{\mathsf{MME}_2, \mathcal{L}_2, \mathcal{S}_2}(\mathcal{A}_2).$$

Proof. The adversaries $\mathcal{A}_1, \mathcal{A}_2$ are described in Fig. 11. To aid this proof, we define $\mathrm{G}_0, \mathrm{G}_1, \mathrm{G}_2$ in Fig. 12 where the adversary $\mathcal{A}$ plays different hybrid games. Let b, b_1, b_2 be the challenge bits in $\mathrm{G}^{\mathrm{ss}}_{\mathsf{LMM}_{\mathrm{u}}, \mathcal{L}_{\mathrm{lm}}, \mathcal{S}_{\mathrm{lm}}}(\mathcal{A})$, $\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME}_1, \mathcal{L}_1, \mathcal{S}_1}(\mathcal{A}_1)$ and $\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME}_2, \mathcal{L}_2, \mathcal{S}_2}(\mathcal{A}_2)$ respectively. Intuitively, each $\mathcal{A}_i$ is playing the semantic security game for MME_i and runs $\mathcal{A}$ – $\mathcal{A}_1$ simulates the encryption of $\mathbf{M}_2$ using a simulator while $\mathcal{A}_2$ uses the MME_1 algorithms to do likewise for $\mathbf{M}_1$.

The outline of the proof is to first transition from the "real-world" into a game where the encryption of $\mathbf{M}_2$ is simulated. Then, transition to a game where token values for $\mathbf{M}_2$ are generated lazily (where we invoke obliviousness). And finally, transition again into a game where the encryption of both $\mathbf{M}_1$ and $\mathbf{M}_2$ are simulated, which matches the "ideal-world."

Our theorem statement will follow from a series of claims together with the triangle inequality. Specifically, we use the following

$$\begin{aligned} &|\Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{LMM}_{\mathrm{u}}, \mathcal{L}_{\mathrm{lm}}, \mathcal{S}_{\mathrm{lm}}}(\mathcal{A}) | b = 1] - Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{LMM}_{\mathrm{u}}, \mathcal{L}_{\mathrm{lm}}, \mathcal{S}_{\mathrm{lm}}}(\mathcal{A}) | b = 0]| \\ &\leq |\Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1]| + |\Pr[\mathrm{G}_1] - \Pr[\mathrm{G}_2]| + |\Pr[\mathrm{G}_2] - \Pr[\mathrm{G}_3]|. \end{aligned}$$

The first and last terms give the theorem statement, and the difference between G_1 and G_2 is 0 by our obliviousness assumption.

Adversary $\mathcal{A}_1(\mathsf{s})$

$K_1 \leftarrow\!\!\$\ \mathsf{MME}_1.\mathsf{KS}$; $(\mathbf{M}, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{s})$
For $\ell \in \{0,1\}^{\mathsf{len}}$ where $\mathbf{M}[\ell] \neq ()$ do
  $(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell]$
  If $b_1 = \cdots = b_n = 0$ then
    $\mathbf{M}_1[\ell] \leftarrow (v_1, \ldots, v_n)$
  Else if $b_1 = \cdots = b_n = 1$ then
    For $i \in [n]$ do $tk_i \leftarrow\!\!\$\ \mathsf{MME}_1.\mathsf{Tok}(K_1, v_i)$
    $\mathbf{M}_2[\ell] \leftarrow (tk_1, \ldots, tk_n)$
$(lk_2, St_2) \leftarrow\!\!\$\ \mathcal{L}_2\big(\mathsf{s}, \mathbf{M}_2\big)$
$(ED_2, St_2') \leftarrow\!\!\$\ \mathcal{S}_2\big(\mathsf{s}, lk_2\big)$
Return $\big(\mathbf{M}_1, (ED_2, St_{\mathrm{a}})\big)$

Adversary $\mathcal{A}_1^{\textsc{Tok}}(\mathsf{q}, ED_1, (ED_2, St_{\mathrm{a}}))$

$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Tok}^*}(\mathsf{q}, (ED_1, ED_2), St_{\mathrm{a}})$; Return b'

Oracle $\textsc{Tok}^*(\ell)$

$(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell]$
$(lk_2, St_2) \leftarrow\!\!\$\ \mathcal{L}_2\big(\mathsf{q}, \ell, St_2\big)$
If $b_1 = \cdots = b_n = 1$ then
  For $i \in [n]$ do $tk_i \leftarrow\!\!\$\ \textsc{Tok}(v_i)$
  $(s', lk) \leftarrow lk_2$; $lk_2 \leftarrow \big((tk_1, \ldots, tk_n), lk\big)$
$(tk_2, St_2') \leftarrow\!\!\$\ \mathcal{S}_2\big(\mathsf{q}, lk_2, St_2'\big)$
$tk_1 \leftarrow\!\!\$\ \textsc{Tok}(\ell)$; Return (tk_1, tk_2)

Adversary $\mathcal{A}_2(\mathsf{s})$

$K_1 \leftarrow\!\!\$\ \mathsf{MME}_1.\mathsf{KS}$
$(\mathbf{M}, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{s})$
For $\ell \in \{0,1\}^{\mathsf{len}}$ if $\mathbf{M}[\ell] \neq ()$ do
  $(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell]$
  If $b_1 = \cdots = b_n = 0$ then
    $\mathbf{M}_1[\ell] \leftarrow (v_1, \ldots, v_n)$
  Else if $b_1 = \cdots = b_n = 1$ then
    For $i \in [n]$ do
      $tk_i \leftarrow\!\!\$\ \mathsf{MME}_1.\mathsf{Tok}(K_1, v_i)$
    $\mathbf{M}_2[\ell] \leftarrow (tk_1, \ldots, tk_n)$
$ED_1 \leftarrow\!\!\$\ \mathsf{MME}_1.\mathsf{Enc}(K_1, \mathbf{M}_1)$
Return $\big(\mathbf{M}_2, (K_1, ED_1, St_{\mathrm{a}})\big)$

Adversary $\mathcal{A}_2^{\textsc{Tok}}(\mathsf{q}, ED_2, St_{\mathrm{a}}')$

$(K_1, ED_1, St_{\mathrm{a}}) \leftarrow St_{\mathrm{a}}'$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Tok}^*}(\mathsf{q}, (ED_1, ED_2), St_{\mathrm{a}})$
Return b'

Oracle $\textsc{Tok}^*(\ell)$

$tk_1 \leftarrow\!\!\$\ \mathsf{MME}_1.\mathsf{Tok}(K_1, \ell)$
$tk_2 \leftarrow\!\!\$\ \textsc{Tok}(\ell)$
Return (tk_1, tk_2)

Fig. 11. Adversaries used in the proof of Theorem 3.

Notice that all the encrypted data structures and tokens in G_0 are generated using the MME primitives. This is the same as what happens in the "real world" of $\mathrm{G}^{\mathrm{ss}}_{\mathsf{LMM}_{\mathrm{u}},\mathcal{L}_{\mathrm{lm}},\mathcal{S}_{\mathrm{lm}}}(\mathcal{A})$. It is also equivalent to the "real world" experienced by $\mathcal{A}_2$. Therefore,

$$\begin{aligned}\Pr[\mathrm{G}_0] &= \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{LMM}_{\mathrm{u}},\mathcal{L}_{\mathrm{lm}},\mathcal{S}_{\mathrm{lm}}}(\mathcal{A}) | b = 1] \\ &= \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME}_2,\mathcal{L}_2,\mathcal{S}_2}(\mathcal{A}_2) | b_2 = 1].\end{aligned}$$

In G_1, the tokens for depth-1 queries and ED_1 are generated using the MME primitive while depth-2 tokens and ED_2 are simulated. This is what happens in the "ideal world" of $\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME}_2,\mathcal{L}_2,\mathcal{S}_2}(\mathcal{A}_2)$, which generates tokens for MME_2 via a simulator. So,

$$\Pr[\mathrm{G}_1] = \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME}_2,\mathcal{L}_2,\mathcal{S}_2}(\mathcal{A}_2) | b_2 = 0].$$

Next, we observe that G_2 captures the "real world" of $\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME}_1,\mathcal{L}_1,\mathcal{S}_1}(\mathcal{A}_1)$, because it recursively replaces the tokens prior to simulating the token for MME_2, just like $\mathcal{A}_1$. This establishes that,

$$\Pr[\mathrm{G}_2] = \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME}_1,\mathcal{L}_1,\mathcal{S}_1}(\mathcal{A}_1) | b_1 = 1].$$

We now invoke perfect obliviousness to claim

$$|\Pr[\mathrm{G}_2] - \Pr[\mathrm{G}_1]| \leq \max_{\mathcal{A}} \Pr[\mathrm{G}^{\mathrm{obliv}}_{\mathcal{L}_2}(\mathcal{A})] = 0,$$

because the values in the leakage output are replaced in equal number of values from $\textsc{Tok}$. If the second part of lk_2 is the same between these games, then everything in the games is identically distributed. Therefore, the difference between these games is bounded by the probability that the second part of lk_2 differs. This is exactly the probability captured by $\mathrm{G}^{\mathrm{obliv}}_{\mathcal{L}_2}$. However, by assuming perfect content obliviousness, any two homomorphic multimaps will have this part of the leakage be exactly the same and because we replace s with the same number of inputs, we are effectively comparing leakage outputs on two homomorphic multimaps.

Game G_0

For $i = 1, 2$ do $K_i \leftarrow\!\!\$\ \mathsf{MME}_i.\mathsf{KS}$
$(\mathbf{M}, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathtt{s})$
For $\ell \in \{0,1\}^{\mathsf{len}}$ where $\mathbf{M}[\ell] \neq ()$ do
  $(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell]$
  If $b_1 = \cdots = b_n = 0$
    then $\mathbf{M}_1[\ell] \leftarrow (v_1, \ldots, v_n)$
  Else if $b_1 = \cdots = b_n = 1$ then
    For $i \in [n]$ do
      $tk_i \leftarrow\!\!\$\ \mathsf{MME}_1.\mathsf{Tok}(K_1, v_i)$
    $\mathbf{M}_2[\ell] \leftarrow (tk_1, \ldots, tk_n)$
For $i = 1, 2$ do
  $ED_i \leftarrow\!\!\$\ \mathsf{MME}_i.\mathsf{Enc}(K_i, \mathbf{M}_i)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Tok}}(\mathtt{q}, (ED_1, ED_2), St_{\mathrm{a}})$
Return $b' = 1$

Oracle $\mathrm{Tok}(\ell)$

$tk_1 \leftarrow\!\!\$\ \mathsf{MME}_1.\mathsf{Tok}(K_1, \ell)$
$tk_2 \leftarrow\!\!\$\ \mathsf{MME}_2.\mathsf{Tok}(K_2, \ell)$
Return (tk_1, tk_2)

Game G_1, $\boxed{\mathrm{G}_2}$

$K_1 \leftarrow\!\!\$\ \mathsf{MME}_1.\mathsf{KS}$; $(\mathbf{M}, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathtt{s})$
For $\ell \in \{0,1\}^{\mathsf{len}}$ where $\mathbf{M}[\ell] \neq ()$ do
  $(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell]$
  If $b_1 = \cdots = b_n = 0$ then
    $\mathbf{M}_1[\ell] \leftarrow (v_1, \ldots, v_n)$
  Else if $b_1 = \cdots = b_n = 1$ then
    For $i \in [n]$ do
      $tk_i \leftarrow\!\!\$\ \mathsf{MME}_1.\mathsf{Tok}(K_1, v_i)$
    $\mathbf{M}_2[\ell] \leftarrow (tk_1, \ldots, tk_n)$
$ED_1 \leftarrow\!\!\$\ \mathsf{MME}_1.\mathsf{Enc}(K_1, \mathbf{M}_1)$
$(lk_2, St_2) \leftarrow\!\!\$\ \mathcal{L}_2(\mathtt{s}, \mathbf{M}_2)$
$(ED_2, St_2') \leftarrow\!\!\$\ \mathcal{S}_2(\mathtt{s}, lk_2)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Tok}}(\mathtt{q}, (ED_1, ED_2), St_{\mathrm{a}})$
Return $b' = 1$

Oracle $\mathrm{Tok}(\ell)$

$(lk_2, St_2) \leftarrow\!\!\$\ \mathcal{L}_2(\mathtt{q}, \ell, St_2)$
$\boxed{\begin{array}{l}(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell] \\ \text{If } b_1 = \cdots = b_n = 1 \text{ then} \\ \quad \text{For } i \in [n] \text{ do} \\ \quad\quad tk_i \leftarrow\!\!\$\ \mathrm{Tok}(v_i) \\ \quad s \leftarrow (tk_1, \ldots, tk_n) \\ \quad (s', lk) \leftarrow lk_2\ ;\ lk_2 \leftarrow (s, lk)\end{array}}$
$tk_1 \leftarrow\!\!\$\ \mathsf{MME}_1.\mathsf{Tok}(K_1, \ell)$
$(tk_2, St_2') \leftarrow\!\!\$\ \mathcal{S}_2(\mathtt{q}, lk_2, St_2')$
Return (tk_1, tk_2)

Game G_3

$K_1 \leftarrow\!\!\$\ \mathsf{MME}_1.\mathsf{KS}$; $(\mathbf{M}, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathtt{s})$
For $\ell \in \{0,1\}^{\mathsf{len}}$ where $\mathbf{M}[\ell] \neq ()$ do
  $(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell]$
  If $b_1 = \cdots = b_n = 0$ then $\mathbf{M}_1[\ell] \leftarrow (v_1, \ldots, v_n)$
  Else if $b_1 = \cdots = b_n = 1$ then
    For $i \in [n]$ do $tk_i \leftarrow\!\!\$\ \mathsf{MME}_1.\mathsf{Tok}(K_1, v_i)$
    $\mathbf{M}_2[\ell] \leftarrow (tk_1, \ldots, tk_n)$
For $i = 1, 2$ do
  $(lk_i, St_i) \leftarrow\!\!\$\ \mathcal{L}_i(\mathtt{s}, \mathbf{M}_i)$; $(ED_i, St_i') \leftarrow\!\!\$\ \mathcal{S}_i(\mathtt{s}, lk_i)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Tok}}(\mathtt{q}, (ED_1, ED_2), St_{\mathrm{a}})$
Return $b' = 1$

Oracle $\mathrm{Tok}(\ell)$

$(lk_1, St_1) \leftarrow\!\!\$\ \mathcal{L}_1(\mathtt{q}, \ell, St_1)$
$(lk_2, St_2) \leftarrow\!\!\$\ \mathcal{L}_2(\mathtt{q}, \ell, St_2)$
$(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell]$
If $b_1 = \cdots = b_n = 1$ then
  For $i \in [n]$ do
    $tk_i \leftarrow\!\!\$\ \mathrm{Tok}(v_i)$
  $s \leftarrow (tk_1, \ldots, tk_n)$
  $(s', lk) \leftarrow lk_2\ ;\ lk_2 \leftarrow (s, lk)$
$(tk_1, St_1') \leftarrow\!\!\$\ \mathcal{S}_1(\mathtt{q}, lk_1, St_1')$
$(tk_2, St_2') \leftarrow\!\!\$\ \mathcal{S}_2(\mathtt{q}, lk_2, St_2')$
Return (tk_1, tk_2)

Fig. 12. Games $\mathrm{G}_0, \mathrm{G}_1, \mathrm{G}_2, \mathrm{G}_3$ used in the proof of Theorem 3.

Finally, we establish

$$\begin{aligned}\Pr[\mathrm{G}_3] &= \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{MME}_1,\mathcal{L}_1,\mathcal{S}_1}(\mathcal{A}_1)|b_1 = 0] \\ &= \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{LMM}_\mathrm{u},\mathcal{L}_\mathrm{lm},\mathcal{S}_\mathrm{lm}}(\mathcal{A})|b = 0].\end{aligned}$$

In G_3 the tokens in $\mathbf{M}_2$ during the setup phase are generated with MME_1, but just like in $\mathcal{S}_\mathrm{lm}$, they will be replaced with simulator-generated ones when the adversary queries the associated label. It also replaces the values with updated tokens just like in both $\mathcal{S}_\mathrm{lm}$ and $\mathcal{A}_1$.

LMM DETAILS. In Section 4, we described how LMM_u, the scheme supporting uniform depth-2 IA-MMs could be extended to support all IA-MMs. The resultant scheme is LMM, an StE scheme for IA, whose pseudocode we present in Fig. 13. As in Section 4, we will assume that all $\mathbf{M}$ being encrypted have $\mathsf{depth}(\mathbf{M}) \geq 2$ to avoid degeneracy. Here, we let $D = \mathsf{IA.dp}$ and have $D+1$ MME schemes $\mathsf{MME}_0, \ldots, \mathsf{MME}_D$, which are all RR except MME_1. As discussed in Section 4, we assume that the keys for the MME primitives can be generated using function family F (i.e. $\{0,1\}^{\mathsf{F.ol}} = \mathsf{MME}_i.\mathsf{KS}$ for $i \in \{0, \ldots, D\}$). Also, we assume $\mathsf{lLen} = \mathsf{len}$, $\mathsf{len} = \mathsf{MME}_1.\mathsf{tl} = \cdots = \mathsf{MME}_{D-1}.\mathsf{tl}$ and that $\mathsf{vLen} = \mathsf{len} + \lceil \log_2(D) \rceil$.

We state the security of LMM with the below theorem, which uses the leakage algorithm $\mathcal{L}_\mathrm{lm}$ depicted in Fig. 14. Though the latter pseudocode looks quite complex, its intuition is straightforward. During the setup phase, its leakage is the sum of all the setup leakage from the $\mathbf{M}_i$ generated in $\mathsf{LMM.Enc}$ under $\mathcal{L}_\mathrm{i}$. When a query is made, the leakage algorithm recursively traverses all the descendants of the query (using $\mathsf{RecLeak}$) made (in the graph visualization of IA-MMs in Section 3) and leaks their depths and their query leakages (under the appropriate $\mathcal{L}_\mathrm{i}$). Additionally, it leaks the query leakage under $\mathcal{L}_0$ for the access to $\mathbf{M}_0$ and any additional accesses to $\mathbf{M}_1$ to retrieve non-RH values.

Theorem 4. *Let* LMM *be the IA-MME scheme for* IA *defined in Fig. 13 using MME primitives* $\mathsf{MME}_0, \ldots, \mathsf{MME}_D$ *(where* $D = \mathsf{IA.dp}$*) and function family* F*. Let* $\mathcal{L}_\mathrm{i}, \mathcal{S}_\mathrm{i}$ *be a leakage algorithm and simulator for* MME_i*. If* $\mathcal{L}_0, \mathcal{L}_2, \ldots, \mathcal{L}_\mathrm{D}$ *are content oblivious and* $\mathcal{L}_\mathrm{lm}$ *is as defined in Fig. 14, then given adversary* $\mathcal{A}$ *one can define* $\mathcal{A}_f, \mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_D, \mathcal{S}_\mathrm{lm}$ *such that:*

$$\begin{aligned}\mathbf{Adv}^{\mathrm{ss}}_{\mathsf{LMM},\mathcal{L}_\mathrm{lm},\mathcal{S}_\mathrm{lm}}(\mathcal{A}) \leq\ & \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A}_f) \\ &+ \mathbf{Adv}^{\mathrm{ss}}_{\mathsf{MME}_0,\mathcal{L}_0,\mathcal{S}_0}(\mathcal{A}_0) \\ &+ \mathbf{Adv}^{\mathrm{ss}}_{\mathsf{MME}_1,\mathcal{L}_1,\mathcal{S}_1}(\mathcal{A}_1) \\ &+ \cdots + \mathbf{Adv}^{\mathrm{ss}}_{\mathsf{MME}_D,\mathcal{L}_\mathrm{D},\mathcal{S}_\mathrm{D}}(\mathcal{A}_D).\end{aligned}$$

Since the proof of this result is standard and is an extension of Theorem 3, we provide only a sketch of this proof.

Proof Sketch. The proof proceeds through standard game transitions, much like those in the proof of Theorem 3. We begin in the "real-world," and make a game transition to a game where we replace MME_0 with a simulator and invoke the content obliviousness of $\mathcal{L}_0$, so that the tokens can be replaced at query time in future games.

Next, we use a series of game transitions to replace MME_2, then MME_3, etc, until MME_D are all replaced with simulators. At each of these steps, we must additionally invoke the content obliviousness of each of $\mathcal{L}_2, \ldots, \mathcal{L}_\mathrm{D}$. Finally, we replace MME_1 with a simulator in our last game transition, which is equivalent to the "simulated world" against $\mathcal{S}_\mathrm{lm}$ with $\mathcal{L}_\mathrm{lm}$, but do not require content oblivious leakage, because it is response-hiding.

In each of these game transitions, we build up to the fully recursive leakage in $\mathcal{L}_\mathrm{lm}$. We add on depth to this recursive leakage as we replace schemes with simulators in subsequent hybrid games.

C Inconsistent Simulators in Prior Work

We observe the use of "leakage rewriting" outlined in Section 4 in SPX, OPX, and LabGraph [23,26,14]. Specifically, these papers construct uniform, depth-2 IA-MMs usings the LMM scheme, but their proofs have a technical issue, which can be fixed without content oblivious assumption. Although this issue does occur multiple times in SPX and OPX, we will only outline a single occurrence for brevity. The other occurrences

Alg $\mathsf{LMM.Enc}(K, \mathbf{M})$

$D \leftarrow \mathsf{depth}(\mathbf{M})$
For $i = 0, \ldots, D$ do
  $K_i \leftarrow \mathsf{F.Ev}(K, i)$
For $\ell \in \{0,1\}^{\mathsf{len}}$ where $\mathbf{M}[\ell] \neq ()$ do
  $(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell]$
  For $i \in [n]$ do
    If $b_i = 0$ then $u_i \leftarrow (0, 0^{\mathsf{len}})$
    Else
      $d_i \leftarrow \mathsf{depth}(\mathbf{M}, v_i)$
      $tk_i \leftarrow\!\!\$\ \mathsf{MME}_{d_i}.\mathsf{Tok}(K_{d_i}, v_i)$
      $u_i \leftarrow (d_i, tk_i)$
  $\{j_k\}_{k\in[m]} \leftarrow \{i \in [n] : b_i = 0\}$
  $\mathbf{M}_1[\ell] \leftarrow (v_{j_1}, \ldots, v_{j_m})$
  $tk \leftarrow\!\!\$\ \mathsf{MME}_1.\mathsf{Tok}(K_1, \ell)$
  $d \leftarrow \mathsf{depth}(\mathbf{M}, \ell)$
  If $d = 1$ then $\mathbf{M}_0[\ell] \leftarrow (1, tk)$
  Else
    If $m \geq 1$ then $u_{j_1} \leftarrow (0, tk)$
    $\mathbf{M}_d[\ell] \leftarrow (u_1, \ldots, u_n)$
    $tk' \leftarrow\!\!\$\ \mathsf{MME}_d.\mathsf{Tok}(K_d, \ell)$
    $\mathbf{M}_0[\ell] \leftarrow ((d, tk'))$
For $d = 0, \ldots, D$ do
  $ED_d \leftarrow\!\!\$\ \mathsf{MME}_d.\mathsf{Enc}(K_d, \mathbf{M}_d)$
Return $(ED_0, \ldots, ED_D)$

Alg $\mathsf{LMM.Tok}(K, \ell)$

$tk \leftarrow\!\!\$\ \mathsf{MME}_0.\mathsf{Tok}\big(\mathsf{F.Ev}(K, 0), \ell\big)$
Return tk

Alg $\mathsf{LMM.Eval}(tk, \mathbf{ED})$

$(ED_0, ED_1, \ldots, ED_D) \leftarrow \mathbf{ED}$
$(d, tk') \leftarrow \mathsf{MME}_0.\mathsf{Eval}(tk, ED_0)$
$c' \leftarrow \mathsf{MME}_d.\mathsf{Eval}(tk', ED_d)$
If $d = 1$ then return $(1, c')$
$\big((d_1, tk_1), \ldots, (d_n, tk_n)\big) \leftarrow c'$
$\{j_k\}_{k\in[m]} \leftarrow \{i \in [n] : d_i = 0\}$
For $i \in [n]$ do
  If $d_i \neq 0$ do $u_i \leftarrow \mathsf{LMM.Eval}(tk_i, \mathbf{ED})$
  Else if $i = j_1$ then
    $u_i \leftarrow \perp$; $c \leftarrow \mathsf{MME}_1.\mathsf{Eval}(tk_i, ED_1)$
  Else $u_i \leftarrow \perp$
If $m \geq 1$ then
  return $\big((0, c), u_1, \ldots, u_n\big)$
Else return $(0, u_1, \ldots, u_n)$

Alg $\mathsf{LMM.Dec}(K, u)$

$K_1 \leftarrow \mathsf{F.Ev}(K, 1)$
If $u = (1, c')$ then return $\mathsf{MME}_1.\mathsf{Dec}(K_1, c')$
$(b, u_1, \ldots, u_n) \leftarrow u$
$\{j_k\}_{k\in[m]} \leftarrow \{i \in [n] : u_i = \perp\}$
For $i = 1, \ldots, n$ where $u_i \neq \perp$ do
  $w_i \leftarrow \mathsf{LMM.Dec}(K_1, u_i)$
If $b = (0, c)$ then
  $(v_1, \ldots, v_m) \leftarrow \mathsf{MME}_1.\mathsf{Dec}(K_1, c)$
  For $i \in [m]$ do $w_{j_i} \leftarrow v_i$
Return $(w_1, \ldots, w_n)$

Fig. 13. Algorithms for IA-MME scheme LMM (i.e. StE for IA). Note that when we assign the $\{j_k\}_{k\in[m]}$ we require that $j_1 < \cdots < j_m$. Since we move values to $\mathbf{M}_1$, by convention, we store the token pointing to $\mathbf{M}_i[\ell]$ in the first location a value would be (j_1).

$\underline{\textbf{Alg } \mathcal{L}_{\mathrm{lm}}(\mathbf{s}, \mathbf{M})}$
$K \leftarrow\!\!\$\ \mathsf{LMM.KS}$; $D \leftarrow \mathsf{depth}(\mathbf{M})$
Generate $\mathbf{M}_0, \ldots, \mathbf{M}_D$ as
  in $\mathsf{LMM.Enc}(K, \mathbf{M})$
For $i = 0, \ldots, D$ do
  $(lk_{\mathrm{i}}, St_i) \leftarrow\!\!\$\ \mathcal{L}_{\mathrm{i}}(\mathbf{s}, \mathbf{M}_i)$
$\mathbf{s} \leftarrow (St_0, \ldots, St_D)$
Return $\big((lk_0, \ldots, lk_{\mathrm{D}}), (\mathbf{s}, \mathbf{M})\big)$

$\underline{\textbf{Alg } \mathcal{L}_{\mathrm{lm}}(\mathsf{q}, \ell, (\mathbf{s}, \mathbf{M}))}$
$(St_0, \ldots, St_D) \leftarrow \mathbf{s}$
$(lk_0, St_0) \leftarrow\!\!\$\ \mathcal{L}_0(\mathsf{q}, \ell, St_0)$
$\mathbf{s} \leftarrow (St_0, \ldots, St_D)$
$(lk, \mathbf{s}) \leftarrow\!\!\$\ \mathsf{RecLeak}(\ell, \mathbf{s}, \mathbf{M})$
Return $\big((lk_0, lk), (\mathbf{s}, \mathbf{M})\big)$

$\underline{\textbf{Alg } \mathsf{RecLeak}(\ell, (St_0, \ldots, St_D), \mathbf{M})}$
$(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell]$
$d \leftarrow \mathsf{depth}(\mathbf{M}, \ell)$; $(lk', St_d) \leftarrow\!\!\$\ \mathcal{L}_{\mathrm{d}}(\mathsf{q}, \ell, St_d)$
If $d = 1$ then $lk \leftarrow lk'$
Else
  If $\exists i \in [n]$ where $b_i = 0$ then
    $(lk_c, St_1) \leftarrow\!\!\$\ \mathcal{L}_1(\mathsf{q}, \ell, St_1)$
  Else $lk_c \leftarrow \perp$
  For $i \in [n]$ do
    If $b_i \neq 0$ then
      $\mathbf{s} \leftarrow (St_0, \ldots, St_D)$
      $(lk_i, (St_0, \ldots, St_D)) \leftarrow \mathsf{RecLeak}(v_i, \mathbf{s}, \mathbf{M})$
    Else $lk_i \leftarrow \perp$
  $lk \leftarrow (lk_c, lk', lk_1, \ldots, lk_n)$
Return $\big((d, lk), (St_0, \ldots, St_D)\big)$

Fig. 14. Leakage algorithm for LMM used in the proof of Theorem 4. Here, each $\mathcal{L}_{\mathrm{i}}$ is a leakage algorithm for MME scheme MME_i. Note that for $i = 0, 2, \ldots, D$, MME_i is RR and $\mathcal{L}_{\mathrm{i}}$ is content oblivious. RecLeak is a helper algorithm used by $\mathcal{L}_{\mathrm{lm}}$ to handle recursive queries.

can be fixed using the same content oblivious assumption. We observe in parts of SPX and OPX the authors specify the construction should use a specific scheme, and the proofs which make this assumption are free from the issue, because those schemes already have content oblivious leakage. We also illustrate how the "chainability" assumption from [14] is insufficient to justify the use of the "leakage rewriting" trick.

C.1 Leakage Rewriting in SPX/OPX

The relevant definition for the issue is Definition 4.3 at the end of Section 4 in SPX. In the ideal game of this definition, the simulator only receives inputs of the form $(DS(q_i), \mathcal{L}_{\mathsf{Q}}(DS, q_i))$ to generate tokens.

We observe in the actual proof, the simulator $\mathcal{S}_{\mathsf{MM}}$ is not fed inputs of the same form as in the security definition. The occurrence we focus on is in section "Appendix F: Proof of Theorem 6.1" of SPX. The base simulator $\mathcal{S}_{\mathsf{MM}}$ is the simulator which exists based on the security Definition 4.3.

In describing the simulator, the authors write,

$$\mathsf{rtk}_{\mathbf{r}} \leftarrow \mathcal{S}_{\mathsf{MM}}((\mathrm{ct}_j)_{j \in [\#\mathbf{r}]}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}(\mathsf{MM}_R, \chi(\mathbf{r})))$$

And, then in the proof pass these $\mathsf{rtk}_{\mathbf{r}}$ into another simulator,

$$\mathbf{tk}_{i,j} \leftarrow \mathcal{S}_{\mathsf{MM}}((\mathsf{rtk}_{\mathbf{r}})_{\mathbf{r} \in \mathsf{DB}_{\mathsf{att}_{i,j}} = X_{i,j}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}(\mathsf{MM}_V, \chi(\mathsf{att}_{i,j})))$$

However, there is no guarantee that this input to $\mathcal{S}_{\mathsf{MM}}$ fits the input form required by Definition 4.3, because $(\mathsf{rtk}_{\mathbf{r}})_{\mathbf{r} \in \mathsf{DB}_{\mathsf{att}_{i,j}} = X_{i,j}}$ (tokens for MM_R) are not necessarily the same tokens contained in $\mathsf{MM}_V(\chi(\mathsf{att}_{i,j}))$. Unless the leakage from the generation of the simulated $\mathsf{rtk}_{\mathbf{r}}$ leaks the tokens in MM_R or a way to generate them, then it is unlikely over a random key choice the $\mathsf{rtk}_{\mathbf{r}}$ generated correspond to the the values stored in MM_V.

Note, this is not a necessary behavior of $\mathcal{S}_{\mathsf{MM}}$, but one that is not ruled out. In Section 4, we illustrate a possible change to the security definition and a sufficient condition on the leakage to avoid such a change. Either of which, will allow the proof to go through.

C.2 Leakage Rewriting in CK10

In LabGraph, the same trick is used in the proof of Theorem 6.2 [14]. At the beginning of the proof, the authors outline a simulator $\mathcal{S}$. In step 2b, $\mathcal{S}$ feeds in $\mathbf{v}_w$ generated from other simulators to generator a token τ_w. However, these simulators may not necessarily generate stored tokens τ^+ and τ^- with high probability.

It is worth noting these authors require their structured encryption algorithms to be "chainable," which places restrictions on both the setup and query leakage. However, the security definition (Definition 4.2)

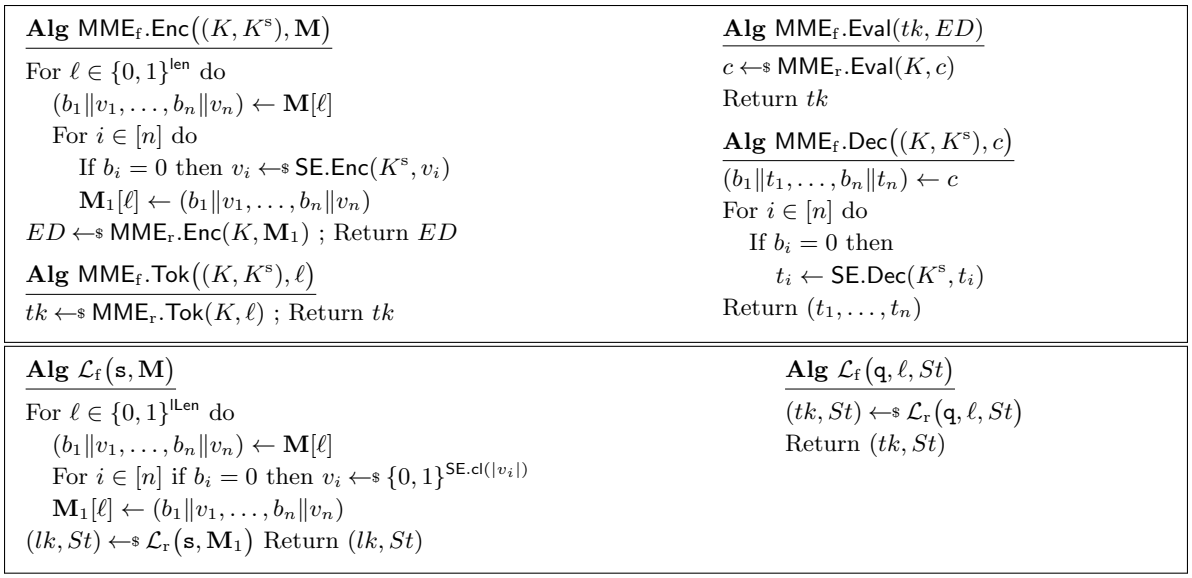

Fig. 15. Algorithms (top) and leakage algorithm (bottom) for RF MME scheme $\mathsf{MME}_\mathrm{f} = \mathbf{RfT}[\mathsf{MME}_\mathrm{r}, \mathsf{SE}]$ constructed using the **RfT** transform, RR MME scheme MME_r (with leakage algorithm $\mathcal{L}_\mathrm{r}$) and symmetric encryption scheme SE.

indicates the simulator for queries will receive input of the form $(\mathcal{L}_2(\delta, q), \mathbf{v}_I)$ where $I := \mathsf{Query}(\delta, q)$. But, the restrictions on the query leakage function in their Definition 6.1 (Chainability) do not rule out the existence of a simulator fitting the security definition but which behaves in an unspecified way on mismatched input.

D Achieving Response Flexibility

RfT TRANSFORM. In Section 5.1 we sketched how RF MME schemes can be constructed generically from RR MME using the **RfT** transform. Intuitively, we will first encrypt the values in $\mathbf{M}$ which are supposed to be hidden from the server using symmetric encryption scheme SE then encrypt this using RR MME scheme MME. More specifically, we define the generic transform MME_f which takes the primitives as input and returns RF MME scheme $\mathsf{MME}_\mathrm{f} = \mathbf{RfT}[\mathsf{MME}_\mathrm{r}, \mathsf{SE}]$. We define $\mathsf{MME}_\mathrm{f}.\mathsf{KS} = \mathsf{MME}_\mathrm{r}.\mathsf{KS} \times \mathsf{SE}.\mathsf{KS}$ and MME_f's algorithms are given in Fig. 15. Note that this satisfies both requirements of an RF MME because MME_r is RR and $\mathsf{MME}_\mathrm{f}.\mathsf{Dec1}((K, K^\mathrm{s}), \cdot) = \mathsf{SE}.\mathsf{Dec}(K^\mathrm{s}, \cdot)$. Note also that the values in $\mathbf{M}_1$ are slightly longer than in $\mathbf{M}$ because of SE's ciphertext expansion (i.e. if $\mathsf{vLen} = x$ in $\mathbf{M}$ then $\mathsf{vLen} = \mathsf{SE.cl}(x)$ in $\mathbf{M}_1$) so we expect $\mathsf{MME}_\mathrm{r}, \mathsf{MME}_\mathrm{f}$ to support the respective value lengths (and pad their values up to it if need be).

RfT PRESERVES SECURITY. The **RfT** transform preserves both (standard) semantic security and TV-security (as defined in Section 5.2) assuming state-of-the-art primitives. We state and prove the TV-security variant of this result and note that the analogous result for semantic security follows directly – by assuming that the adversary has no token values in $\mathbf{M}$.

In Section 5.1, we gave some intuition about deriving $\mathcal{L}_\mathrm{f}$ from $\mathcal{L}_\mathrm{r}$. For the **RfT** transform, the analogous leakage profile is given in Fig. 15. In this version of $\mathcal{L}_\mathrm{f}$, we construct $\mathbf{M}_1$ from $\mathbf{M}$ by transcribing the $b_i = 1$ values without modification, then replacing the $b_i = 0$ values with random strings of the appropriate length. Everything else proceeds in the intuitive way from here on, using $\mathcal{L}_\mathrm{r}$, with $\mathbf{M}_1$ instead of $\mathbf{M}$.

This leakage profile is stated a little different from the one sketched in Section 5.1 in order to simplify the proof. Namely, we sketched a leakage algorithm which uses $\mathcal{L}_\mathrm{r}$ algorithms everywhere except the query leakage, where the non-RR values would be omitted from the query response tuple that is leaked. However, they are essentially equivalent except that in the one below, we are implicitly leaking query patterns and do away with any need for content obliviousness.

Theorem 5. *Let* $\mathsf{MME}_\mathrm{f} = \mathbf{RfT}[\mathsf{MME}_\mathrm{r}, \mathsf{SE}]$ *be the RF MME scheme in Fig. 15. Let* $\mathcal{L}_\mathrm{r}$ *be the leakage algorithm for* MME_r *and* $\mathcal{L}_\mathrm{f}$ *be as defined in Fig. 15. Then, given an adversary* $\mathcal{A}$ *and simulator* $\mathcal{S}$*, one can*

Adversary $\mathcal{A}_1^{\text{Enc}}$

$K \leftarrow\$ \mathsf{MME_r.KS}$; $(\mathbf{M}, St_a) \leftarrow\$ \mathcal{A}(\mathtt{s})$
For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  $(b_1\|u_1, \ldots, b_n\|u_n) \leftarrow \mathbf{M}[\ell]$
  For $i \in [n]$ do
    If $b_i = 0$ then $v_i \leftarrow\$ \textsc{Enc}(v_i)$
    Else $v_i \leftarrow\$ \mathsf{MME.Tok}(K, v_i)$
  $\mathbf{M}_1[\ell] \leftarrow (b_1\|v_1, \ldots, b_n\|v_n)$
$ED \leftarrow\$ \mathsf{MME_r.Enc}(K, \mathbf{M}_1)$
$b' \leftarrow\$ \mathcal{A}^{\text{Tok}}(\mathtt{q}, ED, St_a)$; Return b'

Oracle $\text{Tok}(\ell)$

$tk \leftarrow\$ \mathsf{MME_r.Tok}(K, \ell)$; Return tk

Adversary $\mathcal{A}_2(\mathtt{s})$

$(\mathbf{M}, St_a) \leftarrow\$ \mathcal{A}(\mathtt{s})$
For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  $(b_1\|u_1, \ldots, b_n\|u_n) \leftarrow \mathbf{M}[\ell]$
  For $i \in [n]$ do
    If $b_i = 0$ then
      $v_i \leftarrow\$ \{0,1\}^{\mathsf{SE.cl}(|v_i|)}$
  $\mathbf{M}_1[\ell] \leftarrow (b_1\|v_1, \ldots, b_n\|v_n)$
Return $(\mathbf{M}_1, St_a)$

Adversary $\mathcal{A}_2^{\text{Tok}}(\mathtt{q}, ED, St_a)$

$b' \leftarrow\$ \mathcal{A}^{\text{Tok}}(\mathtt{q}, ED, St_a)$
Return b'

Games $\boxed{\text{G}_0}$, $\boxed{\text{G}_1}$

$K \leftarrow\$ \mathsf{MME_r.KS}$; $K^s \leftarrow\$ \mathsf{SE.KS}$
$(\mathbf{M}, St_a) \leftarrow\$ \mathcal{A}(\mathtt{s})$
For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  $(b_1\|v_1, \ldots, b_n\|v_n) \leftarrow \mathbf{M}[\ell]$
  For $i \in [n]$ do
  If $b_i = 1$ then $v_i \leftarrow\$ \mathsf{MME_r.Tok}(K, v_i)$
  $\boxed{\text{Else } v_i \leftarrow\$ \mathsf{SE.Enc}(K^s, v_i)}$
  $\boxed{\text{Else } v_i \leftarrow\$ \{0,1\}^{\mathsf{SE.cl}(|v_i|)}}$
  $\boxed{\mathbf{M}_1[\ell] \leftarrow (b_1\|v_1, \ldots, b_n\|v_n)}$
$ED \leftarrow\$ \mathsf{MME_r.Enc}(K, \mathbf{M}_1)$
$b' \leftarrow\$ \mathcal{A}^{\text{Tok}}(\mathtt{q}, ED, St_a)$
Return $b' = 1$

Oracle $\text{Tok}(\ell)$

$tk \leftarrow\$ \mathsf{MME_r.Tok}(K, \ell)$
Return tk

Game G_2

$(\mathbf{M}, St_a) \leftarrow\$ \mathcal{A}(\mathtt{s})$
For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  $(b_1\|v_1, \ldots, b_n\|v_n) \leftarrow \mathbf{M}[\ell]$
  For $i \in [n]$ do
  If $b_i = 1$ then
    $v_i \leftarrow\$ \mathsf{MME_r.Tok}(K, v_i)$
  Else $v_i \leftarrow\$ \{0,1\}^{\mathsf{SE.cl}(|v_i|)}$
  $\mathbf{M}_1[\ell] \leftarrow (b_1\|v_1, \ldots, b_n\|v_n)$
$(lk, St) \leftarrow\$ \mathcal{L}_r(\mathtt{s}, \mathbf{M})$
$(ED, St') \leftarrow\$ \mathcal{S}(\mathtt{s}, lk)$
$b' \leftarrow\$ \mathcal{A}^{\text{Tok}}(\mathtt{q}, ED, St_a)$
Return $b' = 1$

Oracle $\text{Tok}(\ell)$

$(lk, St) \leftarrow\$ \mathcal{L}_r(\mathtt{s}, \ell, St)$
$(tk, St') \leftarrow\$ \mathcal{S}(\mathtt{q}, lk, St')$
Return tk

Fig. 16. Adversaries and games used in the proof of Theorem 5.

construct $\mathcal{A}_1, \mathcal{A}_2$ *such that:*

$$\mathbf{Adv}^{\text{tv}}_{\mathsf{MME_f},\mathcal{L}_f,\mathcal{S}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{ind\$}}_{\mathsf{SE}}(\mathcal{A}_2) + \mathbf{Adv}^{\text{tv}}_{\mathsf{MME_r},\mathcal{L}_r,\mathcal{S}}(\mathcal{A}_1).$$

Proof. We define the adversaries $\mathcal{A}_1, \mathcal{A}_2$ in Fig. 16 along with three games $\text{G}_0, \text{G}_1, \text{G}_2$ where the adversary $\mathcal{A}$ plays different hybrid games. Let b, b_1, b_2 be the challenge bits in $\text{G}^{\text{tv}}_{\mathsf{MME_f},\mathcal{L}_f,\mathcal{S}}(\mathcal{A})$, $\text{G}^{\text{ind\$}}_{\mathsf{SE}}(\mathcal{A}_1)$ and $\text{G}^{\text{tv}}_{\mathsf{MME_r},\mathcal{L}_r,\mathcal{S}}(\mathcal{A}_2)$ respectively.

In G_0 (which includes the boxed code), the values in $\mathbf{M}_1$ are either encrypted with SE or tokenized with $\mathsf{MME_r}$. The encrypted data structure and tokens that $\mathcal{A}$ sees are also the outputs of $\mathsf{MME_r}$'s algorithms. This is equivalent to encryption and token generation in the "real world" when $\mathcal{A}$ plays the TV-security game for $\mathsf{MME_f}$. The adversary $\mathcal{A}_2$ (playing the IND\$ game) runs $\mathcal{A}$, simulating everything like in the real world of the TV-security game (with keys of its own choosing). uses its oracle to encrypt values in $\mathbf{M}_1$, but uses $\mathsf{MME_r}$'s algorithms everywhere else. Therefore, the "real world" of $\mathcal{A}_1$ playing the IND\$ game is also equivalent to G_0 and we have

$$\Pr[\text{G}_0] = \Pr[\text{G}^{\text{tv}}_{\mathsf{MME_f},\mathcal{L}_f,\mathcal{S}}(\mathcal{A}) | b = 1] = \Pr[\text{G}^{\text{ind\$}}_{\mathsf{SE}}(\mathcal{A}_1) | b_1 = 1].$$

G_1 is almost the same as G_0 except that in place of encryption with SE, a randomly selected bitstring of the same length will be used. From the above discussion, one can see this is equivalent to $\mathcal{A}_1$ playing the IND\$ game in the "ideal world". At the same time, $\mathcal{A}_2$ also runs $\mathcal{A}$ but replaces all values that should be encrypted in $\mathbf{M}$ with random strings before returning it during the setup phase. Therefore, G_1 is also

Game $\mathrm{G}^{\text{sim-ac-prf}}_{\mathsf{F},\mathsf{S},\mathsf{P}}(\mathcal{A})$	**Oracle** $\textsc{Ev}(u, x)$
For $u \in \{0,1\}^*$ do	$y_1 \leftarrow\!\!\$\ \mathsf{F.Ev}^{\mathsf{P}}(K_u, x)$
$\quad K_u \leftarrow\!\!\$\ \mathsf{F.KS}$	If $u \notin X$ then
$\sigma_{\mathsf{P}} \leftarrow\!\!\$\ \mathsf{P.Init}$	$\quad$ If $T_u[x] = \bot$ then $y_0 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.ol}}$
$\sigma \leftarrow\!\!\$\ \mathsf{S.Init}$	$\quad$ Else $y_0 \leftarrow T_u[x]$
$b \leftarrow\!\!\$\ \{0,1\}$	Else
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Ev},\textsc{Exp},\textsc{Prim}}$	$y_0 \leftarrow\!\!\$\ \mathsf{S.Ev}(x : \sigma)$
Return $b = b'$	$T_u[x] \leftarrow y_0$
Oracle $\textsc{Prim}(x)$	Return y_b
$y_1 \leftarrow\!\!\$\ \mathsf{P.Prim}(x : \sigma_{\mathsf{P}})$	**Oracle** $\textsc{Exp}(u)$
$y_0 \leftarrow\!\!\$\ \mathsf{S.Prim}(x : \sigma)$	$K_1 \leftarrow K_u$
Return y_b	$K_0 \leftarrow\!\!\$\ \mathsf{S.Exp}(u, T_u : \sigma)$
	$X.\mathsf{add}(u)$
	Return K_b

Game $\mathrm{G}^{\text{sim-ac-kp}}_{\mathsf{SE},\mathsf{S},\mathsf{P}}(\mathcal{A})$	**Oracle** $\textsc{Enc}(u, m)$
For $u \in \{0,1\}^*$ do	$c_1 \leftarrow\!\!\$\ \mathsf{SE.Enc}^{\mathsf{P}}(K_u, m)$
$\quad K_u \leftarrow\!\!\$\ \mathsf{SE.KS}$	If $u \notin X$ then
$\sigma_{\mathsf{P}} \leftarrow\!\!\$\ \mathsf{P.Init}$	$\quad c_0 \leftarrow\!\!\$\ \mathsf{S.Enc}_1(\lvert m\rvert : \sigma)$
$\sigma \leftarrow\!\!\$\ \mathsf{S.Init}$	Else
$b \leftarrow\!\!\$\ \{0,1\}$	$\quad c_0 \leftarrow\!\!\$\ \mathsf{S.Enc}_2(u, m : \sigma)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Ev},\textsc{Enc},\textsc{Prim}}$	$M_u.\mathsf{add}(m)\ ;\ C_u.\mathsf{add}(c_b)$
Return $b = b'$	Return c_b
Oracle $\textsc{Prim}(x)$	**Oracle** $\textsc{Exp}(u)$
$y_1 \leftarrow\!\!\$\ \mathsf{P.Prim}(x : \sigma_{\mathsf{P}})$	$K_1 \leftarrow K_u$
$y_0 \leftarrow\!\!\$\ \mathsf{S.Prim}(x : \sigma)$	$K_0 \leftarrow\!\!\$\ \mathsf{S.Exp}(u, M_u, C_u : \sigma)$
Return y_b	$X.\mathsf{add}(u)$
	Return K_b

Fig. 17. SIM-AC security definintions from [21] for PRF and CPA security. The bottom game is a combination of the original CPA game in the paper and a condition on the simulator structure to achieve key private security.

equivalent to $\mathcal{A}_2$ playing the TV-security game for $\mathsf{MME}_{\mathrm{r}}$ in the "real world". This gives us

$$\Pr[\mathrm{G}_1] = \Pr[\mathrm{G}^{\text{ind\$}}_{\mathsf{SE}}(\mathcal{A}_1)|b_1 = 0] = \Pr[\mathrm{G}^{\text{tv}}_{\mathsf{MME}_{\mathrm{r}},\mathcal{L}_{\mathrm{r}},\mathcal{S}}(\mathcal{A}_2)|b_2 = 1].$$

Finally, in G_2, values in $\mathbf{M}$ that should be encrypted are still replaced with random strings, but now this multimap is given to the leakage algorithm $\mathcal{L}_{\mathrm{r}}$ and simulator $\mathcal{S}$ to construct ED, tk. Immediately, we can see that this is equivalent to $\mathcal{A}_2$ playing the TV-security game for $\mathsf{MME}_{\mathrm{r}}$ in the "ideal world". At the same time, we defined $\mathcal{L}_{\mathrm{f}}$ to use the same replacement technique and so it is also equivalent to $\mathcal{A}$ playing the TV-security game for $\mathsf{MME}_{\mathrm{f}}$. This gives us

$$\Pr[\mathrm{G}_2] = \Pr[\mathrm{G}^{\text{tv}}_{\mathsf{MME}_{\mathrm{r}},\mathcal{L}_{\mathrm{r}},\mathcal{S}}(\mathcal{A}_2)|b_2 = 0] = \Pr[\mathrm{G}^{\text{tv}}_{\mathsf{MME}_{\mathrm{f}},\mathcal{L}_{\mathrm{f}},\mathcal{S}}(\mathcal{A})|b = 0].$$

Combining these three equations gives us the advantage bound in the theorem statement.

E TV-Secure RF MME scheme

In this section, we detail the RF MME scheme $\mathsf{MME}^{\mathrm{f}}_{\pi}$ and explain why it achieves sufficient security to be used in SMM. In particular, we reduce the TV-security of $\mathsf{MME}^{\mathrm{f}}_{\pi}$ to the security of its primitives under *adaptive compromise*. This latter notion of security was first defined by JT for both function families (which we use without modification) and symmetric encryption (which we modify slightly to suit the form of encryption done in our scheme).

AC Security Notions. In our proof, we reduce the TV-security of $\mathsf{MME}^{\mathrm{f}}_{\pi}$ to the SIM-AC-PRF security of F, the SIM-AC-KP security of SE and the PRF security of F. All three of the above security notions are

Alg $\mathsf{MME}^{\mathsf{f}}_{\pi}.\mathsf{Enc}^{\mathsf{P}_1,\mathsf{P}_2}\big((K^{\mathsf{f}},K_0),\mathbf{M}\big)$

For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  $K_{1,\ell} \leftarrow \mathsf{F.Ev}^{\mathsf{P}_1}\big(K^{\mathsf{f}},\ell\|1\big)$
  $K_{2,\ell} \leftarrow \mathsf{F.Ev}^{\mathsf{P}_1}\big(K^{\mathsf{f}},\ell\|2\big)$
  $(b_1\|v_1,\ldots,b_n\|v_n) \leftarrow \mathbf{M}[\ell]$
  For $i \in [n]$ do
    $b' \leftarrow \mathsf{SE.Enc}^{\mathsf{P}_2}(K_{1,\ell},b_i)$
    If $b_i = 1$ then
      $c \leftarrow \mathsf{SE.Enc}^{\mathsf{P}_2}(K_{1,\ell},v_i)$
    Else
      $c \leftarrow \mathsf{SE.Enc}^{\mathsf{P}_2}(K_0,v_i)$
    $\mathbf{T}[\mathsf{F.Ev}^{\mathsf{P}_1}(K_{2,\ell},i)] \leftarrow\!\!\$\ (b',c)$
Return $\mathbf{T}$

Alg $\mathsf{MME}^{\mathsf{f}}_{\pi}.\mathsf{Tok}^{\mathsf{P}_1,\mathsf{P}_2}((K^{\mathsf{f}},K_0),\ell)$

$K_{1,\ell} \leftarrow \mathsf{F.Ev}^{\mathsf{P}_1}\big(K^{\mathsf{f}},\ell\|1\big)$
$K_{2,\ell} \leftarrow \mathsf{F.Ev}^{\mathsf{P}_1}\big(K^{\mathsf{f}},\ell\|2\big)$
Return $(K_{1,\ell},K_{2,\ell})$

Alg $\mathsf{MME}^{\mathsf{f}}_{\pi}.\mathsf{Eval}^{\mathsf{P}_1,\mathsf{P}_2}((K_{1,\ell},K_{2,\ell}),\mathbf{T})$

$n \leftarrow 1$
While $\mathbf{T}[\mathsf{F.Ev}^{\mathsf{P}_1}(K_{2,\ell},n)] \neq \perp$ do
  $(b',c) \leftarrow \mathbf{T}[\mathsf{F.Ev}^{\mathsf{P}_1}(K_{2,\ell},n)]$
  $b \leftarrow \mathsf{SE.Dec}^{\mathsf{P}_2}(K_{1,\ell},b')$
  $v \leftarrow \mathsf{SE.Dec}^{\mathsf{P}_2}(K_{1,\ell},c)$
  If $b = 1$ then $v_n \leftarrow 1\|v$ else $v_n \leftarrow 0\|c$
  $n \leftarrow n+1$
$n \leftarrow n-1$
Return $(v_1,\ldots,v_n)$

Alg $\mathsf{MME}^{\mathsf{f}}_{\pi}.\mathsf{Dec}^{\mathsf{P}_1,\mathsf{P}_2}\big((K^{\mathsf{f}},K_0),\mathbf{v}\big)$

$(b_1\|v_1,\ldots,b_n\|v_n) \leftarrow \mathbf{v}$
For $i \in [n]$ do
  If $b_i = 0$ then
    $v_i \leftarrow \mathsf{SE.Dec}^{\mathsf{P}_2}\big(K_0,v_i\big)$
Return $(v_1,\ldots,v_n)$

Algs $\mathcal{L}^{\mathsf{f}}_{\pi}\big(\mathbf{s},\mathbf{M}\big)$

For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  $n \leftarrow n + \#(\mathbf{M}[\ell]))$
Return $\big(N,(\mathbf{M})\big)$

Algs $\mathcal{L}^{\mathsf{f}}_{\pi}\big(\mathsf{q},\ell,\mathbf{l}\big)$

$(\ell_1,\ldots,\ell_q,\mathbf{M}) \leftarrow \mathbf{l}$; $x \leftarrow \min_{\ell_i=\ell} i$
$(b_1\|v_1,\ldots,b_n\|v_n) \leftarrow \mathbf{M}[\ell]$
For $i \in [n]$ do
  If $b_i = 0$ then $lk_i \leftarrow \perp$
  Else $(lk_i,\mathbf{l}) \leftarrow \mathcal{L}^{\mathsf{f}}_{\pi}\big(\mathsf{q},v_i,\mathbf{l}\big)$
$lk \leftarrow (x,lk_1,\ldots,lk_q)$
Return $\big(lk,(\ell_1,\ldots,\ell_q,\ell,\mathbf{M})\big)$

Fig. 18. Algorithms for RF MME scheme $\mathsf{MME}^{\mathsf{f}}_{\pi}$, and its leakage profile.

given with respect to ideal primitives (similar to the random oracle model or ideal cipher model). An ideal primitive P has functions $\mathsf{P.Init}$, which outputs an initial state, and $\mathsf{P.Prim}(x : \sigma_{\mathsf{P}})$, which statefully evaluates inputs. For example, the fixed length n random oracle $\mathsf{P}^n_{\mathsf{rom}}$ outputs an empty table on init and lazily samples and records random elements from $\{0,1\}^n$ on each new input to $\mathsf{P}^n_{\mathsf{rom}}.\mathsf{Prim}(x)$.

Additionally, the two notions of AC-security are defined with respect to simulators. Intuitively, these simulators will be called upon to simulate calls to the ideal primitive, compromised keys, and evaluations/encryption using the cryptographic primitive. For more details on this syntax, the reader may refer to the work of [21]. These, along with the primitives, are indicated in the subscript of the games and advantages.

Our first security notion is SIM-AC-PRF. The relevant security game $\mathrm{G}^{\text{sim-ac-prf}}_{\mathsf{F,S,P}}$ is recalled in Fig. 17 with only minor modifications to allow for concrete notions of security (i.e. input and output lengths are assumed to be defined implicitly by the scheme, instead of in a security parameter). We define the advantage of an adversary $\mathcal{A}$ as $\mathbf{Adv}^{\text{sim-ac-prf}}_{\mathsf{F,S,P}}(\mathcal{A}) = 2\Pr[\mathrm{G}^{\text{sim-ac-prf}}_{\mathsf{F,S,P}}(\mathcal{A})] - 1$.

Also in Fig. 17 is the SIM-AC-KP notion of security captured in security game $\mathrm{G}^{\text{sim-ac-kp}}_{\mathsf{SE,S,P}}$, which is adapted from JT's notion of SIM-AC-CPA security. In particular, we bring together JT's notion of CPA security and our conditions on simulator structure to ensure key private (KP) security. As before, we define $\mathbf{Adv}^{\text{sim-ac-kp}}_{\mathsf{SE,S,P}}(\mathcal{A}) = 2\Pr[\mathrm{G}^{\text{sim-ac-kp}}_{\mathsf{SE,S,P}}(\mathcal{A})] - 1$.

Algorithms and leakage of $\mathsf{MME}^{\mathsf{f}}_{\pi}$. In Section 5, we suggested that the MME techniques of CJJ+ can be extended to achieve a RF MME scheme [10]. In Fig. 18, we give the algorithms and leakage profile of this scheme which we call $\mathsf{MME}^{\mathsf{f}}_{\pi}$.

The primitives used in $\mathsf{MME}^{\mathsf{f}}_{\pi}$ are symmetric encryption scheme SE and function family F. We require that $\mathsf{SE.KS} = \{0,1\}^{\mathsf{F.ol}} = \mathsf{F.KS}$ in $\mathsf{MME}^{\mathsf{f}}_{\pi}$. Note that $\mathsf{MME}^{\mathsf{f}}_{\pi}.\mathsf{KS} = \mathsf{SE.KS} \times \mathsf{F.KS}$. Finally, since we prove security in an idealized model, we give $\mathsf{MME}^{\mathsf{f}}_{\pi}$ access to two ideal primitives P_1 and P_2.

Game G_0

$\sigma_{\mathsf{P}} \leftarrow\!\!\$\ \mathsf{P}_1.\mathsf{Init}$
$\sigma'_{\mathsf{P}} \leftarrow\!\!\$\ \mathsf{P}_2.\mathsf{Init}$
$(\mathbf{M}, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{s})$
For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  If $\mathsf{Search}(\mathbf{M}[\ell], \{\ell\})$ then return false
$ED \leftarrow\!\!\$\ \mathsf{Setup}(\mathbf{M})$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Prim},\textsc{Tok}}(\mathsf{q}, ED, St_{\mathrm{a}})$
Return $b' = 1$

Oracle $\textsc{Prim}(x)$

$(d, x) \leftarrow x$
If $d = 1$ then
  $y \leftarrow\!\!\$\ \mathsf{P}_1.\mathsf{Prim}(x : \sigma_{\mathsf{P}})$
Else
  $y \leftarrow\!\!\$\ \mathsf{P}_2.\mathsf{Prim}(x : \sigma'_{\mathsf{P}})$
Return y

Oracle $\textsc{Tok}(\ell)$

Return $(K_{1,\ell}, K_{2,\ell})$

Alg $\mathsf{Setup}(\mathbf{M})$

$K_0 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.ol}}$
For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  $K_{1,\ell} \leftarrow \mathsf{F.Ev}^{\mathsf{P}_1}\big(K^{\mathrm{f}}, \ell\|1\big)$
  $K_{2,\ell} \leftarrow \mathsf{F.Ev}^{\mathsf{P}_1}\big(K^{\mathrm{f}}, \ell\|2\big)$
For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  $(b_1\|v_1, \ldots, b_n\|v_n) \leftarrow \mathbf{M}[\ell]$
  For $i \in [n]$ do
    $b' \leftarrow \mathsf{SE.Enc}^{\mathsf{P}_2}(K_{1,\ell}, b_i)$
    If $b_i = 1$ then
      $v_i \leftarrow\!\!\$\ (K_{1,v_i}, K_{2,v_i})$
      $c \leftarrow \mathsf{SE.Enc}^{\mathsf{P}_2}(K_{1,\ell}, v_i)$
    Else
      $c \leftarrow \mathsf{SE.Enc}^{\mathsf{P}_2}(K_0, v_i)$
    $\mathbf{T}[\mathsf{F.Ev}^{\mathsf{P}_1}(K_{2,\ell}, i)] \leftarrow\!\!\$\ (b', c)$
Return $\mathbf{T}$

Game G_1

$\sigma_{\mathsf{P}} \leftarrow\!\!\$\ \mathsf{P}_1.\mathsf{Init}$
$\sigma'_{\mathsf{P}} \leftarrow\!\!\$\ \mathsf{P}_2.\mathsf{Init}$
$(\mathbf{M}, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{s})$
For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  If $\mathsf{Search}(\mathbf{M}[\ell], \{\ell\})$ then return false
$ED \leftarrow\!\!\$\ \mathsf{Setup}(\mathbf{M})$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Prim},\textsc{Tok}}(\mathsf{q}, ED, St_{\mathrm{a}})$
Return $b' = 1$

Oracle $\textsc{Prim}(x)$

$(d, x) \leftarrow x$
If $d = 1$ then
  $y \leftarrow\!\!\$\ \mathsf{P}_1.\mathsf{Prim}(x : \sigma_{\mathsf{P}})$
Else
  $y \leftarrow\!\!\$\ \mathsf{P}_2.\mathsf{Prim}(x : \sigma'_{\mathsf{P}})$
Return y

Oracle $\textsc{Tok}(\ell)$

Return $(K_{1,\ell}, K_{2,\ell})$

Alg $\mathsf{Setup}(\mathbf{M})$

$K_0 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.ol}}$
For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  $K_{1,\ell} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.ol}}$
  $K_{2,\ell} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.ol}}$
For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  $(b_1\|v_1, \ldots, b_n\|v_n) \leftarrow \mathbf{M}[\ell]$
  For $i \in [n]$ do
    $b' \leftarrow \mathsf{SE.Enc}^{\mathsf{P}_2}(K_{1,\ell}, b_i)$
    If $b_i = 1$ then
      $v_i \leftarrow\!\!\$\ (K_{1,v_i}, K_{2,v_i})$
      $c \leftarrow \mathsf{SE.Enc}^{\mathsf{P}_2}(K_{1,\ell}, v_i)$
    Else
      $c \leftarrow \mathsf{SE.Enc}^{\mathsf{P}_2}(K_0, v_i)$
    $\mathbf{T}[\mathsf{F.Ev}^{\mathsf{P}_1}(K_{2,\ell}, i)] \leftarrow\!\!\$\ (b', c)$
Return $\mathbf{T}$

Alg $\mathsf{Search}((b_1\|v_1, \ldots, b_n\|v_n), S)$

For $i \in [n]$ do
  If $b_i = 1$ then
    If $v_i \in S$ then return true
    If $\mathsf{Search}(\mathbf{M}[v_i], S \cup v_i)$ then return true
Return false

Fig. 19. Games $\mathrm{G}_0, \mathrm{G}_1$ used in the proof of Theorem 1, and helper function Search used therein.

Game G_2

$\sigma_{\mathsf{P}} \leftarrow\!\!\$\ \mathsf{P}_1.\mathsf{Init}$
$\sigma' \leftarrow\!\!\$\ \mathsf{S}_{\mathsf{kp}}.\mathsf{Init}$
$(\mathbf{M}, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{s})$
For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  If $\mathsf{Search}(\mathbf{M}[\ell], \{\ell\})$ then return false
$ED \leftarrow\!\!\$\ \mathsf{Setup}(\mathbf{M})$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Prim},\textsc{Tok}}(\mathsf{q}, ED, St_{\mathrm{a}})$
Return $b' = 1$

Oracle $\textsc{Prim}(x)$

$(d, x) \leftarrow x$
If $d = 1$ then
  $y \leftarrow\!\!\$\ \mathsf{P}_1.\mathsf{Prim}(x : \sigma_{\mathsf{P}})$
Else
  $y \leftarrow\!\!\$\ \mathsf{S}_{\mathsf{kp}}.\mathsf{Prim}(x : \sigma')$
Return y

Oracle $\textsc{Tok}(\ell)$

If $\mathbf{M}[\ell] = \bot$ then
  $K_{1,\ell} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.ol}}$
Else then
  $(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell]$
  For $t \in T_\ell$
    $M_\ell.\mathsf{add}(\textsc{Tok}(t))$
  $K_{1,\ell} \leftarrow\!\!\$\ \mathsf{S}_{\mathsf{kp}}.\mathsf{Exp}(\ell, M_\ell, C_\ell : \sigma')$
Return $(K_{1,\ell}, K_{2,\ell})$

Alg $\mathsf{Setup}(\mathbf{M})$

For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  $K_{2,\ell} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.ol}}$
  $(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell]$
  For $i \in [n]$ do
    $b' \leftarrow \mathsf{S}_{\mathsf{kp}}.\mathsf{Enc}_1(|b_i| : \sigma')$
    $M_\ell.\mathsf{add}(b_i)$; $C_\ell.\mathsf{add}(b')$
  For $i \in [n]$ do
    $c \leftarrow \mathsf{S}_{\mathsf{kp}}.\mathsf{Enc}_1(2 \cdot \mathsf{F.ol} : \sigma')$
    If $b_i = 1$ then
      $T_\ell.\mathsf{add}(v_i)$; $C_\ell.\mathsf{add}(c)$
    $\mathbf{T}[\mathsf{F.Ev}^{\mathsf{P}_1}(K_{2,\ell}, i)] \leftarrow\!\!\$\ (b', c)$
Return $\mathbf{T}$

Game G_3

$\sigma \leftarrow\!\!\$\ \mathsf{S}_{\mathsf{prf}}.\mathsf{Init}$
$\sigma' \leftarrow\!\!\$\ \mathsf{S}_{\mathsf{kp}}.\mathsf{Init}$
$(\mathbf{M}, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{s})$
For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  If $\mathsf{Search}(\mathbf{M}[\ell], \{\ell\})$ then return false
$ED \leftarrow\!\!\$\ \mathsf{Setup}(\mathbf{M})$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Prim},\textsc{Tok}}(\mathsf{q}, ED, St_{\mathrm{a}})$
Return $b' = 1$

Oracle $\textsc{Prim}(x)$

$(d, x) \leftarrow x$
If $d = 1$ then
  $y \leftarrow\!\!\$\ \mathsf{S}_{\mathsf{prf}}.\mathsf{Prim}(x : \sigma)$
Else
  $y \leftarrow\!\!\$\ \mathsf{S}_{\mathsf{kp}}.\mathsf{Prim}(x : \sigma')$
Return y

Oracle $\textsc{Tok}(\ell)$

If $\mathbf{M}[\ell] = \bot$ then
  $K_{1,\ell} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.ol}}$
  $K_{2,\ell} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.ol}}$
Else then
  For $t \in T_\ell$
    $M_\ell.\mathsf{add}(\textsc{Tok}(t))$
  $K_{1,\ell} \leftarrow\!\!\$\ \mathsf{S}_{\mathsf{kp}}.\mathsf{Exp}(\ell, M_\ell, C_\ell : \sigma')$
  $K_{2,\ell} \leftarrow\!\!\$\ \mathsf{S}_{\mathsf{prf}}.\mathsf{Exp}(\ell, X_\ell : \sigma)$
Return $(K_{1,\ell}, K_{2,\ell})$

Alg $\mathsf{Setup}(\mathbf{M})$

For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  $(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell]$
  For $i \in [n]$ do
    $b' \leftarrow \mathsf{S}_{\mathsf{kp}}.\mathsf{Enc}_1(|b_i| : \sigma')$
    $M_\ell.\mathsf{add}(b_i)$; $C_\ell.\mathsf{add}(b')$
  For $i \in [n]$ do
    $c \leftarrow \mathsf{S}_{\mathsf{kp}}.\mathsf{Enc}_1(2 \cdot \mathsf{F.ol} : \sigma')$
    If $b_i = 1$ then
      $T_\ell.\mathsf{add}(v_i)$; $C_\ell.\mathsf{add}(c)$
    $x \leftarrow \{0,1\}^{\mathsf{F.ol}}$; $X_\ell.\mathsf{add}(x)$
    $\mathbf{T}[x] \leftarrow\!\!\$\ (b', c)$
Return $\mathbf{T}$

Fig. 20. Games $\mathrm{G}_2, \mathrm{G}_3$ used in the proof of Theorem 1. The Search helper function is given in Fig. 19.

Adversary $\mathcal{A}_1^{\text{Ev},\text{Prim}}$

$\sigma'_{\mathsf{P}} \leftarrow\!\!\$\ \mathsf{P}_2.\mathsf{Init}$
$(\mathbf{M}, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{s})$
For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  If $\mathsf{Search}(\mathbf{M}[\ell], \{\ell\})$ then return false
$ED \leftarrow\!\!\$\ \mathsf{Setup}(\mathbf{M})$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\text{PrimSim},\text{TokSim}}(\mathsf{q}, ED, St_{\mathrm{a}})$
Return $b' = 1$

Oracle PrimSim(x)

$(d, x) \leftarrow x$
If $d = 1$ then
  $y \leftarrow\!\!\$\ \text{Prim}(x)$
Else
  $y \leftarrow\!\!\$\ \mathsf{P}_2.\mathsf{Prim}(x : \sigma'_{\mathsf{P}})$
Return y

Oracle TokSim(ℓ)

Return $(K_{1,\ell}, K_{2,\ell})$

Alg $\mathsf{Setup}(\mathbf{M})$

$K_0 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.ol}}$
For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  $K_{1,\ell} \leftarrow \text{Ev}(\ell \| 1)$
  $K_{2,\ell} \leftarrow \text{Ev}(\ell \| 2)$
For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  $(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell]$
  For $i \in [n]$ do
    $b' \leftarrow \mathsf{SE.Enc}^{\mathsf{P}_2}(K_{1,\ell}, b_i)$
    If $b_i = 1$ then
      $v_i \leftarrow\!\!\$\ (K_{1,v_i}, K_{2,v_i})$
      $c \leftarrow \mathsf{SE.Enc}^{\mathsf{P}_2}(K_{1,\ell}, v_i)$
    Else
      $c \leftarrow \mathsf{SE.Enc}^{\mathsf{P}_2}(K_0, v_i)$
    $\mathbf{T}[\mathsf{F.Ev}^{\mathsf{P}_1}(K_{2,\ell}, i)] \leftarrow\!\!\$\ (b', c)$
Return $\mathbf{T}$

Adversary $\mathcal{A}_2^{\text{Enc},\text{Exp},\text{Prim}}$

$\sigma_{\mathsf{P}} \leftarrow\!\!\$\ \mathsf{P}_1.\mathsf{Init}$
$(\mathbf{M}, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{s})$
For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  If $\mathsf{Search}(\mathbf{M}[\ell], \{\ell\})$ then return false
$ED \leftarrow\!\!\$\ \mathsf{Setup}(\mathbf{M})$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\text{PrimSim},\text{TokSim}}(\mathsf{q}, ED, St_{\mathrm{a}})$
Return $b' = 1$

Oracle TokSim(ℓ)

If $\mathbf{M}[\ell] = \bot$ then
  $K_{1,\ell} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.ol}}$
Else then
  $(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell]$
  For $i \in [n]$ do
    If $b_i = 1$ then
      If $K_{1,v_i} = \bot$ then
        $v_i \leftarrow\!\!\$\ \text{TokSim}(v_i)$
      Else $v_i \leftarrow (K_{1,v_i}, K_{2,v_i})$
      $c \leftarrow \text{Enc}(\ell, v_i)$
  $K_{1,\ell} \leftarrow\!\!\$\ \text{Exp}(\ell)$
Return $(K_{1,\ell}, K_{2,\ell})$

Oracle PrimSim(x)

$(d, x) \leftarrow x$
If $d = 1$ then
  $y \leftarrow\!\!\$\ \mathsf{P}_1.\mathsf{Prim}(x : \sigma_{\mathsf{P}})$
Else
  $y \leftarrow\!\!\$\ \text{Prim}(x)$
Return y

Alg $\mathsf{Setup}(\mathbf{M})$

For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  $K_{2,\ell} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.ol}}$
For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  $(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell]$
  For $i \in [n]$ do
    $b' \leftarrow \text{Enc}(\ell, b_i)$
  For $i \in [n]$ do
    If $b_i = 1$ then
      $v_i \leftarrow\!\!\$\ \text{TokSim}(v_i)$
      $c \leftarrow \text{Enc}(\ell, v_i)$
    Else
      $c \leftarrow \text{Enc}(0, v_i)$
    $\mathbf{T}[\mathsf{F.Ev}^{\mathsf{P}_1}(K_{2,\ell}, i)] \leftarrow\!\!\$\ (b', c)$
Return $\mathbf{T}$

Fig. 21. Adversaries used for the first (top) and second (bottom) game hops in the proof of Theorem 1.

Adversary $\mathcal{A}_3^{\text{Ev},\text{Exp},\text{Prim}}$

$\sigma' \leftarrow\!\!\$\ \mathsf{S_{kp}}.\mathsf{Init}$
$(\mathbf{M}, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{s})$
For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  If $\mathsf{Search}(\mathbf{M}[\ell], \{\ell\})$ then return false
$ED \leftarrow\!\!\$\ \mathsf{Setup}(\mathbf{M})$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\text{PrimSim},\text{TokSim}}(\mathsf{q}, ED, St_{\mathrm{a}})$
Return $b' = 1$

Oracle TokSim(ℓ)

If $\mathbf{M}[\ell] = \perp$ then
  $K_{1,\ell} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.ol}}$
  $K_{2,\ell} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.ol}}$
Else then
  For $t \in T_\ell$
    $M_\ell.\mathsf{add}(\text{TokSim}(t))$
  $K_{1,\ell} \leftarrow\!\!\$\ \mathsf{S_{kp}}.\mathsf{Exp}(\ell, M_\ell, C_\ell : \sigma')$
  $K_{2,\ell} \leftarrow\!\!\$\ \text{Exp}(\ell)$
Return $(K_{1,\ell}, K_{2,\ell})$

Oracle PrimSim(x)

$(d, x) \leftarrow x$
If $d = 1$ then
  $y \leftarrow\!\!\$\ \text{Prim}(x)$
Else
  $y \leftarrow\!\!\$\ \mathsf{S_{kp}}.\mathsf{Prim}(x : \sigma')$
Return y

Alg Setup($\mathbf{M}$)

For $\ell \in \{0,1\}^{\mathsf{lLen}}$ do
  $(b_1 \| v_1, \ldots, b_n \| v_n) \leftarrow \mathbf{M}[\ell]$
  For $i \in [n]$ do
    $b' \leftarrow \mathsf{S_{kp}}.\mathsf{Enc}_1(|b_i| : \sigma')$
    $M_\ell.\mathsf{add}(b)$; $C_\ell.\mathsf{add}(b')$
  For $i \in [n]$ do
    $c \leftarrow \mathsf{S_{kp}}.\mathsf{Enc}_1(2 \cdot \mathsf{F.ol} : \sigma')$
    If $b_i = 1$ then
      $T_\ell.\mathsf{add}(v_i)$; $C_\ell.\mathsf{add}(c)$
    $x \leftarrow \text{Ev}(\ell, i)$
    $\mathbf{T}[x] \leftarrow\!\!\$\ (b', c)$
Return $\mathbf{T}$

Fig. 22. Third adversary for the final game hop in the proof of Theorem 1.

Security reduction. We are now ready to state and prove the security reduction of $\mathsf{MME}^{\mathrm{f}}_{\pi}$. The theorem stated below is analogous to the one in [21]. Note that, as mentioned before, we denote the ideal primitives and simulators used in each definition in subscripts of each advantage terms. Additionally, we adopt the same notation used by JT to manage a list L as a queue by using $L.\mathsf{add}, L.\mathsf{dq}$ to queue and dequeue list elements, respectively.

Theorem 1. *Let $\mathsf{MME}^{\mathrm{f}}_{\pi}$ and $\mathcal{L}_{\mathrm{f}}$ be the scheme and leakage respectively described in Figure 18 using PRF F, symmetric encryption scheme SE, and ideal primitives P_1 and P_2. Then, given adversary $\mathcal{A}$ and simulators $\mathsf{S_{prf}}, \mathsf{S_{kp}}$ one can define $\mathcal{S}_{\mathrm{f}}, \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ such that:*

$$\begin{aligned}\mathbf{Adv}^{\mathrm{tv}}_{\mathsf{MME}^{\mathrm{f}}_{\pi},\mathcal{L}_{\mathrm{f}},\mathcal{S}_{\mathrm{f}},\mathsf{P}_1,\mathsf{P}_2}(\mathcal{A}) \le\ & \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F},\mathsf{P}_1}(\mathcal{A}_1) + \mathbf{Adv}^{\text{sim-ac-kp}}_{\mathsf{SE},\mathsf{S_{kp}},\mathsf{P}_2}(\mathcal{A}_2) \\ & + \mathbf{Adv}^{\text{sim-ac-prf}}_{\mathsf{F},\mathsf{S_{prf}},\mathsf{P}_1}(\mathcal{A}_3).\end{aligned}$$

Proof. The proof closely follows the proof in Appendix D of [21]. In particular, we use the same 3 primary game hops, however we reverse the order of the last two for simplicity. To navigate these hops, along with the respective adversaries and games, we focus on the differences from the proof given by JT.

Claim. Let G_0 and G_1 be defined as in Figure (19), and $\mathcal{A}_1$ be defined as in Figure (21). Then,

$$|\Pr[\mathrm{G}_1(\mathcal{A})] - \Pr[\mathrm{G}_0(\mathcal{A})]| \le \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F},\mathsf{P}_1}(\mathcal{A}_1).$$

The first game-hop (Claim 1 in [21]) generates K_1 and K_2 uniformly at random instead of as PRF outputs, this step goes through without any issue and gives the $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F},\mathsf{P}_1}(\mathcal{A}_1)$ term in the statement.

We switch the order of the next 2 gamehops relative to [21] for simplicity. In the second hop, we construct an adversary for an encryption game defined in [21]. This allows us to run a simulator in the setup phase that outputs ciphertexts. Later (when $\mathcal{A}$ calls Tok), we can give the simulator messages we want the ciphertexts to decrypt to and it will return a key doing just that.

Claim. Let G_2 be defined as in Figure (20), G_1 as in Figure (19), and $\mathcal{A}_2$ as in Figure (21). Then,

$$|\Pr[\mathrm{G}_2(\mathcal{A})] - \Pr[\mathrm{G}_1(\mathcal{A})]| \le \mathbf{Adv}^{\text{sim-ac-kp}}_{\mathsf{SE},\mathsf{S_{kp}},\mathsf{P}_2}(\mathcal{A}_2).$$

Alg $\mathcal{S}_{\mathrm{f}}(\mathsf{s}, lk)$
$(N) \leftarrow lk$
$\sigma \leftarrow\!\!\$\ \mathsf{S}_{\mathsf{prf}}.\mathsf{Init}$
$\sigma' \leftarrow\!\!\$\ \mathsf{S}_{\mathsf{kp}}.\mathsf{Init}$
For $i \in [N]$ do
  $b' \leftarrow \mathsf{S}_{\mathsf{kp}}.\mathsf{Enc}_1(1 : \sigma')$
  $c \leftarrow \mathsf{S}_{\mathsf{kp}}.\mathsf{Enc}_1(2 \cdot \mathsf{F.ol} : \sigma')$
  $x \leftarrow \{0,1\}^{\mathsf{F.ol}}$
  $L.\mathsf{add}((x, b', c))$
  $\mathbf{T}[x] \leftarrow\!\!\$\ (b', c)$
$U \leftarrow [N]$
$St \leftarrow (\sigma, \sigma', U, L, T)$
Return $(\mathbf{T}, St)$

Alg $\mathcal{S}_{\mathrm{f}}(\mathsf{q}, lk, St)$
$(\sigma, \sigma', U, L, T) \leftarrow St$; $(tk_1, \ldots, tk_q) \leftarrow T$
$(x, lk_1, \ldots, lk_q) \leftarrow lk$
If $x \geq 1$ then return tk_x
For $i \in [n]$ do
  If $lk_i = \perp$ then $b_i \leftarrow 0$
  Else $b_i \leftarrow 1$; $(v_i, t) \leftarrow\!\!\$\ \mathcal{S}_{\mathrm{f}}(\mathsf{q}, lk_i, St)$
  $(x, b', c_i) \leftarrow L.\mathsf{dq}()$
  $M.\mathsf{add}(b_i)$; $C.\mathsf{add}(b')$; $X.\mathsf{add}(x)$
For $i \in [n]$ do
  If $b_i = 1$ then $M.\mathsf{add}(v_i)$; $C.\mathsf{add}(c)$
$\ell \leftarrow\!\!\$\ U$; $U \leftarrow U \setminus \{\ell\}$
$K_1 \leftarrow\!\!\$\ \mathsf{S}_{\mathsf{kp}}.\mathsf{Exp}(\ell, M, C : \sigma')$
$K_2 \leftarrow\!\!\$\ \mathsf{S}_{\mathsf{prf}}.\mathsf{Exp}(\ell, X : \sigma)$
$tk \leftarrow (K_1, K_2)$; $T \leftarrow (tk_1, \ldots, tk_q, tk)$
$St \leftarrow (\sigma, \sigma', U, L, T)$
Return (tk, St)

Fig. 23. Simulator for the proof of Theorem 1 which uses the leakage function from Figure (18).

At this point, we run into our first complication. The key given to the adversary when Tok is called, must decrypt the revealed entries to other valid tokens. In both games each K_2 is still generated at random, so decrypting to those is just a matter of writing them down at setup and resusing them in the Tok.

However, the K_1 for each token we reveal must also be a valid token. In G_1, we generate K_1 at random just like K_2, which can be used later when Tok is called. However, in G_2, we delay the computing K_1 until Tok is called. However, in order for the simulator S_{kp} to generate K_1, it must recursively determine the tokens for revealed labels. We avoid any circular dependence by requiring an adversary to supply a multimap without cycles in the TV-security game. This jump, using S_{kp} to give usable tokens gives us the $\mathbf{Adv}^{\text{sim-ac-kp}}_{\mathsf{SE},\mathsf{S}_{\mathsf{kp}},\mathsf{P}_2}(\mathcal{A}_2)$ in the statement.

Claim. Let G_3 and G_2 be defined as in Figure (20), and $\mathcal{A}_3$ be defined as in Figure (22). Then,

$$|\Pr[\mathrm{G}_3(\mathcal{A})] - \Pr[\mathrm{G}_2(\mathcal{A})]| \leq \mathbf{Adv}^{\text{sim-ac-prf}}_{\mathsf{F},\mathsf{S}_{\mathsf{prf}},\mathsf{P}_1}(\mathcal{A}_3),$$

For the final jump, we replace the remaining PRF outputs with random outputs to determine where in the data structure ciphertexts are stored. In G_3, the key for these outputs is determined later when Tok is queried. In a similar way K_1 was in the previous hop, the simulator $\mathsf{S}_{\mathsf{prf}}$ provides a key that satisfies some of the previous outputs. This final hop gives us the $\mathbf{Adv}^{\text{sim-ac-prf}}_{\mathsf{F},\mathsf{S}_{\mathsf{prf}},\mathsf{P}_1}(\mathcal{A}_3)$ term in the theorem.

Claim. Let G_3 be defined as in Figure (20), G_0 as in Figure (19), $\mathcal{L}_{\mathrm{f}}$ as in Figure (18), and $\mathcal{S}_{\mathrm{f}}$ as defined in Figure (23). Then,

$$|\Pr[\mathrm{G}_3(\mathcal{A})] - \Pr[\mathrm{G}_0(\mathcal{A})]| = \mathbf{Adv}^{\text{tv}}_{\mathsf{MME}^{\mathrm{f}}_{\pi},\mathcal{L}_{\mathrm{f}},\mathcal{S}_{\mathrm{f}},\mathsf{P}_1,\mathsf{P}_2}(\mathcal{A}),$$

At the end, we end up with a simulator, which outputs simulated ciphertexts in random locations at setup phase. Then, at query time, determines the tokens to match the locations and revealed values with recursive calls. This can be implemented with the recursive leakage $\mathcal{L}^{\mathrm{f}}_{\pi}$ and simulates the ideal game. We observe that G_0 is exactly the real game and G_3 is the ideal game, which with the triangle inequality completes the proof.

F Searchable Encryption using MME and IA-MME

We provide the pseudocode for several Searchable Encryption schemes.

Searchable Encryption Data Type. We begin by formalizing a data type for searchable encryption SEdt. This captures collections of documents which the client seeks to retrieve via keywords that have been

Alg $\mathsf{SE1.Enc}(K, DS)$
For $w \in \bigcup l_{(D,W)\in DS} W$ do
  $\mathbf{D} \leftarrow \big(D : (D,W) \in DS\ ,\ w \in W\big)$
  $\mathbf{M}[w] \leftarrow \mathsf{Part}(\mathbf{D})$
Return $\mathsf{MME.Enc}(K, \mathbf{M})$

Alg $\mathsf{SE1.Tok}(K, w)$
Return $\mathsf{MME.Tok}(K, w)$

Alg $\mathsf{SE1.Eval}(tk, ED)$
Return $\mathsf{MME.Eval}(tk, ED)$

Alg $\mathsf{SE1.Dec}(K, c)$
$\mathbf{D} \leftarrow \mathsf{Unpart}(\mathsf{MME.Dec}(K, c))$
Return $\{D : D \in \mathbf{D}\}$

Alg $\mathsf{SE2.Enc}(K, DS)$
$(D_1, W_1), \ldots, (D_n, W_n) \leftarrow DS$
For $i \in [n]$ do $\mathbf{M}[0\|i] \leftarrow \mathsf{Part}(D_i)$
For $w \in \bigcup_{(D,W)\in DS} W$ do
  $\mathbf{M}[1\|w] \leftarrow \big(0\|i : i \in [n]\ ,\ w \in W_i\big)$
Return $\mathsf{IAMME.Enc}(K, \mathbf{M})$

Alg $\mathsf{SE2.Tok}(K, w)$
Return $\mathsf{IAMME.Tok}(K, 1\|w)$

Alg $\mathsf{SE2.Eval}(tk, ED)$
Return $\mathsf{IAMME.Eval}(tk, ED)$

Alg $\mathsf{SE2.Dec}(K, c)$
$\mathbf{D} \leftarrow \mathsf{Unpart}(\mathsf{IAMME.Dec}(K, c))$
Return $\{D : D \in \mathbf{D}\}$

Fig. 24. Two StE schemes for SEdt (searchable encryption). $\mathsf{SE1}$ uses the strawman solution of in-lined payloads and an MME scheme MME, and $\mathsf{SE2}$ uses depth-two indirect addressing scheme IAMME.

pre-assigned to the documents. For simplicity, we will assume that all keywords are of a constant length len (in practice, this can be achieved with hashing). So we define

$$\mathsf{SEdt.Dom} = \{\{D_i, W_i\}_{i=1}^n : \forall i\ ,\ D_i \in \{0,1\}^*, W_i \subseteq \{0,1\}^{\mathsf{len}}\}\ ,$$
$$\mathsf{SEdt.QS} = \{0,1\}^{\mathsf{len}}\ ,$$
$$\mathsf{SEdt.Eval}(DS, w) = \{D : (D,W) \in DS\ ,\ w \in W\}.$$

StE for SEdt. We now describe the two StE schemes for SEdt. Note that in our pseudocode, we assume the existence of a partitioning algorithm Part which breaks a string of arbitrary length into a tuple of strings of length len, then affixes a leading 0 to them (so they are ready for insertion into the IA-MME at depth-1). We require that this partitioning be invertible via an algorithm Unpart, even when multiple partitioned strings have been concatenated. In other words, for any strings $s_1, \ldots, s_n$ we require that $\mathsf{Unpart}(\mathsf{Part}(s_1), \ldots, \mathsf{Part}(s_n)) = (s_1, \ldots, s_n)$. Additionally, in pseudocode, we assume that "For" loops iterate over sets in a random order.

Our first scheme is $\mathsf{SE1}$ which uses an MME primitive MME. This is based on the strawman "inlined-payloads" technique mentioned in Section 3. Let $\mathsf{SE1.KS} = \mathsf{MME.KS}$, and the pseudocode is given in Fig. 24. Note that since this is using an MME scheme, we do not need the output of Part to have leading 0s.

Our second scheme is $\mathsf{SE2}$ which uses a uniform depth-2 IA-MM for indexing. Let $\mathsf{SE1.KS} = \mathsf{IAMME.KS}$, and the pseudocode is given in Fig. 24. As shown in our simulations in Section 6, this saves storage since each document payload is only stored once in the dataset.

G SQL StE using IA-MME

We now demonstrate various ways that IA-MME can be used to build StE for SQL Databases. To simplify our discussion, we will focus on two types of non-recursive SQL queries: relation retrievals and single-attribute joins. However, we believe that the below StE schemes can extend their query support (e.g. select queries) using standard techniques in the literature [26,12].

SQL Data Type. The data type capturing our desired functionality will be SQLdt.

First, we capture each relation in a SQL database as a tuple of strings with each string representing all the values in a row. We assume that relations in the database can be uniquely identified via a string identifier of length len. The set of possible SQL relations are therefore given by

$$\mathsf{SQLdt.Rltns} = \{(\mathrm{id}, \mathbf{r}) : id \in \{0,1\}^{\mathsf{len}}\ ,\ \mathbf{r} = (r_1, \ldots, r_n)\ ,\ r_1, \ldots, r_n \in \{0,1\}^*\}.$$

With this, we can define the data structures in SQLdt as

$$\mathsf{SQLdt.Dom} = \{DS \ : \ DS = \{(id_i, \mathbf{r}_i)\}_{i=1}^{n}\} \subset \mathsf{SQLdt.Rltns} \ , \ id_1 \neq \ldots \neq id_n\}.$$

We simplify the notation for retrieving the rows in a table by its unique identifier, with $DS[id] = \mathbf{r}$ if and only if $(id, \mathbf{r}) \in DS$. If no such relation exists in DS, then $DS[id] = \perp$.

Joins take the rows of two tables and return some subsection of their cross product based on a predicate For example, in an equijoin, the predicate is the equality of the value in a particular column of the left (i.e. first input) table with another in a particular column of the right (i.e. second input) table. Therefore, we simplify joins by defining join predicates to be the set of functions:

$$\mathsf{JPs} = \{\mathsf{jpred} \ : \ \{0,1\}^* \times \{0,1\}^* \to \{0,1\}\}.$$

We also define the function Join to compute a join. It takes two sets of rows and a join predicate as input and returns a set of tuples of rows:

$$\mathsf{Join}(\mathbf{r}_1, \mathbf{r}_2, \mathsf{jpred}) = \{(r_1, r_2) \ : \ r_1 \in \mathbf{r}_1 \ , \ r_2 \in \mathbf{r}_2 \ , \ \mathsf{jpred}(r_1, r_2) = 1\}.$$

We want to support all queries of the form "**select** * **from** [table]" and a user-defined subset of the queries of the form "**select** * **from** [table1] **join** [table2] **on** [predicate]". To capture this, the scheme defines $\mathsf{SQLdt.JnQs}$ such that:

$$\mathsf{SQLdt.JnQs} \subseteq \{(\mathtt{j}, \mathsf{jpred}, id_1, id_2) \ : \ \mathsf{jpred} \in \mathsf{JPs} \ , \ id_1, id_2 \in \{0,1\}^{\mathsf{len}}\}$$

$$\mathsf{SQLdt.QS} = \{(\mathbf{r}, id) \ : \ id \in \{0,1\}^{\mathsf{len}}\} \ \cup \mathsf{SQLdt.JnQs}.$$

And query evaluation works as follows, for any $DS, \mathsf{jpred}, id, id_1, id_2$:

$$\mathsf{SQLdt.Eval}(DS, (\mathbf{r}, id)) = DS[id]$$

$$\mathsf{SQLdt.Eval}(DS, (\mathtt{j}, \mathsf{jpred}, id_1, id_2)) = \begin{cases} \perp & \text{, if } DS[id_1] = \perp \text{ or } DS[id_2] = \perp \\ \perp & \text{, if } \mathsf{Join}(DS[id_1], DS[id_2], \mathsf{jpred}) = \emptyset \\ \mathsf{Join}(DS[id_1], DS[id_2], \mathsf{jpred}), & \text{otherwise} \end{cases}$$

Indirect Addressing for SQLdt. We will describe three StE schemes for SQLdt that use an arbitrary IA-MME scheme (LMM or SMM). These adapt and extend the techniques from prior work in SQL StE and demonstrates the versatility of IA-MME, and its power to simplify complicated StE schemes.

We also use Part as we did for SE, to partition a string into blocks of length len and add leading zeroes. Similarly, Unpart restores any tuple of the form $\mathsf{Part}(r_1)\| \ldots \|\mathsf{Part}(r_n)$ to $(r_1, \ldots, r_n)$. To simplify notation, we will assume the adversary chooses the set of joins $\mathsf{SQLdt.JnQs}$ and makes queries such that the output is not $\perp$. Additionally, for clarity, we allow arbitrary length labels in IAMME, with the understanding that they can be hashed to length len to match the desired syntax of an IA-MME. In all three cases, the security of the StE scheme reduces to that of IAMME immediately, and we omit the proofs for brevity.

The first scheme is $\mathsf{FP2}$ that uses depth-two indirect addressing to perform fully precomputed join indexing. When SQL StE schemes SPX, OPX and FpSj are restricted to the query support in SQLdt, their schemes are basically equivalent to $\mathsf{FP2}$ with $\mathsf{LMM}_{\mathrm{u}}$ as the underlying primitive [23,26,12]. Let $\mathsf{FP2.KS} = \mathsf{IAMME.KS}$. The algorithms for $\mathsf{FP2}$ are given in Fig. 25.

The next scheme is $\mathsf{PP2}$ that modifies the approach above with partially precomputed join indexing. When SQL StE scheme FpSj is restricted to the query support in SQLdt, the schemes are basically equivalent to $\mathsf{PP2}$. The scheme's key set is given by $\mathsf{PP2.KS} = \mathsf{IAMME.KS} \times \mathsf{SE.KS}$ and its pseudocode is in Fig. 25. Compared to $\mathsf{FP2}$ has strictly less leakage (i.e. it is more secure) and reduces bandwidth and storage on practical datasets [12].

The scheme $\mathsf{PP3}$ is a slight modification of $\mathsf{PP2}$ which makes use of the observation that when partially precomputed joins are indexed, the scheme might store the same combinations of rows in the multimap as it would in a relation retrieval query. This happens whenever an indexed join returns all rows from a relation somewhere in the join output. With a third level of indirect addressing, we avoid indexing the whole set of rows again and instead point to the relevant relation retrieval query. Using our IA-MME primitive, the modification to achieve $\mathsf{PP3}$ is very minimal, as demonstrated in the pseudocode in Fig. 25. Note that $\mathsf{PP3.KS} = \mathsf{IAMME.KS} \times \mathsf{SE.KS}$.

Alg FP2.Enc(K, DS)

For $(id, (r_1, \ldots, r_n)) \in DS$ do
  For $i = 1, \ldots, n$ do
    $\mathbf{M}[0\|id\|i] \leftarrow \mathsf{Part}(r_i)$
  $\mathbf{M}[1\|(\mathtt{r}, id)] \leftarrow (1\|id\|i)$
For $(\mathtt{j}, \mathsf{jpred}, id_1, id_2) \in \mathsf{SQLdt.JnQs}$ do
  $\mathbf{r} \leftarrow ()$; $(r_1, \ldots, r_n) \leftarrow DS[id_1]$
  $(r'_1, \ldots, r'_n) \leftarrow DS[id_2]$
  For $(i, j) \in [n] \times [n]$ do
    If $\mathsf{jpred}(r_i, r'_j) = 1$ then
      $\mathbf{r} \leftarrow \mathbf{r}\|(1\|id\|i, 1\|id\|j)$
  $\mathbf{M}[1\|(\mathtt{j}, \mathsf{jpred}, id_1, id_2)] \leftarrow \mathbf{r}$
Return $\mathsf{IAMME.Enc}(K, \mathbf{M})$

Alg FP2.Tok(K, q)

If $q = (\mathtt{r}, id)$ then
  return $(\mathtt{r}, \mathsf{IAMME.Tok}(K, 1\|q))$
Else return $(\mathtt{j}, \mathsf{IAMME.Tok}(K, 1\|q))$

Alg FP2.Eval$((x, tk), ED)$

Return $(x, \mathsf{IAMME.Eval}(tk, ED))$

Alg FP2.Dec$(K, (x, c))$

$\mathbf{r} \leftarrow \mathsf{IAMME.Dec}(K, c)$
$(r_1, \ldots, r_n) \leftarrow \mathsf{Unpart}(\mathbf{r})$
If $x = \mathtt{r}$ then return $(r_1, \ldots, r_n)$
Else return $((r_1, r_2), \ldots, (r_{n-1}, r_n))$

Algs PP2.Enc$((K, K^{\mathrm{s}}), DS)$, PP3.Enc$((K, K^{\mathrm{s}}), DS)$

For $(id, (r_1, \ldots, r_n)) \in DS$ do
  For $i = 1, \ldots, n$ do $\mathbf{M}[0\|id\|i] \leftarrow \mathsf{Part}(r_i)$
  $\mathbf{M}[1\|(\mathtt{r}, id)] \leftarrow (1\|id\|i)$
For $(\mathtt{j}, \mathsf{jpred}, id_1, id_2) \in \mathsf{SQLdt.JnQs}$ do
  $(r_1, \ldots, r_n) \leftarrow DS[id_1]$; $(r'_1, \ldots, r'_n) \leftarrow DS[id_2]$
  $\ell_1 \leftarrow 1\|\mathtt{L}\|(\mathtt{j}, \mathsf{jpred}, id_1, id_2)$; $\ell_2 \leftarrow 1\|\mathtt{R}\|(\mathtt{j}, \mathsf{jpred}, id_1, id_2)$
  $\mathbf{M}[\ell_1] \leftarrow \big(1\|id_1\|i : i \in [n]\ ,\ \exists j \in [n]\ ,\ \mathsf{jpred}(r_i, r'_j) = 1\big)$
  If $\mathbf{M}[\ell_1] = \mathbf{M}[1\|(\mathtt{r}, id_1)]$ then $\mathbf{M}[\ell_1] \leftarrow (1\|(\mathtt{r}, id_1))$
  $\mathbf{M}[\ell_2] \leftarrow \big(1\|id_2\|j : j \in [n]\ ,\ \exists i \in [n]\ ,\ \mathsf{jpred}(r_i, r'_j) = 1\big)$
  If $\mathbf{M}[\ell_2] = \mathbf{M}[1\|(\mathtt{r}, id_2)]$ then $\mathbf{M}[\ell_2] \leftarrow (1\|(\mathtt{r}, id_2))$
Return $\mathsf{IAMME.Enc}(K, \mathbf{M})$

Algs PP2.Tok$((K, K^{\mathrm{s}}), q)$,PP3.Tok$((K, K^{\mathrm{s}}), q)$

If $q = (\mathtt{r}, id)$ then return $(\mathtt{r}, \mathsf{IAMME.Tok}(K, 1\|q))$
Else If $q = (\mathtt{j}, \mathsf{jpred}, id_1, id_2)$ then
  Return $\big(\mathtt{j}, \mathsf{IAMME.Tok}(K, 1\|\mathtt{L}\|q), \mathsf{IAMME.Tok}(K, 1\|\mathtt{R}\|q), \mathsf{SE.Enc}(K^{\mathrm{s}}, \mathsf{jpred})\big)$

Algs PP2.Eval(tk, ED),PP3.Eval(tk, ED)

If $tk = (\mathtt{r}, tk')$ then return $(\mathtt{r}, \mathsf{IAMME.Eval}(tk, ED))$
Else if $tk = (\mathtt{j}, tk_1, tk_2, c)$ then
  Return $(\mathtt{j}, \mathsf{IAMME.Eval}(tk_1, ED), \mathsf{IAMME.Eval}(tk_2, ED), c)$

Algs PP2.Dec$((K, K^{\mathrm{s}}), c)$,PP3.Eval(tk, ED)

If $c = (\mathtt{r}, c')$ then return $\mathsf{Unpart}\big(\mathsf{IAMME.Dec}(K, c')\big)$
Else if $c = (\mathtt{j}, c_1, c_2, c_3)$ then
  $\mathbf{r}_1 \leftarrow \mathsf{Unpart}\big(\mathsf{IAMME.Dec}(K, c_1)\big)$; $\mathbf{r}_2 \leftarrow \mathsf{Unpart}\big(\mathsf{IAMME.Dec}(K, c_2)\big)$
  Return $\mathsf{Join}(\mathbf{r}_1, \mathbf{r}_2, \mathsf{SE.Dec}(K^{\mathrm{s}}, c_3))$

Fig. 25. Three StE schemes for SQLdt (relation retrievals and joins on SQL data). FP2 and PP2 use depth-two uniform indirect addressing while PP3 uses depth-three non-uniform indirect addressing.

Data	Scheme	LMM				SMM
		$\mathbf{M}_0$	$\mathbf{M}_1$	$\mathbf{M}_2$	$\mathbf{M}_3$	$\mathbf{M}$
2021 ePrint	SE2	–	9.388e7	6.542e3	–	9.389e7
2020 ePrint	SE2	–	9.024e7	6.197e3	–	9.024e7
TPC-H (10MB)	FP2	–	6.893e5	1.039e7	–	1.108e7
TPC-H (10MB)	PP2	–	6.893e5	3.357e5	–	1.025e6
TPC-H (10MB)	PP3	18	6.893e5	8.781e4	20	7.772e5
TPC-H (1GB)	FP2	–	7.165e7	1.155e9	–	1.227e9
TPC-H (1GB)	PP2	–	7.165e7	3.349e7	–	1.051e8
TPC-H (1GB)	PP3	18	7.165e7	8.761e6	20	8.041e7

Fig. 26. Full simulation results computing the sizes of unencrypted data structures when using LMM and SMM. Sizes are computed in blocks of 128-bits (black) or 130-bits (in blue). These demonstrate that SMM leaks less and is more storage efficient.

H Simulation Details

In this section, we provide additional details about our simulations. Our source code and full results can be obtained from [4].

ePrint dataset details. We used a Python script to scrape the ePrint keywords and PDFs for 2020 and 2021 from the online archive [18]. To ensure all documents were accessible, we add the author-selected "category" to the documents' keywords in the event that the authors did not identify any.

TPC-H dataset details. We obtained the TPC-H tool [38] and generated datasets with scale factor 1 (approx 1GB) and scale factor 0.01 (approx 10MB). We used Python to convert this into CSV format and perform the analysis. We followed the TPC-H schema to determine a set of equijoins to support [37]. The summary of the relations and joins used can be found in [4].

Additional simulation details. In all our simulations, we break up depth-1 values (i.e. PDF documents and rows from each relation) into 128-bit blocks to accurately portray how a real-world application might store such payloads. For token values, we assume that each token is also 128-bits.

For the StE schemes for SQLdt, the LMM technique used by FP2, PP2 is consistent with $\mathsf{LMM}_{\mathrm{u}}$ while that of PP3 is consistent with LMM. Additionally, we assume that depth indicators are used in LMM for all multimaps except $\mathbf{M}_1$. All of these choices were done to minimize server storage.

Full results. In Fig. 26 we provide the full simulation results.