Restructuring the NSA Metadata Program

Seny Kamara

Microsoft Research
senyk@microsoft.com

Abstract. During the Summer of 2013, it was revealed through the documents leaked by Edward Snowden that the NSA was collecting the metadata of every US-to-foreign, foreign-to-US and US-to-US call from the largest US telephone providers. This led to public outcry and to President Obama calling for the restructuring of this program. The options initially considered included keeping the data at the providers, entrusting the data to a private entity, entrusting the data to a non-NSA government agency or ending the program all-together.
In this work, we show how cryptography can be used to design a privacy-preserving alternative to the NSA metadata program. We present a protocol based on structured encryption, in particular on graph encryption, and secure function evaluation that provides the following guarantees: (1) providers learn no information about NSA queries; (2) NSA queries can only be executed if validated by a given certification process; (3) the NSA learns nothing about the data beyond what can be inferred from the query results. In addition, these properties are achieved whether the data is stored at the providers, the NSA or on a third-party cloud.

1 Introduction

On June 5th, 2013, Glenn Greenwald published the first document from the Edward Snowden leaks in the Guardian [10]. This was a top secret court order compelling Verizon to hand the metadata of its calls to the National Security Agency (NSA) on a daily basis. This metadata was to include (among other things) the to and from numbers, the time and the duration of every foreign-to-US, US-to-foreign and US-to-US call. The revelation that the NSA was collecting information concerning *every* US citizen was astonishing to many and led to public outcry.

The Snowden revelations have motivated many important questions in a variety of disciplines including in Ethics, Law, Public Policy and Diplomacy. This work explores and formulates new problems in cryptography motivated by these disclosures. In particular, we consider the following question:

> *Can we design a practical privacy-preserving alternative to the NSA telephony metadata program?*

Answering this question will first require us to understand how the program works—as much as is possible from only public sources—and to formulate an appropriate notion of privacy for this setting. While we do propose a concrete cryptographic protocol for this problem, we stress that our proposal should be viewed as a first step and that *our main interest is in formulating and motivating further research in this direction*. It is also worth mentioning that by *practical*, we roughly mean: built out of efficient cryptographic primitives like symmetric-key encryption and hash functions and not from primitives that are currently mostly of theoretical interest like fully-homomorphic [8] or functional encryption [1].

To properly model the problem, we provide in Section 1.1 an overview of how the program works. Our understanding is based on various public sources including [5,4] but we note that this may not reflect exactly how the program works in practice and that there are likely many important aspects of it that have not been disclosed. To provide context for our work, we provide in the full version [13] a high-level survey of the legal questions surrounding the NSA metadata program. This is not needed to understand the protocols we present.

1.1 How the NSA Metadata Program Works

We provide an overview of how the metadata program works. Our understanding of the program relies mostly on the findings of President Obama's Review Group on Intelligence and Communications Technologies [4]. Each day, the telephone providers hand the metadata of every US-to-foreign, foreign-to-US and US-to-US call to the NSA. This metadata consists of the origin and destination numbers, the time and duration of the call, the international mobile subscriber identity (IMSI) number, the trunk identifier and telephone calling card numbers [4]. This data is stored by the NSA and each record has to be deleted after 5 years. The data can only be queried by a subset of 22 NSA analysts (two of which are supervisors) that have received special training. Furthermore, the dataset can only be queried by phone number and each query has to go through an internal NSA certification process. In particular, each query to the database has to be found to be relevant to a particular investigation by at least two analysts. If this is the case, the query has to be approved by at least one of the two supervisors and found to be associated with one of a set of FISA-court-approved Terrorist organizations. If the query passes this certification process, the analyst is allowed to query the database and receives the metadata associated with every number that called or was called from the query number and every number that was either called from or called any one of the latter numbers. Viewing the database as an undirected graph with phone numbers as vertices and edges between any two numbers for which there was a call, the analyst receives the metadata associated with any number that is at most 2 hops away from the query number. [1]

1.2 Our Approach

While the current NSA metadata program has an internal process for query certification and is claimed to include technological procedures to minimize the exposure of private information, [2] the program does not provide any *cryptographic* privacy guarantees. This simply follows from the fact that the telecommunications companies provide the metadata in plaintext to the NSA. At a high-level, our goal in designing a new system will therefore consist of having the providers

[1] Originally, the program allowed for 3-hop queries but this was reduced to 2 hops by the Obama Administration as of January 17th.

[2] Unfortunately, we could not find any details of how these mechanisms worked in public sources.

hand *encrypted* metadata to the NSA in such a way that analysts can then issue (cryptographically-enforced) certified queries on the encrypted data.

To achieve this we design a cryptographic protocol we refer to as MetaCrypt which relies in part on two important building blocks. The first is graph encryption (a special case of structured encryption) [3] which encrypts graphs in such a way that they can be privately queried. The second is secure function evaluation (SFE) which enables a set of parties to evaluate a function without revealing information about their inputs to each other [17,9]. We review both building blocks in Section 5. Our protocol makes a non-trivial use of these primitives and there are several technical difficulties to overcome in order to arrive at a final solution.

To analyze the security of our proposal, we isolate four properties we believe are crucial to any satisfactory solution:

1. isolation: the database should be protected from outsiders;
2. query privacy: the analyst queries should remain hidden from the providers and the server;
3. data privacy: the analyst should not learn any information about the database beyond what it can infer from the 2-hop queries it makes;
4. query certification: the analyst should only be able to make queries that satisfy the certification process described above (i.e., two analysts agree about the relevance of the query, at least one supervisor approves it and the associated organization is on a FISA-approved list of organizations).

In the full version of this work [13], we formalize these security properties in the ideal/real-world paradigm which is typically used to analyze the security of multi-party computation protocols. This paradigm has several advantages including modularity and simplicity.

Applications beyond the metadata program. Though the focus of this work is on the metadata program, the cryptographic techniques and protocols introduced have applications beyond this specific application. In particular, our main protocol can be used in any setting where a client wishes to privately query a set of privacy-sensitive graph datasets generated by various providers.

2 Related Work

As far as we know, the first discussion of privacy-preserving alternatives to the NSA programs appeared in a blog post of the author from July, 2013 [12]. There, a protocol is described that would enable an intelligence agency to privately query encrypted data generated by a provider in such a way that the query is certified by a third-party judge. The protocol makes use of MACs, secure two-party computation [17] and a keyword OT protocol of Freedman, Ishai, Pinkas and Reingold [7]. In [11], Jarecki, Jutla, Krawczyk, Rosu and Steiner describe a protocol similar in functionality to the one proposed in [12] but that, in addition, supports boolean keyword searches over the encrypted data. Concurrently with this work, Kroll, Felten and Boneh describe in [15] a set of protocols that allow an investigator to

privately retrieve the encrypted records of providers in such a way that investigator queries are certified by a judge. The protocols of [15] provide accountability but, unlike the solutions proposed in [12] and [11], do not support any form of search functionality over encrypted records (i.e., investigators can only access a record by an identifier).

We note that none of the protocols above are directly applicable to the problem considered in this work. This is simply because, as discussed in Section 1.1, the NSA metadata system is designed to support 2-hop neighbor queries on the *call graph* (i.e., the graph that underlies the providers' datasets) and such a functionality is not directly supported by these works. Presumably, 2-hop neighbor queries could be instantiated on top of these protocols by having the client perform an interactive breadth-first search, but this would require $O(d)$ rounds, where d is the degree of the vertex queried.

In this work, we provide a solution with a completely non-interactive query phase. More precisely, it only uses interaction to certify queries, not to execute the 2-hop queries over the encrypted datasets. We achieve this in part by making use of a 1-hop graph encryption scheme which, roughly speaking, allows one to encrypt a graph in such a way that it can be privately queried. Graph encryption was introduced by Chase and Kamara in [3], where constructions supporting various types of queries were proposed (adjacency queries, 1-hop neighbor queries and focused sub-graph queries). Graph encryption is a special case of structured encryption which encrypts arbitrarily-structured data in such a way that it can be privately queried [3]. To certify queries, we make use of secure multi-party computation as introduced by Yao for the two-party case [17] and by Goldreich, Micali and Wigderson for the n-party case [9].

In the 90's, a team at NSA led by Binney, Loomis and Wiebe designed a system called ThinThread for large-scale data analysis. The system was designed to provide some form of privacy protection for US citizens. Unfortunately, it was never deployed on a large scale and the only official document that discusses it is so heavily redacted that no information about its design can be gleaned [16].

3 Preliminaries and Notation

Graphs and graph databases. A graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ consists of a set of vertices V and a set of edges $\mathrm{E} \subseteq \mathrm{V} \times \mathrm{V}$. For any vertex $v \in \mathrm{V}$, we denote its d-hop neighbors by $\Gamma_d^{\mathrm{G}}(v)$ and its neighbors at a distance of at most d hops by $\Gamma_{\leq d}^{\mathrm{G}}(v)$. Given two graphs $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ and $\mathrm{G}' = (\mathrm{V}', \mathrm{E}')$ such that $\mathrm{E}' \subseteq (\mathrm{V} \cup \mathrm{V}') \times (\mathrm{V} \cup \mathrm{V}')$, we refer to the graph $\mathrm{G}'' = (\mathrm{V} \cup \mathrm{V}', \mathrm{E} \cup \mathrm{E}')$ as the sum of G and G' and to G' as an update to G. We sometime write $\mathrm{G}'' = \mathrm{G} + \mathrm{G}'$.

We view the metadata generated by a telecommunications provider as a directed graph, $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, where the vertices $v \in \mathrm{V}$ correspond to telephone numbers and where there is a directed edge from v_1 to v_2 if there is a call from the number associated with v_1 to the number associated with v_2. We associate to each *undirected* edge $e = \{v_1, v_2\}$ in E: (1) a unique identifier $\mathsf{id}(e)$ that is independent of the numbers/vertices in e; and (2) a document $\mathrm{D}_{\mathsf{id}(e)}$ that stores

information about calls between v_1 and v_2 such as time, originating number, destination number and duration. We refer to a graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ and its auxiliary documents $\mathbf{D} = \{\mathrm{D}_{\mathsf{id}(e)}\}_{e \in \mathrm{E}}$ as a graph database $\mathsf{GDB} = (\mathrm{G}, \mathbf{D})$. We denote the documents associated with edges 2 hops away from v as follows:

$$\mathsf{GDB}(v) = \left\{\mathrm{D}_{\mathsf{id}(v,w)} \in \mathbf{D} : w \in \Gamma_1^{\mathrm{G}}(v)\right\} \bigcup \left\{\mathrm{D}_{\mathsf{id}(w,z)} \in \mathbf{D} : z \in \Gamma_1^{\mathrm{G}}(w)\right\}_{w \in \Gamma_1^{\mathrm{G}}(v)}.$$

Parties and adversarial structures. The participants in our protocol include t providers $(\mathrm{Prv}_1, \ldots, \mathrm{Prv}_t)$ that generate the metadata; a server Srv that stores the (encrypted) metadata; two analysts An_1 and An_2 which query the metadata; two supervisors Sup_1 and Sup_2 that validate queries; and a FISA judge J that provides a watch list WL of organizations. The analysts and supervisors are assumed to belong to a single *agency*. In our security analysis, we will consider the cases where the server is managed by the providers and where the server is managed by the agency. We assume private and authenticated channels between all parties.

4 The MetaDB Functionality

As mentioned in Section 1, we use the ideal/real-world paradigm to analyze the security of our protocol. Here, we give an overview of the ideal functionality that captures the security properties we want (a detailed security definition is provided in the full version). The functionality, which we refer to as the MetaDB functionality, supports the operations of the NSA metadata program as described in Section 1.1, but with privacy guarantees for the analyst queries and the graph databases, and with a cryptographically-enforced query certification process.

The functionality is executed between t providers $(\mathrm{Prv}_1, \ldots, \mathrm{Prv}_t)$, a server Srv, two analysts An_1 and An_2, two supervisors Sup_1 and Sup_2 and a judge J. It is parameterized by three leakage functions $\mathcal{L}_{\mathsf{S}}$, $\mathcal{L}_{\mathsf{N}}$ and $\mathcal{L}_{\mathsf{U}}$ and is a $(t+6)$-party reactive functionality. Throughout, we will assume that the first analyst An_1 is primarily interested in making the query and that the purpose of the second analyst An_2 is to provide support for the query (i.e., to satisfy the constraint that at least two analysts must determine that the query is relevant to the investigation). This is without loss of generality since the roles can be inversed.

In the first phase, each provider Prv_i sends its graph database $\mathsf{GDB}_i = (\mathrm{V}_i, \mathrm{E}_i, \mathbf{D}_i)$ to the trusted party while the judge J sends a watch-list WL. At the end of this phase, the functionality sends leakage $(\mathcal{L}_{\mathsf{S}}(\mathsf{GDB}_1), \ldots, \mathcal{L}_{\mathsf{S}}(\mathsf{GDB}_t))$ to the server Srv. In the next phases the parties either update the data or query it. To query the data, the parties do the following. The two analysts send their query vertices v and v' to the functionality and the two supervisors Sup_1 and Sup_2 send tuples $(v_1, b_1, \mathrm{org}_1)$ and $(v_2, b_2, \mathrm{org}_2)$, respectively. Here, v_1 and v_2 are the vertices under consideration, b_1 and b_2 are bits indicating whether the respective vertices are authorized, and org_1 and org_2 are the organizations associated with the vertex. If $v = v'$ and if at least one of the supervisors' inputs has the form $(v, 1, \mathrm{org})$, the functionality checks that $\mathrm{org} \in \mathrm{WL}$. If this is the case, it returns the documents $\bigcup_{i=1}^{t} \mathsf{GDB}_i(v)$ to analyst An_1. It also sends leakage $(\mathcal{L}_{\mathsf{N}}(\mathsf{GDB}_1, v), \ldots, \mathcal{L}_{\mathsf{N}}(\mathsf{GDB}_t, v))$ to the server Srv. To update a graph database,

each provider Prv_i sends a tuple $\mathsf{up}_i = (\mathrm{V}_i^+, \mathrm{E}_i^+, \mathbf{D}_i^+)$, where V_i^+ is either a set of new vertices or $\perp$, E_i^+ is a set of new edges in $(\mathrm{V}_i \times \mathrm{V}_i^+) \cup (\mathrm{V}_i \times \mathrm{V}_i^+)$ and $\mathbf{D}_i^+ = \{\mathrm{D}_{\mathsf{id}(e^+)}\}_{e^+ \in \mathrm{E}_i^+}$ is a set of new documents. The functionality then sends leakage $(\mathcal{L}_{\mathsf{U}}(\mathsf{GDB}_1, \mathsf{up}_1), \ldots, \mathcal{L}_{\mathsf{U}}(\mathsf{GDB}_t, \mathsf{up}_t))$ to the server.

Other certification processes. We briefly note that while the MetaDB functionality captures a very specific certification process—essentially the one described in [4]—any new or different process could be easily formalized by changing or extending the functionality described above. Since our concrete protocol relies in part on (general-purpose) secure function evaluation for query certification, it could also be extended to capture a new/different certification process.

5 Cryptographic Building Blocks

We review the building blocks used in the MetaCrypt protocol. These include SFE [17,9] and graph encryption [3]. An SFE protocol allows n parties to evaluate a function f on their private inputs $\mathbf{x} = (x_1, \ldots, x_n)$ in such a way they cannot learn any information about each other's inputs beyond what can be inferred from the result. A graph encryption scheme encrypts a graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ in such that way that the graph structure (i.e., the edges E) is hidden and that it can be queried without disclosing the query. We describe each of these in more detail.

Secure function evaluation. An SFE protocol securely computes any polytime computable function $f : X_1 \times \cdots \times X_n \to Y_1 \times \cdots \times Y_n$. The protocol is executed between n parties $(P_1, \ldots, P_n)$, where the ith party holds input x_i and receives output $y_i = f_i(x_1, \ldots, x_n)$. An MPC protocol is a protocol that securely computes any polynomial-time reactive functionality $\mathcal{F} = (f_1, \ldots, f_\ell)$.

5.1 Graph Encryption

Graph encryption was introduced in [3] as a special case of structured encryption. A graph encryption scheme takes a graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ and produces an encrypted graph EGR that hides the structure of the graph. [3] The encrypted graph can then be queried using tokens that can only be generated with knowledge of a secret key. In [3], several constructions are proposed that support various kinds of queries, including adjacency queries (i.e., given two vertices, do they share an edge?) and 1-hop neighbor queries (i.e., given a vertex v, return all the vertices that share an edge with v). In this work, we require a scheme that supports 1-hop neighbor queries but that, in addition, is *associative*, *dynamic* and is *edge-centric*. We augment the syntax and security definitions of [3] to capture such a scheme.

Definition 1 (Dynamic graph encryption with 1-hop neighbor queries). *A dynamic and associative graph encryption scheme that supports 1-hop neighbor queries* $\mathsf{Graph} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Nghbr}, \mathsf{Token}^+, \mathsf{Add})$ *consists of five polynomial-time algorithms that work as follows:*

[3] Typically, the number of vertices is revealed but this can be hidden using padding.

– $(K, \mathsf{EGR}) \leftarrow \mathsf{Setup}(1^k, \mathrm{G}, \mathsf{sp})$*: is a probabilistic algorithm that takes as input a security parameter* k*, a graph* $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ *and semi-private information* $\mathsf{sp} = \big(e, s_e\big)_{e \in \mathrm{E}}$*. It outputs a secret key* K *and an encrypted graph* EGR*.*

– $\mathsf{tk} := \mathsf{Token}(K, v)$*: is a deterministic algorithm that takes as input a secret key* K *and a vertex* $v \in \mathrm{V}$ *and outputs a token* tk*.*

– $\{(\mathsf{id}, s_{\mathsf{id}})\}_{\mathsf{id} \in I} := \mathsf{Nghbr}(\mathsf{EGR}, \mathsf{tk})$*: is a deterministic algorithm that takes as input an encrypted graph* EGR *and a token* tk *and returns a set of id/string pairs* $\{(\mathsf{id}, s_{\mathsf{id}})\}_{\mathsf{id} \in I}$*, where* $I \subseteq \{\mathsf{id}(e)\}_{e \in \mathrm{E}}$*.*

– $\mathsf{atk} := \mathsf{Token}^+(K, \mathrm{G}^+, \mathsf{sp}^+)$*: is a deterministic algorithm that takes as input a secret key* K*, a graph update* G^+ *and semi-private information* sp^+ *and returns an add token* atk*.*

– $\mathsf{EGR}' := \mathsf{Add}(\mathsf{EGR}, \mathsf{atk})$*: is a deterministic algorithm that takes as input an encrypted graph* EGR *and a token* atk *and outputs an encrypted graph* EGR'*.*

A note on deletion. Recall that the metadata program requires the NSA to remove from its database all information associated with calls older than 5 years. Note, however, that the formulation of graph encryption given in Definition 1 does not support deletions. The reason is essentially that since the deletion of the (encrypted) documents cannot be enforced (e.g., the server holding the documents can always make copies) there is no security-related reason for the encrypted graphs to support deletion. The value of supporting deletion would mostly be efficiency (e.g., to avoid returning old documents) but that can be handled using non-cryptographic mechanisms (e.g., not returning any encrypted document that was received past a certain date).

Security. Intuitively, a graph encryption scheme is secure if, given an encrypted graph EGR and a token tk, the adversary cannot learn anything about the underlying graph and query. This exact intuition is difficult to achieve (efficiently) so the security guarantee is usually weakened to allow for some form of leakage. The leakage is formalized by parameterizing the security definition with a set of leakage functions $\mathcal{L}_{\mathsf{S}}$, $\mathcal{L}_{\mathsf{N}}$ and $\mathcal{L}_{\mathsf{U}}$ which precisely capture the leakage of the scheme's Setup, Nghbr and Update algorithms, respectively. We recall in Definition 2 below the notion of adaptive semantic security for graph encryption which is a special case of the definition from [3] which itself generalizes the definition from [6].

Definition 2 (Adaptive semantic security [6,3]). *Let* $\mathsf{Graph} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Nghbr}, \mathsf{Token}^+, \mathsf{Add})$ *be a dynamic and associative graph encryption scheme supporting* 1*-hop neighbor queries and consider the following probabilistic experiments where* $\mathcal{A}$ *is an adversary,* $\mathcal{S}$ *is a simulator and* $\mathcal{L}_{\mathsf{S}}$*,* $\mathcal{L}_{\mathsf{N}}$ *and* $\mathcal{L}_{\mathsf{U}}$ *are (stateful) leakage algorithms:*

Real$_{\mathsf{Graph},\mathcal{A}}(k)$*: the adversary* $\mathcal{A}$ *generates a graph* $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ *and semi-private information* sp *from which the challenger creates an encrypted graph* EGR*, where* $(K, \mathsf{EGR}) \leftarrow \mathsf{Setup}(1^k, \mathrm{G}, \mathsf{sp})$*. Given* EGR*, the adversary* $\mathcal{A}$ *makes a polynomial number of adaptive queries and updates. For each neighbor query* v*,* $\mathcal{A}$ *receives a token* $\mathsf{tk} := \mathsf{Token}_K(v)$ *from the challenger and for each*

graph update G^+ *and semi-private information* sp^+ *it receives an add token* $\mathsf{atk} := \mathsf{Token}^+(K, \mathrm{G}^+, \mathsf{sp}^+)$. *Finally,* $\mathcal{A}$ *returns a bit* b *that is output by the experiment.*

$\mathbf{Ideal}_{\mathsf{Graph},\mathcal{A},\mathcal{S}}(k)$*: the adversary* $\mathcal{A}$ *outputs a graph* $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ *and semi-private information* $\mathsf{sp} = \big(e, s_e\big)_{e \in \mathrm{E}}$. *Given leakage* $\mathcal{L}_{\mathsf{S}}(\mathrm{G}, \mathsf{sp})$*, the simulator* $\mathcal{S}$ *returns an encrypted graph* EGR. *The adversary then makes a polynomial number of adaptive queries and updates. For each query* v *the simulator is given* $\mathcal{L}_{\mathsf{N}}(\mathrm{G}, v)$ *and* $\big\{s_{(v,w)}\big\}_{w \in \Gamma_1^{\mathrm{G}}(v)}$ *and returns a token* tk *to* $\mathcal{A}$. *For each graph update* G^+ *and new semi-private information* sp^+*, the simulator receives* $\mathcal{L}_{\mathsf{U}}(\mathrm{G}, \mathrm{G}^+, \mathsf{sp}^+)$ *and returns an add token* atk *to* $\mathcal{A}$. *Finally,* $\mathcal{A}$ *returns a bit* b *that is output by the experiment.*

We say that Graph *is adaptively* $(\mathcal{L}_{\mathsf{S}}, \mathcal{L}_{\mathsf{N}}, \mathcal{L}_{\mathsf{U}})$*-secure if for all* PPT *adversaries* $\mathcal{A}$*, there exists a* PPT *simulator* $\mathcal{S}$ *such that*

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

Instantiating 1-hop graph encryption. In [3], Chase and Kamara show how to construct a static, non-interactive, associative 1-hop graph encryption scheme from any static, non-interactive, associative, and *chainable* searchable symmetric encryption (SSE) scheme. Roughly speaking, chainability means that the scheme's $\mathcal{L}_{\mathsf{S}}$ leakage does not reveal any information about the semi-private information. For a discussion and formalization see the full version of [3]. Non-interactive means search requires only a single message from the client.

The high-level idea of the CK transformation is as follows: document identifiers are set to the vertex labels and a vertex label v' is added to a document with identifier v if the graph has either a (directed) edge (v, v') or (v', v). A 1-hop undirected neighbor query for vertex v then consists of searching for keyword v. It is not hard to see that this transformation can be extended to the dynamic case as well if the underlying SSE scheme supports both add operations (i.e., adding documents) and edits (i.e., adding words to an existing document).

Unfortunately, we cannot use the Chase-Kamara transformation here. The reason is that a direct application of it yields a "vertex-centric" 1-hop graph encryption scheme in the sense that the semi-private strings are associated to *vertices* and that the Nghbr algorithm returns *vertex* identifiers. This is in contrast to the kind of scheme we need and that is described above which is "edge-centric" in the sense that the semi-private strings are associated with *edges* and that the Nghbr algorithm returns *edge* identifiers. Nevertheless, in the full version of this work [13], we show how to construct such an edge-centric 1-hop graph encryption scheme based on the dynamic SSE schemes of Kamara, Papamanthou and Roeder [14] and of Cash *et al.*[2] (in particular, based on the scheme Π^+_{bas}).

6 The MetaCrypt Protocol

In this Section, we describe our main protocol, MetaCrypt, which securely computes the MetaDB functionality. The protocol is described in detail in the full version of this work [13] and, at a high level, works as follows. It makes use of an SFE

protocol Π, a graph encryption scheme $\mathsf{Graph} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Nghbr}, \mathsf{Token}^+, \mathsf{Add})$ that supports 1-hop neighbor queries, a public-key encryption scheme $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, a pseudo-random function F and a random oracle (RO) H. The RO can be removed at the cost of increased storage complexity. We assume private and authenticated channels between all parties which can be instantiated using standard cryptographic primitives. The protocol supports three operations: setup, queries and updates, which are described next.

Setup. During setup, the agency generates a public/private key pair $(\mathrm{pk}, \mathrm{sk})$. The secret key is sent to all its analysts (we discuss in the full version how to augment the protocol to support individual analyst keys) and the public key is sent to the providers $(\mathrm{Prv}_1, \ldots, \mathrm{Prv}_t)$. The t providers encrypt their graph databases $\mathsf{GDB}_1, \ldots, \mathsf{GDB}_t$ and send the result to the server. This encryption step, however, does not consist of simply applying the underlying graph encryption scheme as there are three main difficulties to overcome. The first is that in our setting—unlike in the standard structured/searchable encryption setting—the intended recipient of the data (the analyst) is not the owner of the data (the provider). The second difficulty is that the graph encryption schemes we have only support 1-hop neighbor queries, whereas we need to handle 2 hops. A third, and more subtle, issue has to do with how the documents are encrypted. In fact, unlike the standard client/server setting where structured/searchable encryption is typically applied, in our setting we cannot use any CPA-secure symmetric encryption scheme to encrypt the documents. The difficulty is that in the adversarial structures we consider, the adversary not only corrupts the server but the analyst as well which means the adversary will have access to the *decrypted* documents that are relevant to the queries. To satisfy our adaptive and simulation-based definition where the ideal adversary learns the contents of the documents only *after* having committed to simulated ciphertexts, the underlying encryption scheme has to be non-committing.

We handle the first issue using hybrid encryption and the third using the "standard" PRF-based symmetric-key encryption scheme—though we replace the PRF with a RO for efficiency reasons. Recall that each graph database GDB consists of a graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ and a set of documents $\mathbf{D}$. Each provider encrypts its documents by computing, for all $e \in \mathrm{E}$, $c_{\mathsf{id}(e)} := \langle \mathrm{D}_{\mathsf{id}(e)} \oplus H(K_{\mathsf{id}(e)}, r_{\mathsf{id}(e)}, r_{\mathsf{id}(e)}) \rangle$, where $K_{\mathsf{id}(e)} := F_{K^h}(\mathsf{id}(e))$, $K^h \xleftarrow{\$} \{0,1\}^k$ and $r_{\mathsf{id}(e)} \xleftarrow{\$} \{0,1\}^k$. Each key $K_{\mathsf{id}(e)}$ is then encrypted under the agency public key pk. We refer to these public-key-encrypted keys as *key encapsulations* and denote them $\mathsf{ke}_{\mathsf{id}(e)}$. The encapsulations are stored as semi-private information in encrypted graphs constructed from G. This is done so that the result of a query includes both the identifiers and the encapsulations of the relevant encrypted documents. After receiving the encrypted documents and encapsulations from the server, the analyst uses the agency secret key sk to recover the symmetric keys with which it decrypts the documents.

To handle 2-hop neighbor queries based on a scheme that only supports 1-hop queries, we use the chaining approach first used in [3] to construct web graph encryption schemes for focused subgraph queries. The high-level idea is to encrypt the graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ twice and to store tokens for the second encryption as

semi-private information in the first encryption. More specifically, we generate a second-level encrypted graph by computing $(K^{(2)}, \mathsf{EGR}^{(2)}) \leftarrow \mathsf{Setup}(1^k, \mathrm{G}, \mathsf{sp}^{(2)})$, where $\mathsf{sp}^{(2)}$ is the semi-private information $\big(e, \mathsf{ke}_{\mathsf{id}(e)}\big)_{e\in \mathrm{E}}$. We then generate tokens for all vertices $v \in \mathrm{V}$ by computing $\mathsf{tk}_v^{(2)} \leftarrow \mathsf{Token}_{K^{(2)}}(v)$ and create a first-level encrypted graph by computing $(K^{(1)}, \mathsf{EGR}^{(1)}) \leftarrow \mathsf{Setup}(1^k, \mathrm{G}, \mathsf{sp}^{(1)})$, where $\mathsf{sp}^{(1)}$ is the semi-private information $\big(e, \langle \mathsf{tk}_{e_2}^{(2)}, \mathsf{ke}_{\mathsf{id}(e)}\rangle\big)_{e\in \mathrm{E}}$, where e_2 is the terminating vertex of e. Finally, the provider sets its key to $K = (K^h, K^{(1)}, K^{(2)})$ and sends an encrypted graph database $\mathsf{EGDB} = \big(\mathsf{EGR}^{(1)}, \mathsf{EGR}^{(2)}, \{c_{\mathsf{id}(e)}\}_{e\in \mathrm{E}}\big)$ to the server.

Updates. To update an encrypted graph database with a new graph $\mathrm{G}^+ = (\mathrm{V}^+, \mathrm{E}^+)$ and new documents $\mathbf{D}^+$, a provider does the following. It first encrypts the documents as in the Setup phase: it generates a key $K_{\mathsf{id}(e^+)} := F_{K^h}(\mathsf{id}(e^+))$ for each new edge $e^+ \in \mathrm{E}^+$; creates a key encapsulation $\mathsf{ke}_{\mathsf{id}(e^+)} \leftarrow \mathsf{PKE.Enc}_{\mathrm{pk}}(K_{\mathsf{id}(e^+)})$; and encrypts the document by computing $c_{\mathsf{id}(e^+)} := \langle \mathrm{D}_{\mathsf{id}(e^+)} \oplus H(K_{\mathsf{id}(e^+)}, r_{\mathsf{id}(e^+)}), r_{\mathsf{id}(e^+)}\rangle$, where $r_{\mathsf{id}(e^+)} \xleftarrow{\$} \{0,1\}^k$. It then stores the key encapsulations as semi-private information in an update to the second-level encrypted graph. More precisely, it generates an add token $\mathsf{atk}^{(2)} := \mathsf{Token}^+_{K^{(2)}}(\mathrm{G}^+, \mathsf{sp}^{(2)})$, where $\mathsf{sp}^{(2)} = (e^+, \mathsf{ke}_{\mathsf{id}(e^+)})_{e^+\in \mathrm{E}^+}$. It then generates second-level query tokens for every vertex in G^+ by computing, for all $v^+ \in \mathrm{V}^+$, $\mathsf{tk}_{v^+}^{(2)} := \mathsf{Token}_{K^{(2)}}(v^+)$. These second-level query tokens are then stored as semi-private information in an update to the first-level encrypted graph. Specifically, the provider computes $\mathsf{atk}^{(1)} := \mathsf{Token}^+_{K^{(1)}}(\mathrm{G}^+, \mathsf{sp}^{(1)})$, where $\mathsf{sp}^{(1)} = \big(e^+, \langle \mathsf{tk}_{e_2^+}^{(2)}, \mathsf{ke}_{\mathsf{id}(e^+)}\rangle\big)_{e^+\in \mathrm{E}^+}$. The provider then sends an update $(\mathsf{atk}^{(1)}, \mathsf{atk}^{(2)}, \{c_{\mathsf{id}(e^+)}\}_{e^+\in \mathrm{E}^+})$ to the server who uses the add tokens to update the encrypted graphs and stores the new ciphertexts. If a new document encryption $c_{\mathsf{id}(e^+)}$ is for an edge for which there already exists ciphertexts (perhaps the new document contains metdata on new calls conducted between the vertices), then the server just concatenates the new ciphertext to the olds ones.

Queries. During the query phase, the parties interact in such a way that the analyst An_1 receives a token for his vertex if the latter is certified with respect to the policy outlined in Section 1. This phase mainly consists of the execution of an SFE protocol Π between the providers $(\mathrm{Prv}_1, \ldots, \mathrm{Prv}_t)$ who input the keys $(K_1^{(1)}, \ldots, K_t^{(1)})$, the analysts An_1 and An_2 who input their query vertices v and v', the supervisors Sup_1 and Sup_2 who input tuples $(v_1, b_1, \mathrm{org}_1)$ and $(v_2, b_2, \mathrm{org}_2)$ and the judge J who inputs the watch list WL.

The function f that is evaluated is defined as follows. First, it checks whether the query vertices v and v' of the analysts are equal. If so, it verifies that at least one supervisor authorizes the query by verifying that either $b_1 = 1$ or $b_2 = 1$. In the following suppose, without loss of generality, that $b_1 = 1$, i.e., the first supervisor Sup_1 approved the query. The function checks that the vertex v_1 approved by Sup_1 is indeed the same as the vertex input by the analysts. This is to avoid a potential attack where an analyst, say An_1, asks a supervisor, say Sup_1, to approve a query vertex v_1 but inputs a vertex $v \neq v_1$ into the SFE protocol. If this is the case, the function checks that the organization org_1 submitted by Sup_1 is on the watch list submitted by the judge. If this is the case, the function uses

the keys $(K_1^{(1)}, \ldots, K_t^{(1)})$ to generate query tokens $(\mathsf{tk}_1^{(1)}, \ldots, \mathsf{tk}_t^{(1)})$ for vertex v. The function returns these tokens to the analyst.

The analyst then sends the tokens to the server who uses them to query the providers' encrypted graph databases $(\mathsf{EGDB}_1, \ldots, \mathsf{EGDB}_t)$. More precisely, for each encrypted database $\mathsf{EGDB}_i = (\mathsf{EGR}_i^{(1)}, \mathsf{EGR}_i^{(2)}, \{c_{\mathsf{id}(e)}\}_{e \in \mathrm{E}})$ the server does the following. It queries the first-level encrypted graph by computing $\mathsf{Nghbr}(\mathsf{EGR}_i^{(1)}, \mathsf{tk}_i^{(1)})$. This results in either $\perp$ or a set of tuples $\left(\mathsf{id}(v,w), \left\langle \mathsf{tk}_w^{(2)}, \mathsf{ke}_{\mathsf{id}(v,w)} \right\rangle\right)_{w \in \Gamma_1^{\mathrm{G}}(v)}$, consisting of an edge identifier $\mathsf{id}(v,w)$, a second-level token $\mathsf{tk}_w^{(2)}$ and a key encapsulation $\mathsf{ke}_{\mathsf{id}(v,w)}$. For each $w \in \Gamma_1^{\mathrm{G}}(v)$, the server uses the second-level token to query the second-level encrypted graph $\mathsf{EGR}_i^{(2)}$, which results in tuples $\left(\mathsf{id}(w,z), \mathsf{ke}_{\mathsf{id}(w,z)}\right)_{z \in \Gamma_1^{\mathrm{G}}(w)}$, consisting of an edge identifier $\mathsf{id}(w,z)$ and a key encapsulation $\mathsf{ke}_{\mathsf{id}(w,z)}$. The server then returns the encryptions and key encapsulations of all the edges recovered.

Security of the MetaCrypt protocol. To analyze the security of our protocol, we show in the full version that it securely computes the MetaDB functionality. Our analysis, however, is slightly different and less general than what is typically found in the literature. In particular, we are not interested in threshold adversarial structures since, in our setting, each party plays a very distinct role and since, in practice, we are concerned with very specific threats. Specifically, the two main adversarial structures that concern us are: (1) when the adversary corrupts the server Srv, the analysts An_1 and An_2, the supervisors Sup_1 and Sup_2 and the judge J; and (2) when the adversary corrupts the server Srv and the providers Prv_1 through Prv_t. The first structure captures the setting where the agency-affiliated parties (plus the judge) are corrupted and collude. Showing that our protocol is secure under this structure essentially lets us analyze the security afforded to providers—and by extension to the users whose metadata is included in the datasets—when the Government acts dishonestly. The second structure captures the setting where the providers are corrupted and colluding. Showing that our protocol is secure under this structure lets us reason about the security afforded to the agency when the providers act dishonestly. Notice that in both cases, we include the server in the adversarial structure. This effectively guarantees that the security of the protocol still holds no matter who manages the server. In the full version, we show the security of our protocol in the presence of a semi-honest adversary against these adversarial structures and discuss how to achieve malicious security.

References

1. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. Theory of Cryptography Conference (TCC '11), 2011.
2. D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *Network and Distributed System Security Symposium (NDSS '14)*, 2014.

3. M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology - ASIACRYPT '10*, 2010.
4. R. Clarke, M. Morell, G. Stone, C. Sunstein, and P. Swire. Liberty and security in a changing world. See `http://www.whitehouse.gov/sites/default/files/docs/2013-12-12_rg_final_report.pdf`, 2013.
5. United States Foreign Intelligence Surveillance Court. Primary order. See `http://www.dni.gov/files/documents/PrimaryOrder_Collection_215.pdf`, April 2013.
6. R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *ACM Conference on Computer and Communications Security (CCS '06)*, pages 79–88. ACM, 2006.
7. M. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography Conference (TCC '05)*. Springer, 2005.
8. C. Gentry. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing (STOC '09)*, pages 169–178. ACM Press, 2009.
9. O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In *ACM Symposium on the Theory of Computation (STOC '87)*, pages 218–229. 1987.
10. G. Greenwald. NSA collecting phone records of millions of verizon customers daily, July 2013. See `http://www.theguardian.com/world/2013/jun/06/nsa-phone-records-verizon-court-order`.
11. S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Outsourced symmetric private information retrieval. In *ACM Conference on Computer and Communications Security (CCS '13)*, pages 875–888, 2013.
12. S. Kamara. Are compliance and privacy always at odds? See `http://outsourcedbits.org/2013/07/23/are-compliance-and-privacy-always-at-odds/`, July 2013.
13. S. Kamara. Restructuring the NSA metadata program (full version). See `http://research.microsoft.com/en-us/projects/metacrypt/`, April 2014.
14. S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *ACM Conference on Computer and Communications Security (CCS '12)*. ACM Press, 2012.
15. J. Kroll, E. Felten, and D. Boneh. Secure protocols for accountable warrant execution. See `http://www.cs.princeton.edu/~felten/warrant-paper.pdf`, April 2014.
16. Office of the Inspector General of the Department of Defense. Requirements for the trailblazer and thinthread systems. See `https://www.fas.org/irp/agency/dod/ig-thinthread.pdf`, 2004.
17. A. Yao. Theory and application of trapdoor functions. In *IEEE Symposium on Foundations of Computer Science (FOCS '82)*, pages 80–91. IEEE Computer Society, 1982.