Structured Encryption and Controlled Disclosure

Melissa Chase and Seny Kamara

Microsoft Research
{melissac,senyk}@microsoft.com

Abstract. We consider the problem of encrypting structured data (e.g., a web graph or a social network) in such a way that it can be efficiently and privately queried. For this purpose, we introduce the notion of structured encryption which generalizes previous work on symmetric searchable encryption (SSE) to the setting of arbitrarily-structured data.
We present a model for structured encryption, a formal security definition and several efficient constructions. We present schemes for performing queries on two simple types of structured data, specifically lookup queries on matrix-structured data, and search queries on labeled data. We then show how these can be used to construct efficient schemes for encrypting graph data while allowing for efficient neighbor and adjacency queries.
Finally, we consider data that exhibits a more complex structure such as labeled graph data (e.g., web graphs). We show how to encrypt this type of data in order to perform focused subgraph queries, which are used in several web search algorithms. Our construction is based on our labeled data and basic graph encryption schemes and provides insight into how several simpler algorithms can be combined to generate an efficient scheme for more complex queries.

1 Introduction

The most common use of encryption is to provide confidentiality by hiding all useful information about the plaintext. Encryption, however, often renders data useless in the sense that one loses the ability to operate on it. In certain settings this is undesirable and one would prefer encryption schemes that allow for some form of computation over encrypted data.

One example is in the context of remote data storage, or so-called "cloud storage", where a data owner wishes to store *structured* data (e.g., a collection of web pages) on an untrusted server and only retain a constant amount of information locally. To guarantee confidentiality, the owner could encrypt the data before sending it to the server but this approach is unsatisfactory because the data loses its structure and, in turn, the owner loses the ability to query it efficiently.

To address this problem we introduce the notion of *structured encryption*. A structured encryption scheme encrypts structured data in such a way that it can be queried through the use of a query-specific token that can only be generated

with knowledge of the secret key. In addition, the query process reveals no useful information about either the query or the data. An important consideration in this context is the efficiency of the query operation on the server side. In fact, in the context of cloud storage, where one often works with massive datasets, even linear time operations can be infeasible.

Roughly speaking, we view structured data as a combination of a data structure δ and a sequence of data items $\mathbf{m} = (m_1, \ldots, m_n)$ such that δ encodes the data's structure and $\mathbf{m}$ represents the actual data. For example, in the case of graph-structured data such as a social network, δ is a graph with n nodes and the ith element of $\mathbf{m}$ is the data associated with node i. To query the data efficiently, one queries δ to recover a set of pointers $I \subseteq [1, n]$ and then retrieves the items in $\mathbf{m}$ indexed by I.

At a high level, a structured encryption scheme takes as input structured data $(\delta, \mathbf{m})$ and outputs an encrypted data structure γ and a sequence of ciphertexts $\mathbf{c} = (c_1, \ldots, c_n)$. Using the private key, a token τ can be constructed for any query such that pointers to the encryptions of $(m_i)_{i \in I}$ can be recovered from γ and τ. Furthermore, given the private key, one can decrypt any ciphertext c_i.

A certain class of symmetric searchable encryption (SSE) schemes [18, 11, 15] can be viewed as structured encryption schemes for the special purpose of private keyword search over encrypted document collections. Of course, the functionality provided by structured encryption can be achieved using general techniques like oblivious RAMs [20], secure two-party computation [36] and fully-homomorphic encryption (FHE) [17]. In our context, however, we are interested in solutions that are non-interactive and, at worst, linear in the number of data items as opposed to linear in the length of the data. All the schemes described in this work are non-interactive and optimal in that the query time is linear in the number of data items to be returned.

Informally, a basic notion of security for structured encryption guarantees that (1) an encrypted data structure γ and a sequence of ciphertexts $\mathbf{c}$ reveal no partial information about the data $\mathbf{m}$; and that (2) given, in addition, a sequence of tokens $(\tau_1, \ldots, \tau_t)$ for queries $\mathbf{q} = (q_1, \ldots, q_t)$ no information is leaked about either $\mathbf{m}$ or $\mathbf{q}$ beyond what can be inferred from some limited leakage which is a function of δ, $\mathbf{m}$ and $\mathbf{q}$. A stronger notion, introduced in [15], guarantees that (2) holds even when the queries are generated *adaptively*.

All known constructions that can be considered efficient structured encryption schemes (i.e., the index-based SSE schemes [18, 11, 15]) reveal some limited information about the data items and queries. In particular, for any query they reveal at least (1) the *access pattern*, which consists of the pointers I; and (2) the *query pattern*, which reveals whether two tokens were for the same query [1].

1.1 Applications of Structured Encryption

Private queries on encrypted data. The most immediate application of structured encryption is for performing private queries on encrypted data. In this setting,

[1] While the public-key encryption scheme with keyword search of [7] yields a SSE scheme that hides the access and query patterns, it is interactive.

a client encrypts its (structured) data $(\delta, \mathbf{m})$ resulting in an encrypted data structure γ and a sequence of ciphertexts $\mathbf{c}$. It then sends $(\gamma, \mathbf{c})$ to the server. Whenever the client wishes to query the data, it sends a token τ to the server which the latter uses to recover pointers J to the appropriate ciphertexts. Using a structured encryption scheme in this manner enables the client to store its data remotely while simultaneously guaranteeing confidentiality against the server (in the sense outlined above) and efficient querying and retrieval. While this problem has received considerable attention for the special case of document collections [33, 18, 5, 35, 11, 1, 15, 3, 31, 7], as far as we know, it has never been considered for other kinds of data.

Controlled disclosure for local algorithms. While the original motivation for structured encryption was to perform private queries on encrypted data (or more precisely, private *searches* on encrypted data), we introduce here a new application which we refer to as *controlled disclosure*.

In this setting, the client not only wants to store its data remotely but expects the server (or some third party) to perform some computation over the data. In particular, while the client is willing to reveal the information necessary for the server to perform its task, the client does not want to reveal anything else. Consider, e.g., a client that stores a large-scale social network remotely and that, at some point, needs the server to analyze a small subset of the network. If the social network were encrypted using a classical encryption scheme the client would have to reveal the entire network, leaking extra information to the server. Ideally, what we want in this setting is a mechanism that allows the client to encrypt the data and later disclose the "pieces" of it that are necessary for the server to perform its task.

Another application of controlled disclosure is to the emerging area of (cloud-based) data brokerage services, such as Microsoft's Dallas [14] and Infochimps [23]. Here, the cloud provider acts as a broker between a data provider that wishes to sell access to a massive dataset and a data consumer that needs access to the data. The data is stored "in the cloud" and the cloud operator manages the consumer's access to the provider's data. Using controlled disclosure, the provider could encrypt its data before storing it in the cloud and release tokens to the consumer as appropriate. Such an approach would have several advantages including (1) enabling the producer to get an accurate measure of the consumer's use of the data; and (2) ensuring the producer that the consumer can only access the authorized segments of data, even if the consumer and the cloud operator collude.

Clearly, if the algorithm executed by the server (or the data consumer) is "global", in the sense that it needs to read all the data, then controlled disclosure provides no security. On the other hand, if the algorithm is "local", in that it only needs to read part of the data, then controlled disclosure preserves the confidentiality of the remaining data. There are numerous algorithms that exhibit this kind of local behavior and they are used extensively in practice to solve a variety of problems. For example, many optimization problems like the traveling salesman problem or vertex cover are handled in practice using local search al-

gorithms (e.g., hill climbing, genetic algorithms or simulated annealing). Several link-analysis algorithms for web search such as Kleinberg's seminal HITS algorithm [26] (and the related SALSA [27] algorithm) are local. Finally, the recent work of Brautbar and Kearns on "jump and crawl" algorithms [10] motivates and proposes several local algorithms for social network analysis, including for finding vertices with high-degree and high clustering coefficient.

Controlled disclosure can be viewed as a compromise between full security on the one hand and efficiency and functionality on the other. In settings where computation needs to be performed on massive datasets and "fully secure" solutions like multi-party computation [36, 19, 13] and fully-homomorphic encryption [17] are prohibitively expensive, controlled disclosure provides a practical solution without completely compromising security.

1.2 Our Results

Performing private queries on encrypted data is an important goal that is well motivated by the recent trend towards cloud storage. Giving clients the means to encrypt their data without losing the ability to efficiently query and retrieve it provides obvious benefits to the client but also frees the cloud provider from many legal exposures (see [2, 24, 32] for discussion of these issues). It additionally provides a mechanism by which clients from regulated industries can make use of cloud storage (e.g., to store medical records or financial documents) while remaining compliant.

While the recent work on searchable encryption constitutes an important step towards this goal, we note that a noticeable fraction of the data generated today is *not* text data. Indeed, many large-scale datasets (e.g., image collections, social network data, maps or location information) exhibit a different and sometimes more complex structure that cannot be handled properly using searchable encryption. To address this, we:

1. introduce the notion of structured encryption, which generalizes index-based symmetric searchable encryption [18, 11, 15] to arbitrarily-structured data and propose a novel application of structured encryption (and therefore of SSE) to the problem of controlled disclosure.
2. extend the adaptive security definition of [15] to the setting of structured encryption,
3. give constructions of adaptively-secure structured encryption schemes for a variety of structures and queries including:
   (a) (lookup queries on matrix-structured data) given a matrix and pair (i, j), return the value stored at row i and column j. This captures, e.g., lookup queries on digital images or retrieval of maps.
   (b) (search queries on labeled data) given a set of labeled items and keyword w, return the items labeled with w. This captures the familiar setting of searchable encryption. We briefly note that our construction exhibits a combination of useful properties that, as far as we know, no previous scheme achieves.

(c) (neighbor queries on graph-structured data) given a graph and a node i, return all the nodes adjacent to i. This captures, e.g., retrieving a user's "friend list" in a social network.
(d) (adjacency queries on graph-structured data) given a graph and two nodes i and j, return 1 if they are adjacent and return 0 otherwise. This captures, e.g., testing whether two users are "friends" in a social network.

While the previous constructions are useful in their own right, an important goal with respect to structured encryption is to construct schemes that are able to encrypt complex structures and to handle expressive queries that take full advantage of the complexity of the data's structure. As an example, consider the case of web graphs (i.e., subgraphs of the Web) which are composed of pages with both text and hyperlinks. Encrypting the pages of a web graph using a searchable encryption scheme will only enable keyword search over the encrypted pages. Web graphs, however, exhibit a much richer structure and we typically want to perform more complex queries on them. Towards this goal, our final contribution is to show how to encrypt web graphs and, more generally, what we refer to as *labeled graph data*. In particular, we:

4. give a structured encryption scheme for labeled graphs that handles *focused subgraph* queries. Roughly speaking, for a given search keywork, a focused subgraph query on a web graph returns a subgraph that encodes enough information about it to yield a good ranking of the pages for that search. These queries are an essential part of Kleinberg's seminal HITS algorithm [26] (and its many successors).
   Our construction uses as building blocks some of the schemes mentioned above. We stress, however, that it is not sufficient to use the schemes "as-is" and we show a novel way of combining structured encryption schemes for simple structures in order to build schemes that handle more complex data and more expressive queries. The approach is general and can be adapted to other complex data types.

2 Related Work

We already mentioned work on oblivious RAMs, secure two-party computation and FHE so we restrict the following discussion to searchable and functional encryption.

Searchable encryption. As mentioned above, structured encryption is a generalization of the notion of a secure index first proposed by Goh [18] for the purpose of building symmetric searchable encryption schemes [33]. In [18], Goh gives a formal security definition for secure indexes and a construction based on Bloom filters. This was followed by [11] and [15], the latter of which gave stronger security definitions and more efficient constructions. Our security definitions for structured encryption in section 4 generalize the ones in [15] to

arbitrarily-structured data. Searchable encryption has also been considered in the public-key setting [5, 35, 1, 9, 7, 8].

Functional encryption. Functional encryption [34] is a recent paradigm that generalizes work on a variety of problems including identity-based encryption [29, 6], attribute-based encryption [28, 22, 4], and predicate encryption [25, 30].

Roughly speaking, a structured encryption scheme can be viewed as a functional encryption scheme for which a token can only be used on a single ciphertext. We provide a more detailed comparison between the two approaches in the full version [12].

3 Notation and Preliminaries

Notation. Given a sequence $\mathbf{v}$ of n elements, we refer to its i^{th} element as v_i. If f is a function with domain $\mathcal{U}$ and $S \subseteq \mathcal{U}$, then $f[S]$ refers to the image of S under f. The set of all $\lambda_1 \times \lambda_2$ matrices over a set S is denoted $S^{\lambda_1 \times \lambda_2}$. $\mathcal{G}_n$ and $\boldsymbol{\mathcal{G}}_n$ are the sets of all undirected and directed graphs of size n, respectively. An undirected graph $G = (V, E)$ consists of a set of vertices V and a set of edges $E = \{(i, j)\}$ where $i, j \in V$. We denote by $\deg(i)$ the degree of node i. If G is directed, then the pairs (i, j) are ordered and we refer to i as the tail and to j as the head of the edge. In addition, we denote i's in and out degrees by $\deg^-(i)$ and $\deg^+(i)$, respectively.

Data types. An *abstract data type* is a collection of objects together with a set of operations defined on those objects. For simplicity and visual clarity we define data types as having a single operation but this can be extended to model data types with multiple operations in the natural way. Formally, a data type $\mathscr{T}$ is defined by a universe $\mathcal{U} = \{\mathcal{U}_k\}_{k \in \mathbb{N}}$ and an operation $\mathsf{Query} : \mathcal{U} \times \mathcal{Q} \to \mathcal{R}$, where $\mathcal{Q} = \{\mathcal{Q}_k\}_{k \in \mathbb{N}}$ is the operation's query space and $\mathcal{R} = \{\mathcal{R}_k\}_{k \in \mathbb{N}}$ is its response space. The universe, query and response spaces are ensembles of finite sets indexed by the security parameter k. In this work, we assume the universe is a totally ordered set, and that the response space includes a special element $\perp$ denoting failure.

4 Definitions

In this section we formalize structured encryption schemes and present our main security definition. Before doing so, however, we make explicit two properties of structured encryption which we will make use of throughout this work.

Induced permutation. Unlike previous work on searchable encryption we choose to include the data items (i.e., the documents in the case of searchable encryption) and their encryptions in our definitions. We prefer this approach because explicitly capturing each component of the system can bring to light subtle interactions between them. As an example, consider the correlation between the

location of the data items in the sequence $\mathbf{m}$ and the locations of their corresponding ciphertexts in $\mathbf{c}$. More precisely, let π be the permutation over $[n]$ such that for all $i \in [n]$, $m_i := \mathsf{Dec}_K(c_{\pi(i)})$. We refer to π as the permutation *induced* by $\mathbf{m}$ and $\mathbf{c}$.

The reason most SSE constructions (with the exception of oblvious RAMs) leak the access pattern is because π is the identity function. This means that in order to (efficiently) retrieve items $\{m_i : i \in I\}$ the server must know I. Our constructions hide part of the access pattern essentially because they break this correlation by inducing a (pseudo-)random permutation between $\mathbf{m}$ and $\mathbf{c}$.

Associativity. We also make explicit a property possessed by some constructions (e.g., the non-adaptively secure SSE construction of [15]) that we refer to as *associativity*. Intuitively, a scheme is associative if one can associate an item v_i with data item m_i in such a way that a query operation returns, in addition to the pointers J, the strings $(v_i)_{i \in I}$. We capture this by re-defining the message space of our encryption algorithms to take, in addition to a data structure δ, a sequence $\mathbf{M} = ((m_1, v_1), \ldots, (m_n, v_n))$ of pairs that consist of a private data item m_i and a semi-private [2] item v_i . We sometimes refer to the sequences $(m_1, \ldots, m_n)$ and $(v_1, \ldots, v_n)$ as $\mathbf{m}$ and $\mathbf{v}$, respectively.

Associativity is useful for several reasons. The most direct application is to provide the client the ability to associate some meta-data with the ciphertexts that may be useful to the server (e.g., file name or size). In situations where the client wishes to grant the server access to the data, the semi-private items could even be decryption keys for the associated ciphertexts. As we will see in Section 6, however, associativity can also be used to "chain" structured encryption schemes together in order to construct complex schemes from for simpler ones.

Definition 1 (Private-key structured encryption). *Let $\mathcal{T}$ be an abstract data type supporting operation $\mathsf{Query} : \mathcal{U} \times \mathcal{Q} \to \mathcal{R}$ where $\mathcal{R} = [n]$ for $n \in \mathbb{N}$. An associative private-key structured encryption scheme for $\mathcal{T}$ is a tuple of five polynomial-time algorithms $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Query}_\mathsf{e}, \mathsf{Dec})$ such that:*

$K \leftarrow \mathsf{Gen}(1^k)$: is a probabilistic algorithm that takes as input a security parameter k and outputs a private key K.
$(\gamma, \mathbf{c}) \leftarrow \mathsf{Enc}(K, \delta, \mathbf{M})$: is a probabilistic algorithm that takes as input a private key K, a data structure δ of type $\mathcal{T}$, and a sequences of private and semi-private data $\mathbf{M}$. It outputs an encrypted data structure γ and a sequence of ciphertexts $\mathbf{c}$. We sometimes write this as $(\gamma, \mathbf{c}) \leftarrow \mathsf{Enc}_K(\delta, \mathbf{M})$.
$\tau \leftarrow \mathsf{Token}(K, q)$: is a (possibly probabilistic) algorithm that takes as input a private key K and a query $q \in \mathcal{Q}$ and outputs a token τ. We sometimes write this as $\tau \leftarrow \mathsf{Token}_K(q)$.
$(J, \mathbf{v}_I) := \mathsf{Query}_\mathsf{e}(\gamma, \tau)$: is a deterministic algorithm that takes as input an encrypted data structure γ and a token τ. It outputs a set of pointers $J \subseteq [n]$ and a sequence of semi-private data $\mathbf{v}_I = (v_i)_{i \in I}$, where $I = \pi^{-1}[J]$.

[2] We refer to the items $(v_1, \ldots, v_n)$ as semi-private since, unlike $(m_1, \ldots, m_n)$, they can be recovered given an appropriate token.

$m_j := \mathsf{Dec}(K, c_j)$*: is a deterministic algorithm that takes as input a secret key* K *and a ciphertext* c_j *and outputs a message* m_j*.*

We say that Π *is correct if for all* $k \in \mathbb{N}$*, for all* K *output by* $\mathsf{Gen}(1^k)$*, for all* $\delta \in \mathcal{U}_k$*, for all* $\mathbf{M}$*, for all* $(\gamma, \mathbf{c})$ *output by* $\mathsf{Enc}(K, \delta, \mathbf{M})$*, for all* $q \in \mathcal{Q}_k$*, for all* τ *output by* $\mathsf{Token}(K, q)$*, for* $(J, \mathbf{v}_I)$ *output by* $\mathsf{Query}_{\mathsf{e}}(\gamma, \tau)$*,*

$$J = \pi\left[\mathsf{Query}(\delta, q)\right] \bigwedge \mathsf{Dec}_K(c_j) = m_j \text{ for all } j \in [n],$$

where π *is the permutation induced by* $\mathbf{m}$ *and* $\mathbf{c}$*.*

The intuitive security guarantee we seek is that (1) given an encrypted data structure γ and a sequence of ciphertexts $\mathbf{c}$, no adversary can learn any partial information about $\mathbf{m}$; and that (2) given, in addition, a sequence of tokens $\tau = (\tau_1, \ldots, \tau_t)$ for an adaptively generated sequence of queries $\mathbf{q} = (q_1, \ldots, q_t)$, no adversary can learn any partial information about either $\mathbf{m}$ or $\mathbf{q}$ beyond what is revealed by the semi-private data $(\mathbf{v}_{I_1}, \ldots, \mathbf{v}_{I_t})$.

This exact intuition can be difficult to achieve and in some settings is unnecessarily strong. Consider, e.g., the fact that the number of data items n is immediately revealed to the adversary since it receives the ciphertexts $\mathbf{c}$. Another example is in the setting of SSE where, as discussed earlier, all known efficient and non-interactive schemes [18, 11, 15] reveal the access and query patterns. We would therefore like to weaken the definition appropriately by allowing some limited information about the messages and the queries to be revealed. On the other hand, it is not clear that such leakage is always necessary in order to achieve efficiency (e.g., the number of data items can be easily hidden by padding) so we prefer not to "hardcode" this leakage in our definition. To formalize this we parameterize the definition with two leakage functions $\mathcal{L}_1$ and $\mathcal{L}_2$ that capture precisely what is being leaked by the ciphertext and the tokens.

We now present our security definition for adaptive adversaries which is a generalization of the definition of [16]. Intuitively, we require that the view of an adversary (i.e., the encrypted data structure, the sequence of ciphertexts, and the sequence of tokens) generated from any adaptive query strategy be simulatable given the leakage information and the semi-private data.

Definition 2 (CQA2-security). *Let* $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Query}_{\mathsf{e}}, \mathsf{Dec})$ *be an associative private-key structured encryption scheme for data of type* $\mathcal{T}$ *supporting operation* $\mathsf{Query} : \mathcal{U} \times \mathcal{Q} \to [n]$*, for some* $n \in \mathbb{N}$*, and consider the following probabilistic experiments where* $\mathcal{A}$ *is an adversary,* $\mathcal{S}$ *is a simulator and* $\mathcal{L}_1$ *and* $\mathcal{L}_2$ *are (stateful) leakage algorithms:*

Real$_{\Sigma,\mathcal{A}}(k)$*: the challenger begins by running* $\mathsf{Gen}(1^k)$ *to generate a key* K*.* $\mathcal{A}$ *outputs a pair* $(\delta, \mathbf{M})$ *and receives* $(\gamma, \mathbf{c}) \leftarrow \mathsf{Enc}_K(\delta, \mathbf{M})$ *from the challenger. The adversary makes a polynomial number of adaptive queries and, for each query* q*, receives a token* $\tau \leftarrow \mathsf{Token}_K(q)$ *from the challenger. Finally,* $\mathcal{A}$ *returns a bit* b *that is output by the experiment.*

$\mathbf{Ideal}_{\Sigma,\mathcal{A},\mathcal{S}}(k)$: *$\mathcal{A}$ outputs a tuple $(\delta, \mathbf{M})$. Given $\mathcal{L}_1(\delta, \mathbf{M})$, $\mathcal{S}$ generates and sends a pair $(\gamma, \mathbf{c})$ to $\mathcal{A}$. The adversary makes a polynomial number of adaptive queries and for each query q the simulator is given $(\mathcal{L}_2(\delta, q), \mathbf{v}_I)$, where $I := \mathsf{Query}(\delta, q)$. The simulator returns a token τ. Finally, $\mathcal{A}$ returns a bit b that is output by the experiment.*

We say that Σ is $(\mathcal{L}_1, \mathcal{L}_2)$-secure against adaptive chosen-query attacks if for all PPT *adversaries $\mathcal{A}$, there exists a* PPT *simulator $\mathcal{S}$ such that*

$$|\Pr[\,\mathbf{Real}_{\Sigma,\mathcal{A}}(k) = 1\,] - \Pr[\,\mathbf{Ideal}_{\Sigma,\mathcal{A},\mathcal{S}}(k) = 1\,]| \leq \mathsf{negl}(k).$$

As previously discussed, the $\mathcal{L}_2$ leakage of our constructions mainly consists of the query and intersection patterns. Intuitively, the query pattern reveals when a query is repeated while the intersection pattern reveals when the same items are accessed. The intersection pattern reveals when the same items are accessed but not *which* items are accessed (i.e., their locations in $\mathbf{m}$). The latter is hidden in our definition below by applying a random permutation to the item's locations in $\mathbf{m}$.

Definition 3 (Query and intersection patterns). *Let $\mathbf{q}$ be a non-empty sequence of queries. For any $q_t \in \mathbf{q}$, the query pattern* $\text{QP}(q_t)$ *is a binary vector of length t with a 1 at location i if $q_t = q_i$ and a 0 otherwise. The intersection pattern* $\text{IP}(q_t)$ *is a sequence of length t with $f[I]$ at location t, where f is a fixed random permutation over $[n]$ and $I := \mathsf{Query}(\delta, q_t)$.*

5 Structured Encryption for Basic Structures

In this Section we present constructions of structured encryption schemes for data with simple structures. In Section 6 we will use some of these as building blocks to design schemes for data that exhibits a more complex structure. We stress, however, that the constructions presented here are of independent interest.

5.1 Lookup Queries on Matrices

We describe a structured encryption scheme for matrix-structured data which consists of an $\lambda_1 \times \lambda_2$ matrix M of pointers into a sequence of n data items $\mathbf{m}$. Here, the matrix type has universe $\mathcal{U} = [n]^{\lambda_1 \times \lambda_2}$ and supports the lookup operation $\mathsf{Lkp} : [n]^{\lambda_1 \times \lambda_2} \times [\lambda_1] \times [\lambda_2] \to [n]$ that takes as input a matrix M and a pair (α, β) and returns $M[\alpha, \beta]$.

Matrix-structured data is ubiquitous and includes any kind of two-dimensional data. Consider, e.g., the case of digital images which can be viewed as a pair $(M, \mathbf{m})$, where M is a matrix such that the cell at location (α, β) points to some m_i that encodes the color of the pixel at location (α, β) in the image.

Our construction, described in Figure 1 below, is associative. At a high level, encryption is done by (1) padding the data items to be of the same length; (2) randomly permuting the location of the data items, (3) randomly permuting the

location of the matrix cells using a PRP; and (4) encrypting the contents of the cells (and the semi-private data) using the output of a PRF. The purpose of the last two steps are immediate. Steps (1) and (2) are what allow us hide part of the access pattern by inducing a pseudo-random permutation between $\mathbf{m}$ and $\mathbf{c}$.

Lookup queries are handled by sending the permuted location of a cell (which can be recovered by the client since it stores the key to the PRP) and the output of the PRF used to encrypt the contents (which can also be recovered since the client stores the key to the PRF).

Let $F : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^*$ be a pseudo-random function, $P : \{0,1\}^k \times [\lambda_1] \times [\lambda_2] \to [\lambda_1] \times [\lambda_2]$ be pseudo-random permutation and $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a private-key encryption scheme. Our encryption scheme $\mathsf{Matrix} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Lkp_e}, \mathsf{Dec})$ is defined as follows:

- $\mathsf{Gen}(1^k)$: generate two random k-bit strings K_1, K_2 and a key $K_3 \leftarrow \Pi.\mathsf{Gen}(1^k)$. Set $K := (K_1, K_2, K_3)$.
- $\mathsf{Enc}(K, M, \mathbf{M})$: construct a $\lambda_1 \times \lambda_2$ matrix C as follows:
  1. parse $\mathbf{M}$ as $\mathbf{m}$ and $\mathbf{v}$
  2. choose a pseudo-random permutation $G : \{0,1\}^k \times [n] \to [n]$
  3. sample a k-bit string K_4 uniformly at random
  4. for all $(\alpha, \beta) \in [\lambda_1] \times [\lambda_2]$,
     store $\langle G_{K_4}(i), v_i \rangle \oplus F_{K_1}(\alpha, \beta)$ where $i := M[\alpha, \beta]$, at location $(\alpha', \beta') := P_{K_2}(\alpha, \beta)$ in C.
     If $M[\alpha, \beta] = \perp$, then $\langle G_{K_4}(i), v_i \rangle$ above is replaced with a random string of appropriate length.

  Let $\mathbf{m}^*$ be the sequence that results from padding the elements of $\mathbf{m}$ so that they are of the same length and permuting them according to G_{K_4}. For $1 \leq j \leq n$, let $c_j \leftarrow \Pi.\mathsf{Enc}_{K_3}(m_j^*)$. Output $\gamma := C$ and $\mathbf{c} = (c_1, \ldots, c_n)$.
- $\mathsf{Token}(K, \alpha, \beta)$: output $\tau := (s, \alpha', \beta')$, where $s := F_{K_1}(\alpha, \beta)$ and $(\alpha', \beta') := P_{K_2}(\alpha, \beta)$.
- $\mathsf{Lkp_e}(\gamma, t)$: parse τ as (s, α', β'); compute and output $(j, v) := s \oplus C[\alpha', \beta']$.
- $\mathsf{Dec}(K, c_j)$: return $m_j := \Pi.\mathsf{Dec}_{K_3}(c_j)$.

Fig. 1. An associative structured encryption scheme for matrices.

In Theorem 1 below we show that the construction above is secure against adaptive chosen-query attacks.

Theorem 1. *If F, P and G are pseudo-random and if Π is CPA-secure then Matrix is $(\mathcal{L}_1, \mathcal{L}_2)$-secure against adaptive chosen-query attacks, where $\mathcal{L}_1(M, \mathbf{M}) = (\lambda_1, \lambda_2, n, \ell)$ and $\mathcal{L}_2(M, \alpha, \beta) = (\mathrm{QP}(\alpha, \beta), \mathrm{IP}(\alpha, \beta))$.*

The proof is omitted due to lack of space but appears in [12].

5.2 Search Queries on Labeled Data

We now present a structured encryption scheme for labeled data which consists of a "labeling" L and a sequence of data items $\mathbf{m}$. Informally, a labeling just associates a set of keywords to each data item. More formally, the labeling data type has as universe $\mathcal{U}$ the set of all binary relations between $[n]$ and W, where W is a set of keywords. In addition, it supports the operation $\mathsf{Search} : \mathcal{U} \times W \rightarrow 2^{[n]}$ that takes as input a labeling and a keyword w and returns the set $L(w) = \{i \in [n] : (i, w) \in L\}$.

Our construction, described in Figure 2, is efficient, associative and adaptively secure and, as far as we know, is the first scheme to achieve all three properties. It is based on the first scheme of [15] (SSE-1) which is efficient and associative but not adaptively secure [3]. The second scheme of [15], on the other hand, is adaptively secure but is inefficient and not associative.

Our construction makes use of a dictionary which is a data structure that stores pairs (a, b) such that given a, the corresponding value b can be recovered efficiently. We refer to a as the "search key" and to b as the value. Dictionaries can be implemented in a variety of ways, including using search trees or hash tables. Intuitively, encryption proceeds as follows in our scheme. As in our previous construction, we pad and permute the data items with a PRP G. For each keyword w an array is constructed where each cell stores (1) a pointer j from the set $L^*(w) = G_K[L(w)]$ and (2) the corresponding semi-private item v_i. The array is then padded up to a standard length, and encrypted using the output of a PRF and is stored in a dictionary using as search key the output of another PRF on the keyword. Search queries are handled by sending the search key (which can be recovered by the client using the key to the second PRF) and the output of the PRF used to encrypt the array (which can be recovered using the key to the first PRF). The efficiency of our search operation depends on how the underlying dictionary is implemented but in this context any solution based on hash tables is appropriate and will give search time that is $O(|I|)$, which is optimal.

Theorem 2. *If F, P and G are pseudo-random and if Π is CPA-secure then* Label *is $(\mathcal{L}_1, \mathcal{L}_2)$-secure against adaptive chosen-query attacks, where $\mathcal{L}_1(L, \mathbf{M}) = (|W|, n, \ell)$ and $\mathcal{L}_2(L, w) = (|I|, \mathrm{QP}(w), \mathrm{IP}(w))$.*

The proof is omitted due to lack of space but appears in [12].

5.3 Neighbor Queries on Graphs

We now consider encryption of graph-structured data and, in particular, of graphs that support neighbor queries. Formally, the graph type we consider has universe $\mathcal{U} = \mathcal{G}_n$ and supports the neighbor operation $\mathsf{Neigh} : \mathcal{G}_n \times [n] \rightarrow 2^{[n]}$

[3] While our scheme achieves the same efficiency as SSE-1 with respect to search time, SSE-1 is more efficient with respect to storage.

Let $F : \{0,1\}^k \times W \to \{0,1\}^*$ and $P : \{0,1\}^k \times W \to \{0,1\}^k$ be pseudo-random functions and $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a private-key encryption scheme. Our scheme $\mathsf{Label} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Search}_\mathsf{e}, \mathsf{Dec})$ is defined as follows:

- $\mathsf{Gen}(1^k)$: sample two random k-bit keys K_1, K_2, and generate a key $K_3 \leftarrow \Pi.\mathsf{Gen}(1^k)$. Set $K := (K_1, K_2, K_3)$.
- $\mathsf{Enc}(K, L, \mathbf{M})$: construct a dictionary T as follows:
  1. parse $\mathbf{M}$ as $\mathbf{m}$ and $\mathbf{v}$.
  2. choose a pseudo-random permutation $G : \{0,1\}^k \times [n] \to [n]$
  3. sample a k-bit string K_4 uniformly at random
  4. for each $w \in W$ such that $L(w) \neq \emptyset$, let $\kappa_w := P_{K_2}(w)$ and store $\langle (G_{K_4}(i), v_i)_{i \in L(w)} \rangle \oplus F_{K_1}(w)$ in T with search key κ_w. Use padding to ensure that the strings $\langle (G_{K_4}(i), v_i)_{i \in L(w)} \rangle$ are all of the same length.

  Let $\mathbf{m}^*$ be the sequence that results from padding the elements of $\mathbf{m}$ so that they are of the same length and permuting them according to G_{K_4}. For $1 \leq j \leq n$, let $c_j \leftarrow \Pi.\mathsf{Enc}_{K_3}(m_j^*)$. Output $\gamma := T$ and $\mathbf{c} = (c_1, \ldots, c_n)$.
- $\mathsf{Token}(K, w)$: output $\tau := (F_{K_1}(w), P_{K_2}(w))$.
- $\mathsf{Search}_\mathsf{e}(\gamma, \tau)$: parse τ as (α, β) and compute $s := T(\beta) \oplus \alpha$, where $T(\beta)$ refers to the value stored in T with search key β. If β is not in T then output $J = \emptyset$ and $v_I = \perp$. Otherwise parse s as $\langle (j_1, v_{i_1}), \ldots, (j_t, v_{i_t}) \rangle$ and output $J = (j_1, \ldots, j_t)$ and $\mathbf{v}_I = (v_{i_1}, \ldots, v_{i_t})$.
- $\mathsf{Dec}(K, c_j)$: output $m_j := \Pi.\mathsf{Dec}_{K_3}(c_j)$.

Fig. 2. An associative structured encryption scheme for labeled data.

that takes as input an undirected graph G with n nodes and a node i and returns the nodes adjacent to i.

Our approach here is to encode the graph as a labeling and to apply a structured encryption scheme for labeled data (such as the one described in the previous Section). Given some graph-structured data $(G, \mathbf{m})$, where $G = (V, E)$, we construct the labeled data $(L, \mathbf{m})$ such that L assigns to each data item m_i a label set corresponding to the set of nodes adjacent to the ith node. Neighbor queries are handled by sending a token for "keyword" $i \in V$ which allows the server to recover pointers to all the data items associated with i by the labeling. Our construction is described in detail in Figure 3 below.

Theorem 3. *If* Label *is* $(\mathcal{L}_1, \mathcal{L}_2)$*-secure against adaptive chosen-query attacks, then* Graph *is* $(\mathcal{L}_1, \mathcal{L}_2)$*-secure against adaptive chosen-query attacks as well.*

The theorem follows by construction. Note that if Label is instantiated with the scheme from Section 5.2, then $\mathcal{L}_1$ leaks the size of the graph, the number of data items and the length of the largest data item while $\mathcal{L}_2$ leaks the degree of the node and the query and intersection patterns.

We now discuss a slight variation of this construction to handle incoming and outgoing neighbor queries on directed graphs. This will be useful as a building

Let $\mathsf{Label} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Search}_\mathsf{e}, \mathsf{Dec})$ be an associative structured encryption scheme for labeled data. Our scheme $\mathsf{Graph} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Neigh}_\mathsf{e}, \mathsf{Dec})$ is defined as follows:

- $\mathsf{Gen}(1^k)$: generate and output $K \leftarrow \mathsf{Label.Gen}(1^k)$.
- $\mathsf{Enc}(K, G, \mathbf{M})$: parse $\mathbf{M}$ as $\mathbf{m}$ and $\mathbf{v}$ and construct a labeling L that associates to each m_i the set $\{j \in [n] : (i,j) \in E\}$, where E is the set of edges in G. Output $(\gamma, \mathbf{c}) \leftarrow \mathsf{Label.Enc}_K(L, \mathbf{M})$.
- $\mathsf{Token}(K, i)$: compute and output $\tau \leftarrow \mathsf{Label.Token}_K(i)$.
- $\mathsf{Neigh}_\mathsf{e}(\gamma, \tau)$: output $J := \mathsf{Label.Search}(\gamma, \tau)$.
- $\mathsf{Dec}(K, c_j)$: output $m_j := \mathsf{Label.Dec}_K(c_j)$.

Fig. 3. A structured encryption scheme for graphs supporting neighbor queries.

block for the construction we describe in Section 6. An incoming neighbor query is: given a node i return all the nodes that point to it; and an outgoing neighbor query is: given a node i return all the nodes that it points to. We stress that the changes we describe do not affect security in any way.

Consider the scheme $\mathsf{Graph}^+ = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Neigh}_\mathsf{e}, \mathsf{Dec})$ defined exactly as Graph except that the Enc algorithm constructs the labeling in the following manner: instead of associating a data item m_i to the set of nodes adjacent to node i, associate m_i to the nodes that are pointed to by node i. Similarly, a scheme Graph^- can be constructed by associating to data item m_i the set of nodes that point to node i.

5.4 Adjacency Queries on Graphs

In this Section we give a simple scheme to encrypt graphs supporting adjacency queries based on any matrix encryption scheme. The approach is straightforward and, at a high level, consists of encrypting the graph's adjacency matrix. Given data $(G, \mathbf{m})$, where $G = (V, E)$ is a directed graph of size n and each data item m_i is assigned to some edge in E, encryption proceeds as follows. We create a matrix M that holds at location (α, β) a pointer to the data item associated with edge $(\alpha, \beta) \in V$ (or $\perp$ when there is no such edge). We then use the matrix encryption scheme on $(M, \mathbf{m})$. Our construction is described in detail in Figure 4.

Theorem 4. *If* Matrix *is* $(\mathcal{L}_1, \mathcal{L}_2)$*-secure against adaptive chosen-query attacks, then so is* Graph*.*

Again, the theorem follows by construction. If Matrix is instantiated with the construction from Section 5.1, then $\mathcal{L}_1$ leaks the size of the graph, the number of edges [4] the number of data items and the length of the largest data item. $\mathcal{L}_2$ leaks the query and intersection patterns.

[4] The number of edges can be hidden by padding $\mathbf{m}$ with $n^2 - |E|$ random strings whose lengths are distributed similarly to real data items.

Let $\mathsf{Matrix} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Lkp_e}, \mathsf{Dec})$ be an associative encryption scheme for matrix-structured data. Our scheme $\mathsf{Graph} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Adj_e}, \mathsf{Dec})$ is defined as follows:

- $\mathsf{Gen}(1^k)$: generate and output $K \leftarrow \mathsf{Matrix.Gen}(1^k)$.
- $\mathsf{Enc}(K, G, \mathbf{M})$: construct a matrix M as follows: if $(\alpha, \beta) \in V$, then $M[\alpha, \beta]$ stores a pointer to the item assigned to edge (α, β); if $(\alpha, \beta) \notin V$ then $M[\alpha, \beta] = \perp$. Output $(\gamma, \mathbf{c}) \leftarrow \mathsf{Matrix.Enc}_K(M, \mathbf{M})$.
- $\mathsf{Token}(K, i, j)$: compute and output $\tau \leftarrow \mathsf{Matrix.Token}_K(i, j)$.
- $\mathsf{Adj_e}(\gamma, \tau)$: output $J := \mathsf{Matrix.Lkp_e}(\gamma, \tau)$.
- $\mathsf{Dec}(K, c_j)$: output $m_j := \mathsf{Matrix.Dec}_K(c_j)$.

Fig. 4. A structured encryption scheme for graphs supporting adjacency queries.

6 Structured Encryption for Labeled Graphs

In this Section we describe an adaptively secure structured encryption scheme for data that is both labeled and associated with a graph-structure. As an example, consider a web graph where each page is labeled with a set of keywords (which could be the set of all the words in the page) and points to a set of other pages. Another example is social network data which consists of user profiles (with some associated meta-data) that link to other users.

While the constructions from the previous Section can be used to encrypt this type of data, the queries they support (i.e., keyword search, adjacency, and neighbor queries) are limited in this setting since they are only relevant to part of the data's structure. Indeed, if we were to encrypt a web graph using a scheme for labeled data, then we could only perform keyword search. Similarly, if we were to use a graph encryption scheme that supports only neighbor queries then we could only retrieve pages that are linked from a particular page. But web graphs, and labeled graph data in general, exhibit a much richer structure and ideally we would like to design schemes that support more complex queries that take advantage of this structure.

Focused subgraph queries. One example of complex queries on web graphs are focused subgraph queries. These queries are an essential part of a certain class of search engine algorithms which includes Kleinberg's seminal HITS algorithm [26] and the SALSA algorithm [27]. At a high level, they work as follows. Given a keyword w a keyword search is performed over the web pages. This results in a subset of pages called the *root* graph. A focused subgraph is then constructed by adding all the pages that either link to pages in the root graph or are linked from pages in the root graph. An iterative algorithm is then applied to the focused subgraph which returns, for each page, a score that quantifies its relevance with respect to keyword w. The key property of these "link-analysis" algorithms (and the reason for their success) is that they take advantage not only of the information provided by the keywords associated with the pages, but also of the

implicit information embedded in the graph structure (i.e., the links) of the web graph.

Our approach. At a high level, our approach is to decompose the complex structure into simpler structures (e.g., in the case of a web graph into its graph and its labeling) and then use different structured encryption schemes to handle each "sub-structure". We note, however, that the sub-structures cannot be handled in isolation. In particular, for this approach to work the individual schemes have to be combined in a particular way. This is where we make essential use of associativity, which will allow us to "chain" the schemes together in order to obtain the functionality we want (this technique will be illustrated in our discussion below).

Our construction. We now illustrate our second approach for the case of web graphs but note that our construction applies to any labeled graph data. A detailed description of our construction is given in Figure 5. We note that it is not associative. A web graph will be viewed as a tuple $(G, L, \mathbf{m})$, which consists of a directed graph $G \in \boldsymbol{\mathcal{G}}_n$ of size n, a labeling L over a keyword space W, and text pages $\mathbf{m}$. The graph G encodes the link structure of the web graph and the labeling assigns keywords to each page [5]. The focused subgraph operation $\mathsf{Subgraph} : \boldsymbol{\mathcal{G}}_n \times W \to \mathcal{G}_{\leq n}$ takes as input a directed graph G of size n and a keyword w and returns the subgraph $G(w)$ that consists of (1) the nodes i in $L(w)$; (2) any node that links to the nodes in $L(w)$; and (3) any node that is linked from the nodes in $L(w)$.

Our construction makes use of three structured encryption schemes: Label that supports search over labeled data, Graph^- that supports incoming neighbor queries over graph-structured data, and Graph^+ that supports outgoing neighbor queries over graph-structured data. We stress that Label must be associative. Given a web graph $(G, L, \mathbf{m})$ we encrypt $(G, \mathbf{m})$ using both Graph^+ and Graph^-, resulting in ciphertexts $\mathbf{c}^+$ and $\mathbf{c}^-$. Now, for each node i in G, we generate a pair of tokens (τ_i^+, τ_i^-). We then use Label to encrypt $(L, \mathbf{m})$ using the token pairs (τ_i^+, τ_i^-) as semi-private data (recall that Label is associative). We then output the encryption $\mathbf{c}^{\mathsf{l}}$ of $(L, \mathbf{m})$.

A focused subgraph query on keyword w is handled as follows. A token $\tau^{\mathsf{l}} \leftarrow \mathsf{Label.Token}_K(w)$ is generated and sent to the server. When used with the ciphertext $\mathbf{c}^{\mathsf{l}}$, this token will reveal to the server (1) pointers to all the (encrypted) web pages labeled with keyword w; and (2) for each of these encrypted pages c_j, the semi-private information which consists of tokens (τ_j^+, τ_j^-). For each encrypted page, the server can then use the token pairs with ciphertexts $\mathbf{c}_j^+$ and $\mathbf{c}_j^-$ to recover pointers to any incoming and outgoing neighbors of page c_j.

Theorem 5. *If* Label, Graph^+ *and* Graph^- *are respectively (stateless)* $(\mathcal{L}_1^l, \mathcal{L}_2^l)$*-secure,* $(\mathcal{L}_1^+, \mathcal{L}_2^+)$*-secure and* $(\mathcal{L}_1^-, \mathcal{L}_2^-)$*-secure against adaptive chosen query at-*

[5] If we wish to perform full-text search then the labeling can simply assign a page to all of its words.

Let $\mathsf{Label} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Search}_\mathsf{e}, \mathsf{Dec})$ be an encryption scheme for labeled data, $\mathsf{Graph}^+ = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Neigh}_\mathsf{e}, \mathsf{Dec})$ and $\mathsf{Graph}^- = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Neigh}_\mathsf{e}, \mathsf{Dec})$ be graph encryption schemes that support neighbor queries. Our scheme $\mathsf{LabGraph} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Subgraph}_\mathsf{e}, \mathsf{Dec})$ is defined as follows:

- $\mathsf{Gen}(1^k)$: generate three keys $K_1 \leftarrow \mathsf{Graph}^+.\mathsf{Gen}(1^k)$, $K_2 \leftarrow \mathsf{Graph}^-.\mathsf{Gen}(1^k)$ and $K_3 \leftarrow \mathsf{Label}.\mathsf{Gen}(1^k)$. Let $K = (K_1, K_2, K_3)$.
- $\mathsf{Enc}(K, G, \mathbf{m})$:
  1. compute $(\gamma^+, \mathbf{c}^+) \leftarrow \mathsf{Graph}^+.\mathsf{Enc}_{K_1}(G, \mathbf{m})$,
  2. compute $(\gamma^-, \mathbf{c}^-) \leftarrow \mathsf{Graph}^-.\mathsf{Enc}_{K_2}(G, \mathbf{m})$,
  3. for $1 \leq i \leq n$,
     (a) compute $\tau_i^+ \leftarrow \mathsf{Graph}^+.\mathsf{Token}_{K_1}(i)$,
     (b) compute $\tau_i^- \leftarrow \mathsf{Graph}^-.\mathsf{Token}_{K_2}(i)$,
  4. let L be the labeling generated from all the words in $\mathbf{m}$ (i.e., each m_i is labeled with the words it contains) and let $\mathbf{v} = \{(t_i^+, t_i^-)_i\}$,
  5. compute $(\gamma^\mathsf{I}, \mathbf{c}^\mathsf{I}) \leftarrow \mathsf{Label}.\mathsf{Enc}_{K_3}(L, \mathbf{M})$, where $\mathbf{M}$ is composed of $\mathbf{m}$ and $\mathbf{v}$,
  6. output $\gamma = (\gamma^+, \gamma^-, \gamma^\mathsf{I})$ and $\mathbf{c} = (\mathbf{c}^+, \mathbf{c}^-, \mathbf{c}^\mathsf{I})$.
- $\mathsf{Token}(K, w)$: output $\tau \leftarrow \mathsf{Label}.\mathsf{Token}_{K_3}(w)$.
- $\mathsf{Subgraph}_\mathsf{e}(\gamma, \tau)$:
  1. compute $(J^\mathsf{I}, \mathbf{v}_I) := \mathsf{Label}.\mathsf{Search}(\gamma^\mathsf{I}, \tau)$
  2. for all $j \in J^\mathsf{I}$,
     (a) compute $J_j^+ := \mathsf{Graph}^+.\mathsf{Neigh}(\gamma^+, \tau_j^+)$,
     (b) compute $J_j^- := \mathsf{Graph}^-.\mathsf{Neigh}(\gamma^-, \tau_j^-)$,
  3. output $J = \left(J^\mathsf{I}, \left(J_j^+, J_j^-\right)_{j \in J^\mathsf{I}}\right)$.
- $\mathsf{Dec}(K, c_j)$: return $m_j := \Pi.\mathsf{Dec}_{K_3}(c_j)$.

Fig. 5. A structured encryption scheme for web graphs supporting focused subgraph queries.

tacks, then the scheme described above is $(\mathcal{L}_1, \mathcal{L}_2)$*-secure against adaptive chosen-query attacks, where*

$$\mathcal{L}_1(G, L, \mathbf{m}) = (\mathcal{L}_1^I(L, \mathbf{m}), \mathcal{L}_1^+(G, \mathbf{m}), \mathcal{L}_2^-(G, \mathbf{m}))$$

and

$$\mathcal{L}_2(G, L, w) = \left(\mathcal{L}_2^I(L, w), \left(\mathcal{L}_2^+(G, i)\right)_{i \in |R(w)|}, \left(\mathcal{L}_2^-(G, i)\right)_{i \in |R(w)|}\right).$$

The proof is omitted due to lack of space but appears in [12].

7 Conclusions and Future Directions

Several interesting future directions are suggested by this work. The most immediate is whether efficient and non-interactive structured encryption can be

achieved while leaking less than the query and intersection pattern. The construction of efficient *dynamic* structured encryption schemes (i.e., that allow for updates to the encrypted data) is another direction left open by this work. Of course, the construction of schemes that handle other types of structured data and more complex queries on the data types considered here would also be interesting.

Acknowledgements

We are grateful to Kristin Lauter for encouragement during the early stages of this work, to Sherman Chow and Satya Lokam for useful discussions regarding graph encryption and to Susan Hohenberger for insisting on a thorough comparison with functional encryption. We are also grateful to Adam O'Neill for several helpful discussions on functional encryption. Finally, we thank Emily Shen and Charalampos Papamanthou for useful feedback on the manuscript and the anonymous reviewers for helpful suggestions.

References

1. M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. M. Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In *CRYPTO '05*, pp. 205–222.
2. J. Bardin, J. Callas, S. Chaput, P. Fusco, F. Gilbert, C. Hoff, D. Hurst, S. Kumaraswamy, L. Lynch, S. Matsumoto, B. O'Higgins, J. Pawluk, G. Reese, J. Reich, J. Ritter, J. Spivey, and J. Viega. Security guidance for critical areas of focus in cloud computing. Technical report, Cloud Security Alliance, April 2009.
3. M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *CRYPTO '07*, pp. 535–552.
4. J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symp. on Security and Privacy '07*, pp. 321–334.
5. D. Boneh, G. di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *EUROCRYPT '04*, pp. 506–522.
6. D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO '01*, pp. 213–229.
7. D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. Skeith. Public-key encryption that allows PIR queries. In *CRYPTO '07*, pp. 50–67.
8. D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC '07*, pp. 535–554.
9. X. Boyen and B. Waters. Anonymous hierarchical identity-based encryption (without random oracles). In *CRYPTO '06*, pp. 290–307.
10. M. Brautbar and M. Kearns. Local algorithms for finding intersting individuals in large networks. In *ICS '10*.
11. Y. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *ACNS '05* pp. 442–455.
12. M. Chase and S. Kamara. Structured encryption and controlled disclosure. IACR ePrint report, 2010. See `http://eprint.iacr.org`.

13. D. Chaum, C. Crépeau, and I. Damgard. Multiparty unconditionally secure protocols. In *STOC '88*, pp. 11–19.
14. Micrsoft Corp. Codename "Dallas". http://www.microsoft.com/windowsazure/dallas.
15. R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *ACM CCS '06*, pp. 79–88.
16. R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. Journal version (under submission), 2010.
17. C. Gentry. Fully homomorphic encryption using ideal lattices. In *ACM STOC '09*, pp. 169–178.
18. E-J. Goh. Secure indexes. Technical Report 2003/216, IACR ePrint Cryptography Archive, 2003. See `http://eprint.iacr.org/2003/216`.
19. O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In *ACM STOC '87*, pp. 218–229.
20. O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.
21. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.
22. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS '06*, pp. 89–98.
23. Infochimps. http://www.infochimps.org.
24. S. Kamara and K. Lauter. Cryptographic cloud storage. In *Workshop on on Real-Life Cryptographic Protocols and Standardization*, volume 6054 of *LNCS*, pages 136–149. Springer, 2010.
25. J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT '08*, pp. 146–162.
26. J. Kleinberg. Authoritative sources in a hyperlinked environment. In *SODA '98*, pp. 668–677.
27. R. Lempel and S. Moran. SALSA: The stochastic approach for link-structure analysis. *ACM Transactions on Information Systems*, 19(2):131–160, April 2001.
28. A. Sahai and B. Waters. Fuzzy identity-based encryption. In *EUROCRYPT '05*, pp. 457–473.
29. Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO '84*, pp. 47–53.
30. E. Shen, E. Shi, and B. Waters. Predicate privacy in encryption systems. In *TCC '09*, pp. 457–473.
31. E. Shi, J. Bethencourt, T. Chan, D. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *IEEE Symp. on Security and Privacy '07*, pp. 350–364.
32. C. Soghoian. Caught in the cloud: Privacy, encryption, and government back doors in the web 2.0 era. *Journal on Telecommunications and High Technology Law*, 8(2), 2010.
33. D. Song, D. Wagner, and A. Perrig. Practical techniques for searching on encrypted data. In *IEEE Symp. on Research in Security and Privacy '00*, pp. 44–55.
34. B. Waters. Functional encryption: An overview and survey slides. Presented at "Crypto in the Clouds Workshop" at MIT, 2009.
35. B. Waters, D. Balfanz, G. Durfee, and D. Smetters. Building an encrypted and searchable audit log. In *NDSS '04*.
36. A. Yao. Protocols for secure computations. In *FOCS '82*, pp. 160–164.