GOOSE: A Secure Framework for Graph Outsourcing and SPARQL Evaluation

Radu Ciucanu[1(✉)] and Pascal Lafourcade[2]

[1] INSA Centre Val de Loire, Univ. Orléans, LIFO EA 4022, Orléans, France
radu.ciucanu@insa-cvl.fr

[2] Université Clermont Auvergne, LIMOS CNRS UMR 6158, Clermont-Ferrand, France
pascal.lafourcade@uca.fr

Abstract. We address the security concerns that occur when outsourcing graph data and query evaluation to an *honest-but-curious* cloud i.e., that executes tasks dutifully, but tries to gain as much information as possible. We present GOOSE, a secure framework for Graph OutsOurcing and SPARQL Evaluation. GOOSE relies on cryptographic schemes and secure multi-party computation to achieve desirable security properties: (i) no cloud node can learn the graph, (ii) no cloud node can learn at the same time the query and the query answers, and (iii) an external network observer cannot learn the graph, the query, or the query answers. As query language, GOOSE supports *Unions of Conjunctions of Regular Path Queries* (UCRPQ) that are at the core of the W3C's SPARQL 1.1, including recursive queries. We show that the overhead due to cryptographic schemes is linear in the input's and output's size. We empirically show the scalability of GOOSE via a large-scale experimental study.

Keywords: Unions of Conjunctions of Regular Path Queries · Secure SPARQL evaluation · Secure graph outsourcing · Honest-but-curious cloud

1 Introduction

Outsourcing data and computations to a public cloud gained increasing popularity over the last years. Many cloud providers offer an important amount of data storage and computation power at a reasonable price e.g., Google Cloud Platform, Amazon Web Services, Microsoft Azure. However, cloud providers do not usually address the fundamental problem of protecting data security. The outsourced data can be communicated over some network and processed on some machines where malicious cloud admins could learn and leak sensitive data. We address the data security issues that occur when outsourcing an RDF graph database to a public cloud and querying the outsourced graph with SPARQL.

We depict the considered scenario in Fig. 1, where a *data owner* outsources a graph to the cloud, then a *user* is allowed to query the graph by submitting

© IFIP International Federation for Information Processing 2020
Published by Springer Nature Switzerland AG 2020

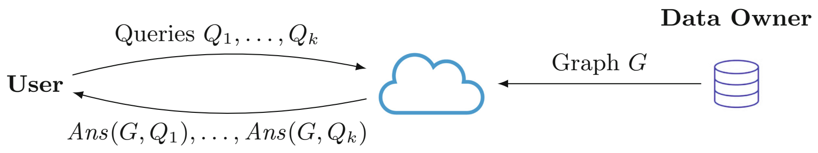

Data Owner
Queries $Q_1, \ldots, Q_k$
User
Graph G
$Ans(G, Q_1), \ldots, Ans(G, Q_k)$

Fig. 1. Outsourcing data and computations.

queries to the cloud, which computes and returns the queries' answers to the user. Our scenario is inspired by the *database as a service* cloud computing service model, which usually considers relational databases, and security is well-known as a major concern [8]: "*A significant barrier to deploying databases in the cloud is the perceived lack of privacy, which in turn reduces the degree of trust users are willing to place in the system.*" A typical solution to this concern (developed in systems such as CryptDB [18]) is to outsource encrypted data and use SQL-aware encryption schemes to answer queries directly on encrypted data.

Although SQL and SPARQL share some common functionalities, adapting a system such as CryptDB to securely answer SPARQL queries on outsourced graphs is not trivial because SPARQL allows to naturally express classes of queries that are cumbersome to express in SQL. This is the case for the recursive queries, which can be easily expressed using the Kleene star in the property paths of SPARQL 1.1[1]. To express such recursive queries in SQL, one needs to define recursive views. After analyzing the source code of the SQL parser inside CryptDB[2], we concluded that such queries are beyond the scope of CryptDB and it is unclear how hard it is to extend their system to support recursive queries.

We propose GOOSE, a framework for Graph OutsOurcing and SPARQL Evaluation, which allows the data owner to securely outsource to the cloud a graph that can be then queried by the user. We assume that the cloud is *honest-but-curious* i.e., executes tasks dutifully, but tries to gain as much information as possible. Similarly to CryptDB, GOOSE evaluates queries on encrypted data without any change to the query engine, which in our case is the standard Apache Jena for evaluating SPARQL queries. As query language, GOOSE supports *Unions of Conjunctions of Regular Path Queries* (UCRPQ) that are at the core of the W3C's SPARQL 1.1, including recursive queries via the Kleene star.

The key ingredients of GOOSE are: (i) *secure multi-party computation* i.e., the graph storage is distributed among 3 cloud participants, which can jointly compute the query answers for each submitted query, but none of the cloud participants can learn the graph, and none of the cloud participants can learn at the same time the query and the answers of the query on the graph, and (ii) *cryptographic schemes* i.e., all messages exchanged between GOOSE participants are encrypted with AES-CBC [1,5] such that an external network observer cannot learn the graph, the query, or the answers of the query on the graph.

[1] https://www.w3.org/TR/sparql11-property-paths/.
[2] https://css.csail.mit.edu/cryptdb/#Software.

Related Work. GOOSE follows a recent line of research on tackling the security concerns related to RDF graph data storage and querying [10–12,14–16].

The state-of-the-art system for query evaluation on encrypted graphs is HDT$_{crypt}$ [11], which focuses on (non-recursive) SPARQL queries defined as triple patterns. HDT$_{crypt}$ combines HDT (a compression technique useful for reducing RDF storage space) and encryption (to hide particular subgraphs from unauthorized users). Our work is complementary to this related research direction since we assume that query evaluation is outsourced to the cloud and our security goals are different from theirs: we want to avoid that the cloud nodes and network observer learn the entire graph, queries or query answers, whereas their goal is to allow multiple users with different access rights to query the graph. A common idea between HDT compression and our GOOSE is to map nodes and edge labels to integers, but for different goals. For HDT$_{crypt}$, the goal is to reduce storage and bandwidth usage. For GOOSE, the combination of this technique with secure multi-party computation is particularly useful for achieving security since the actual mapping functions are not shared with the node responsible for query evaluation, which is able to evaluate UCRPQ without knowing which are the true nodes and edge labels that it manipulates.

If one chooses to store RDF graphs in a relational database and query them with SQL, then one can choose CryptDB [18] to run queries directly in the encrypted domain. As already mentioned, CryptDB does not currently support recursive queries, and such queries are anywise cumbersome to express in SQL as they require recursive views. CryptDB has been extended as CryptGraphDB [2] to run Neo4j queries on encrypted graphs, but again without considering recursive queries. GOOSE is complementary to these systems since our goal is to propose a system that is able to run UCRPQ while enjoying similar security properties. We choose to rely on UCRPQ because this class of queries is at the core of the W3C's SPARQL 1.1 property paths, including recursive queries via the Kleene star. A recent large-scale analytical study of SPARQL query logs [7] includes more than a million such recursive queries, which suggests that a secure protocol for evaluating recursive graph queries would be also useful in practice. To the best of our knowledge, GOOSE is the first provably-secure system that is able to run UCRPQ on outsourced graphs, without doing any change to the standard SPARQL engine.

Our work is also related to query-based linked data anonymization [9], where the idea is that the data owner, before publishing a graph, adds some noise, specified declaratively using SPARQL. Then, a user is able to download the anonymized graph and query it. However, their hypothesis and ours are different as we assume that the bulk of computations is outsourced to the cloud and our user does not need to do any computation effort other than decrypting the query answers received from the cloud. For us, the challenge is to design a distributed protocol that guarantees that the cloud cannot learn the graph, queries, and query answers, while minimizing the overhead due to cryptographic primitives. On the other hand, their challenge is to anonymize the graph before publishing, while finding a good compromise between privacy and utility.

Summary of Contributions and Paper Organization. In Sect. 2, we introduce some basic notions: graph data and queries, and cryptographic tools. Then, Sect. 3 is the core of our contribution:

- We propose the GOOSE framework for secure graph outsourcing and SPARQL evaluation.
- We formally prove that GOOSE satisfies desirable security properties that we precisely characterize:
  1. No cloud node can learn the graph.
  2. No cloud node can learn at the same time the query and the answers of the query on the graph.
  3. An external network observer cannot learn the graph, the query, or the answers of the query on the graph.
- We analyze the theoretical complexity of GOOSE, by quantifying the number of calls of cryptographic primitives: GOOSE uses a number of AES-CBC encryptions/decryptions that is linear in the input's and output's size.

In Sect. 4, we report on a large-scale empirical evaluation that confirms the theoretical complexity, and shows the scalability of GOOSE. Finally, we conclude our paper and outline directions for future work in Sect. 5.

2 Preliminaries

In Sect. 2.1 we define graph data and queries. In Sect. 2.2 we introduce the AES-CBC symmetric encryption scheme and the notion of IND-CPA security that is useful for proving our protocol's security.

2.1 Graph Data and Queries

Graph Data. An RDF (Resource Description Framework[3]) graph database is a set of triples (s, p, o) where s is the *subject*, p is the *predicate*, and o is the *object*. According to the specification, $s \in \mathcal{I} \cup \mathcal{B}, p \in \mathcal{I}$, $o \in \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$, where $\mathcal{I}, \mathcal{B}, \mathcal{L}$ are three disjoint sets of Internationalized Resource Identifiers (IRIs), blank nodes, and literals, respectively. For the goal of this paper, the distinction between IRIs, blank nodes, and literals is not important. Therefore, we simply assume that a graph database $G = (V, E)$ is a *directed, edge-labeled graph*, where V is a set of nodes and $E \subseteq V \times \Sigma \times V$ is a set of directed edges between nodes of V and with labels from an *alphabet* Σ. For example, the graph in Fig. 2 has:

- Set of nodes $V = \{\text{Alice}, \text{Bob}, \text{Charlie}, \text{David}, \text{Milan}, \text{Paris}\}$. The first four nodes correspond to persons and the last two correspond to cities.
- Alphabet $\Sigma = \{\text{Follows}, \text{ReadsAbout}, \text{TravelsTo}\}$. The first label occurs between two persons and defines the follower relation as in a social network e.g., Twitter. The other two labels occur between a person and a city.

[3] https://www.w3.org/TR/rdf11-concepts/.

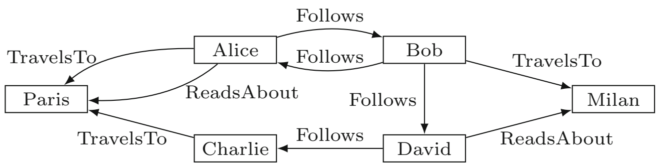


Fig. 2. Example of graph database.

- Set of edges E such as (Alice, Follows, Bob), (Alice, TravelsTo, Paris), etc. There are 9 edges in total, corresponding to the 9 arrows in Fig. 2.

Graph Queries. We focus on *Unions of Conjunctions of Regular Path Queries* (UCRPQ), which are at the core of the W3C's SPARQL 1.1[4]. Recall that Σ is an alphabet and let $\Sigma^+ = \{a, a^- \mid a \in \Sigma\}$, where a^- denotes the *inverse* of the edge label a. Let $V = \{?x, ?y, \ldots\}$ be a set of variables and $n > 0$. A *query rule* is an expression of the form *head* $\leftarrow$ *body*, more precisely:

$$(\overline{?v}) \leftarrow (?x_1, r_1, ?y_1), \ldots, (?x_n, r_n, ?y_n)$$

where: for each $1 \leq i \leq n$, it is the case that $?x_i, ?y_i$ are variables from V and $\overline{?v}$ is a vector of zero or more of these variables, the length of which is called the *arity* of the rule. For each $1 \leq i \leq n$, it is the case that r_i is a regular expression over Σ^+ using $\{\cdot, +, *\}$ (i.e., *concatenation*, *disjunction*, and *Kleene star*). A *query* $Q \in \mathsf{UCRPQ}$ is a finite non-empty set of query rules of the same arity. By $Ans(G, Q)$ we denote the answers of query Q over a graph G, using standard SPARQL semantics. For example, the UCRPQ query

$$(?x, ?z) \leftarrow (?x, \text{Follows}^+, ?y), (?y, \text{TravelsTo}, ?z)$$

selects nodes $?x, ?z$ such that there exists node $?y$ such that one can go from $?x$ to $?y$ with a path in the language of "Follows$^+$" and can go from $?y$ to $?z$ with a path in the language of "TravelsTo". The answers of this query on the graph from Fig. 2 are (Alice, Milan), (Alice, Paris), (Bob, Milan), (Bob, Paris), (David, Paris). For example, the tuple (Alice, Paris) is an answer because of paths Alice $\xrightarrow{\text{Follows}}$ Bob $\xrightarrow{\text{Follows}}$ David $\xrightarrow{\text{Follows}}$ Charlie and Charlie $\xrightarrow{\text{TravelsTo}}$ Paris, where $?x, ?y, ?z$ are mapped to Alice, Charlie, Paris, respectively.

2.2 Cryptographic Tools

We next introduce AES-CBC symmetric encryption and IND-CPA security.

AES-CBC Symmetric Encryption. AES [1] is a NIST standard for symmetric encryption that encrypts messages of 128 bits. AES is used as a block cipher, for instance using CBC mode (Cipher Block Chaining). The AES-CBC cryptosystem is a symmetric encryption scheme defined by a triple of polynomial-time algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ and a security parameter λ such that:

[4] https://www.w3.org/TR/sparql11-query/.

- $\mathsf{KeyGen}(\lambda)$ generates Key, a uniformly random symmetric key whose size depends on λ.
- $\mathsf{Enc}(\mathsf{Key}, m, IV)$ splits m in blocks of 128 bits $m_0, \ldots, m_n$ (padding bits may be added if m_n is smaller than 128 bits). Enc computes $c_0 = \mathsf{E}(\mathsf{Key}, m_0 \oplus IV)$, where E is the AES encryption [1] and IV is a random 128-bits number. By $x \oplus y$ we denote the standard bit-wise xor operation between two bit strings x and y. Then, Enc computes $c_i = \mathsf{E}(\mathsf{Key}, c_{i-1} \oplus m_i)$ for $1 \leq i \leq n$ and returns the tuple $((c_0, \ldots, c_n), IV)$.
- $\mathsf{Dec}(\mathsf{Key}, c, IV)$ splits c in blocks of 128 bits $c_0, \ldots, c_n$ and computes $m_0 = \mathsf{D}(\mathsf{Key}, c_0) \oplus IV$, where D is the AES decryption [1]. Similarly, Dec computes $m_i = \mathsf{D}(\mathsf{Key}, c_i) \oplus c_{i-1}$ for $1 \leq i \leq n$ and returns $m_0, \ldots, m_n$.

IND-CPA [5] (INDistinguishability under Chosen-Plaintext Attack)**.** Let $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a cryptographic scheme. The *probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ tries to break the security of Π. The IND-CPA game, denoted by $\mathrm{EXP}(\mathcal{A})$, works as follows: the adversary $\mathcal{A}$ chooses two messages (m_0, m_1) and receives a challenge $c = \mathsf{Enc}(LR_b(m_0, m_1))$ from the *challenger* who selects a bit $b \in \{0, 1\}$ uniformly at random, and where $LR_b(m_0, m_1)$ is equal to m_0 if $b = 0$, and m_1 otherwise. The adversary, knowing m_0, m_1 and c, is allowed to perform any number of polynomial computations or encryptions of any messages, using the encryption oracle, in order to output a guess b' of the encrypted message in c chosen by the challenger. Intuitively, Π is IND-CPA if there is no PPT adversary that can guess b with a probability significantly better than $\frac{1}{2}$. By $\alpha = \Pr[b' \leftarrow \mathrm{EXP}(\mathcal{A}); b = b']$, we denote the probability that $\mathcal{A}$ correctly outputs her guessed bit b' when the bit chosen by the challenger in the experiment is b. A scheme is IND-CPA secure if $\alpha - \frac{1}{2}$ is negligible function in λ, where a function γ is negligible in λ, denoted $negl(\lambda)$, if for every positive polynomial $p(\cdot)$ and sufficiently large λ, $\gamma(\lambda) < 1/p(\lambda)$. In particular, if f and g are negligible in λ, then $f(\lambda) + g(\lambda)$ is also negligible in λ.

AES-CBC is IND-CPA under the standard assumption that AES is a pseudo-random permutation [5]. We also point out that all theoretical security properties of our protocol also hold if we choose any other IND-CPA symmetric scheme instead of AES-CBC. Our choice to rely on AES-CBC is due to practical reasons since AES-CBC is a NIST standard, and moreover, is very efficient in practice and implemented in standard libraries for modern programming languages.

3 Secure Graph Outsourcing and SPARQL Evaluation

We define the security model and the desired security properties in Sect. 3.1. Then, we propose our secure protocol GOOSE (Sect. 3.2), and we analyze its correctness (Sect. 3.3), security properties (Sect. 3.4), and complexity (Sect. 3.5).

3.1 Security Model

We assume that the cloud is *honest-but-curious* i.e., it executes tasks dutifully, but tries to extract as much information as possible from the data that it sees.

Our model follows the classical formulation in [13] (Ch. 7.5, where *honest-but-curious* is denoted *semi-honest*), in particular (i) each cloud node is trusted: it correctly does the required computations, it does not sniff the network and it does not collude with other nodes, and (ii) an external observer has access to all messages exchanged over the network. The aforementioned security model is of practical interest in a real-world cloud environment. In particular, to satisfy all our theoretical security properties while achieving the no-collusion hypothesis, it suffice to host each cloud node of our protocol by a different cloud provider. This should be feasible as our protocol requires only three cloud nodes.

As already outlined in the introduction and in Fig. 1, we assume that the data (i.e., the graph) and the computations (i.e., the query evaluation algorithm) are outsourced. More precisely, the data owner outsources the graph G to the cloud once at the beginning. Then, the user sends query Q_1 to the cloud and receives $Ans(G, Q_1)$, then the user sends query Q_2 to the cloud and receives $Ans(G, Q_2)$, etc. The user does not have to do any query evaluation on her side. We expect the following *security properties*:

1. No cloud node can learn the graph G.
2. No cloud node can learn at the same time a query Q submitted by the user and the answers $Ans(G, Q)$ of the query Q on the graph G.
3. By analyzing network messages, an external observer cannot learn the graph G, cannot learn any query Q, and cannot learn any $Ans(G, Q)$.

Next, we propose GOOSE, a distributed protocol that satisfies these properties. Intuitively, we achieve the aforementioned properties by exchanging only encrypted messages, and moreover, by distributing the computations among several cloud node participants, each of them having access only to the specific data that it needs for performing its task and nothing else. The challenge is to efficiently distribute tasks among as few cloud participants as possible, while minimizing the time needed for cryptographic primitives.

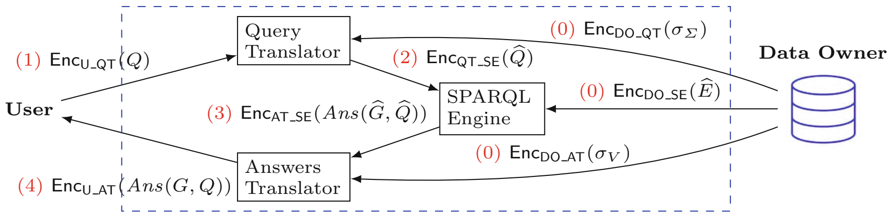

Query Translator
SPARQL Engine
Answers Translator
User
Data Owner
(1) $\mathsf{Enc}_{\mathsf{U\_QT}}(Q)$
(2) $\mathsf{Enc}_{\mathsf{QT\_SE}}(\widehat{Q})$
(3) $\mathsf{Enc}_{\mathsf{AT\_SE}}(Ans(\widehat{G}, \widehat{Q}))$
(4) $\mathsf{Enc}_{\mathsf{U\_AT}}(Ans(G, Q))$
(0) $\mathsf{Enc}_{\mathsf{DO\_QT}}(\sigma_\Sigma)$
(0) $\mathsf{Enc}_{\mathsf{DO\_SE}}(\widehat{E})$
(0) $\mathsf{Enc}_{\mathsf{DO\_AT}}(\sigma_V)$

Fig. 3. Architecture of GOOSE. The dashed rectangle is the cloud. Graph outsourcing (step 0) is done only once at the beginning. Query evaluation (steps 1–4) is done for each submitted query. The hat on some data means that the data is hidden using functions σ_V, σ_Σ, or both (depends on step, see GOOSE description for details).

3.2 Overview of GOOSE

In Fig. 3, we depict the architecture of GOOSE, which has 5 participants: *data owner* (DO), who owns the graph that it outsources to the cloud in order to be queried, *user* (U), who submits graph queries to the cloud and receives query answers, and 3 cloud participants: *query translator* (QT), *SPARQL engine* (SE) and *answers translator* (AT). We next explain each step via a running example.

By Enc_{A_B} or Enc_{B_A} we denote symmetric AES-CBC encryption using the key shared between participants A and B. We have 7 such shared keys because there are 7 combinations of participants exchanging messages, hence 7 arrows in Fig. 3. We assume that the sharing of AES keys has been done before starting the actual protocol and there are many classical key exchange protocols in the literature for doing this.

Step 0. The *graph outsourcing* (i.e., the 3 outgoing arrows from DO in Fig. 3) is done only once at the beginning by DO. Intuitively, DO sends to each cloud participant a piece of the graph such that each participant can perform its task during query evaluation but no participant can reconstruct the entire graph. As shown in the pseudocode of DO in Fig. 4(a), DO generates two random bijections: σ_Σ and σ_V, one for the edge labels and another one for the graph nodes, respectively. By σ^{-1} we denote the inverse of σ (this is needed later on at the end of query evaluation). For our example graph in Fig. 2, DO may generate:

$$\begin{aligned}\sigma_V =& \{\text{Alice} \to 5, \text{Bob} \to 3, \text{Charlie} \to 0, \text{David} \to 1, \text{Milan} \to 2, \text{Paris} \to 4\}\\ \sigma_\Sigma =& \{\text{Follows} \to 1, \text{ReadsAbout} \to 2, \text{TravelsTo} \to 0\}.\end{aligned}$$

Then, DO uses these two functions to hide the graph edges. As shown in Fig. 4(a), by $\widehat{E}$ we denote the hidden set of edges generated from E, where the nodes are replaced using σ_V, and the edge labels are replaced using σ_Σ. On our example graph in Fig. 2, edge (Alice, Follows, Bob) becomes (5, 1, 3), edge (Alice, ReadsAbout, Paris) becomes (5, 2, 4), etc., and finally:

$$\widehat{E} = \{(5,1,3),(5,2,4),(5,0,4),(3,1,5),(3,1,1),(3,0,2),(0,0,4),(1,1,0),(1,2,2)\}.$$

As shown in Figs. 3 and 4(a), DO sends σ_Σ, σ_V, and $\widehat{E}$ to cloud nodes QT, AT, and SE, respectively. Each message sent over the network is encrypted with the key shared between DO and the corresponding cloud participant, which can decrypt the message upon reception. Messages are encrypted to avoid that an external observer that sees them in clear is able to learn the graph G, thus violating one of the desirable security properties stated in Sect. 3.1. Moreover, the distribution of graph storage among cloud participants makes that none of them can learn the graph G, which is also a desirable security property cf. Sect. 3.1.

We next discuss *query evaluation* i.e., steps 1–4 cf. Fig. 3, done for each query submitted by U. Similarly to graph outsourcing, each message exchanged over the network during query evaluation is encrypted with the key shared between corresponding participants, such that an external observer cannot learn the query and its answers to satisfy another desirable security property cf. Sect. 3.1.

```
Let $\sigma_\Sigma$ = random bijection : $\Sigma \rightarrow \{0, \ldots, |\Sigma|-1\}$
Let $\sigma_V$ = random bijection : $V \rightarrow \{0, \ldots, |V|-1\}$
Let $\widehat{E} = \{(\sigma_V(s), \sigma_\Sigma(p), \sigma_V(o)) \mid (s, p, o) \in E\}$
Send $\mathsf{Enc}_{\mathsf{DO\_QT}}(\sigma_\Sigma)$ to QT
Send $\mathsf{Enc}_{\mathsf{DO\_AT}}(\sigma_V)$ to AT
Send $\mathsf{Enc}_{\mathsf{DO\_SE}}(\widehat{E})$ to SE
```

(a) Pseudocode of outsourcing graph $G = (V, E)$ by DO (step 0).

```
Generate query $\widehat{Q}$ from $Q$ by replacing each occurrence of a label $p$ with $\sigma_\Sigma(p)$
Send $\mathsf{Enc}_{\mathsf{QT\_SE}}(\widehat{Q})$ to SE
```

(b) Pseudocode of QT during query evaluation (step 2).

```
Let $\widehat{G} = (\bigcup_{(\widehat{s},\widehat{p},\widehat{o}) \in \widehat{E}} \{\widehat{s}, \widehat{o}\}, \widehat{E})$
Let $Ans(\widehat{G}, \widehat{Q})$ be the answers of $\widehat{Q}$ on $\widehat{G}$, computed with some SPARQL engine
Send $\mathsf{Enc}_{\mathsf{AT\_SE}}(Ans(\widehat{G}, \widehat{Q}))$ to AT
```

(c) Pseudocode of SE during query evaluation (step 3).

```
Let $Ans(G, Q) = \{(\sigma_V^{-1}(v_1), \ldots, \sigma_V^{-1}(v_n)) \mid (v_1, \ldots, v_n) \in Ans(\widehat{G}, \widehat{Q})\}$
Send $\mathsf{Enc}_{\mathsf{U\_AT}}(Ans(G, Q))$ to U
```

(d) Pseudocode of AT during query evaluation (step 4).

Fig. 4. Pseudocode of the non-trivial steps of GOOSE cf. Fig. 3.

Step 1. U submits query Q to QT. For example, recall the query $(?x, ?z) \leftarrow (?x, \text{Follows}^+, ?y), (?y, \text{TravelsTo}, ?z)$ from Sect. 2.

Step 2. QT translates the received query Q by replacing all labels used in Q using the function σ_Σ received from DO, as shown in Fig. 4(b). By $\widehat{Q}$ we denote the query Q translated using σ_Σ. On our running example, the query from step 1 becomes $(?x, ?z) \leftarrow (?x, 1^+, ?y), (?y, 0, ?z)$.

Step 3. As shown in Fig. 4(c), SE evaluates translated query $\widehat{Q}$ received from QT at step 2 on the graph with hidden nodes and edge labels as defined by $\widehat{E}$ received from DO during step 0. To do so, SE simply uses some standard SPARQL engine as a black-box, without any change to the query engine. We denote the result of SE by $Ans(\widehat{G}, \widehat{Q})$, where the true answers $Ans(G, Q)$ are still hidden using function σ_V. On our running example, $Ans(\widehat{G}, \widehat{Q}) = \{(5, 2), (5, 4), (3, 2), (3, 4), (1, 4)\}$.

Step 4. AT uses the function $\sigma_{V^{-1}}$ to translate the received hidden query answers $Ans(\widehat{G}, \widehat{Q})$ into the true query answers, as shown in Fig. 4(d). On our running example, AT recovers $Ans(G, Q) = \{(\text{Alice}, \text{Milan}), (\text{Alice}, \text{Paris}), (\text{Bob}, \text{Milan}), (\text{Bob}, \text{Paris}), (\text{David}, \text{Paris})\}$ that AT sends to U.

3.3 Correctness of GOOSE

To show the correctness, we point out a reduction from GOOSE to the standard SPARQL evaluation engine used as a black-box in SE. Take a graph G outsourced by DO and a query Q submitted by U. If we remove all encryptions/decryptions of GOOSE, hence all messages are communicated in clear between participants, then we obtain protocol GOOSE$'$ that yields exactly the same result as GOOSE. This happens because of the consistency property of AES-CBC i.e., if we encrypt message M using Enc to obtain ciphertext C, then if we decrypt C using Dec we obtain exactly M. Next, take the SE participant of GOOSE$'$, which evaluates query $\widehat{Q}$ (cf. Fig. 4(b)) over graph $\widehat{G}$ (cf. Fig. 4(c)). Since DO and QT use the same function σ_Σ for hiding edge labels, then $Ans(\widehat{G}, \widehat{Q}) = Ans(\widehat{G}', Q)$, where $\widehat{G}' = (\{0, \ldots, |V|-1\}, \{(\sigma_V(s), p, \sigma_V(o)) \mid (s,p,o) \in E\})$. Then, AT inverses the function σ_V on each value of each tuple of $Ans(\widehat{G}, \widehat{Q})$ cf. Fig. 4(d) and generates exactly $Ans(G, Q)$ because AT uses the same function σ_V that is used in $\widehat{G}'$. In conclusion, U receives the correct answers $Ans(G, Q)$.

3.4 Security of GOOSE

We next show that GOOSE satisfies the desirable properties outlined in Sect. 3.1, proven as Theorems 1, 2, and 3. In the sequel, by $data_A$, we denote the data to which A has access, where A can be a cloud participant (QT, SE, AT) or an external network observer (ext). We first characterize $data$ for all participants. Given a graph $G = (V, E)$ and a workload of k queries $Q_1, \ldots, Q_k$:

$$data_{\mathsf{QT}} = \{\sigma_\Sigma\} \cup \bigcup_{1 \le i \le k} \{Q_i\},$$

$$data_{\mathsf{SE}} = \{\widehat{E}\} \cup \bigcup_{1 \le i \le k} \{\widehat{Q}_i,\ Ans(\widehat{G}, \widehat{Q_i})\},$$

$$data_{\mathsf{AT}} = \{\sigma_V\} \cup \bigcup_{1 \le i \le k} \{Ans(G, Q_i)\},$$

$$data_{ext} = \{\mathsf{Enc}_{\mathsf{DO\_QT}}(\sigma_\Sigma), \mathsf{Enc}_{\mathsf{DO\_SE}}(\widehat{E}), \mathsf{Enc}_{\mathsf{DO\_AT}}(\sigma_V)\} \cup \bigcup_{1 \le i \le k} \{\mathsf{Enc}_{\mathsf{U\_QT}}(Q_i), \mathsf{Enc}_{\mathsf{QT\_SE}}(\widehat{Q}_i), \mathsf{Enc}_{\mathsf{AT\_SE}}(Ans(\widehat{G}, \widehat{Q_i})), \mathsf{Enc}_{\mathsf{U\_AT}}(Ans(G, Q_i))\}.$$

We next show that GOOSE satisfies Property 1 from Sect. 3.1.

Theorem 1. *For each cloud participant $A \in \{\mathsf{QT}, \mathsf{SE}, \mathsf{AT}\}$, A cannot guess from $data_A$ the graph $G = (V, E)$ with probability better than random under the assumption that bijections σ_Σ and σ_V are pseudorandom.*

Proof.
- QT. By construction, $data_{\mathsf{QT}}$ does not include information on the nodes of G. In particular, QT does not know V or even $|V|$. Moreover, although $data_{\mathsf{QT}}$ include σ_Σ from which QT can infer Σ, there is no other information available on E, not even $|E|$. Hence, if QT wants to guess G, its

best strategy is random i.e., pick random set of nodes V' and edges (s, p, o), with $s, o \in V'$ and $p \in \Sigma$.

- SE. By construction, SE can learn, from $data_{\mathsf{SE}}$, a graph $\widehat{G}$ (cf. Fig. 4(c)) isomorphic to G. If SE can learn G from $\widehat{G}$, this implies that SE can learn the two pseudorandom bijections σ_Σ and σ_V. This cannot be done with probability better than random because $data_{\mathsf{SE}}$ does not include information on the nodes and edges of G hence SE sees no information on the domains of σ_Σ and σ_V.
- AT. By construction, $\sigma_V \in data_{\mathsf{AT}}$ thus AT can infer V. However, $data_{\mathsf{AT}}$ does not include information on the edges of G, not even $|E|$ or $|\Sigma|$. Hence, if AT wants to guess G, its best strategy is random i.e., pick random alphabet Σ' and random edges (s, p, o), with $s, o \in V$ and $p \in \Sigma'$. □

We next show that GOOSE satisfies Property 2 from Sect. 3.1.

Theorem 2. *For each cloud participant $A \in \{\mathsf{QT}, \mathsf{SE}, \mathsf{AT}\}$, A cannot guess from $data_A$, at the same time, a query Q and its answers $Ans(G, Q)$ with probability better than random under the assumption that bijections σ_Σ and σ_V are pseudorandom.*

Proof.
- QT. By construction, QT knows the query Q, but does not see any information on $Ans(G, Q)$. Its best strategy to guess $Ans(G, Q)$ is random i.e., pick a random set of nodes V' and output a random set of tuples of the same arity as Q using nodes of V'.
- SE. By construction, SE knows the number of answers $|Ans(G, Q))|$, without knowing to which true nodes the answers correspond and what is query Q. The best strategy of SE for guessing $Ans(G, Q)$ is random, similar to the QT case.
- AT. By construction, AT knows the query result $Ans(G, Q)$, but does not see any information on Q or on alphabet Σ. Hence, if AT wants to guess Q, its best strategy is random i.e., pick a random alphabet Σ' and output a random query over Σ' that has the same arity as $Ans(G, Q)$. □

We next prove the security of an external observer, more precisely we show that GOOSE satisfies Property 3 from Sect. 3.1.

Theorem 3. *Given $data_{ext}$, then the graph $G = (V, E)$, any query Q, and any query answers $Ans(G, Q)$ are* indistinguishable *of random for an external network observer of* GOOSE *under the assumption that the symmetric encryption used is IND-CPA.*

Proof. This proof relies on the notion of IND-CPA security as defined in Sect. 2. Before proving the theorem, we first need to introduce some notation:

- By $\mathcal{A}^{pb}(data_{ext})$ we denote the guess of a Probabilistic Polynomial Time (PPT) adversary $\mathcal{A}$ that knows $data_{ext}$ and tries to solve problem pb among: *guessE* (that returns the $\mathcal{A}$'s guess of some graph edge in E), *guessQ* (that returns the $\mathcal{A}$'s guess of some query Q in the workload), *guessAns* (that returns the $\mathcal{A}$'s guess of the answers $Ans(G, Q)$ of some query Q in the workload).

- By construction of $data_{ext}$, we infer that *ext* can learn, based on $data_{ext}$, size estimates of the graph components $|E|$, $|V|$, $|\Sigma|$ (from the messages exchanged at step 0), and size estimates $|Q|$ for each query and $|Ans(G, Q)|$ for its query answers (from the messages exchanged at steps 1–4). By $p_E(data_{ext})$, $p_Q(data_{ext})$, $p_{Ans}(data_{ext})$, we denote the probability that *ext* randomly outputs, based on $data_{ext}$, a correct graph edge, a correct query, or a correct query answers set, respectively.

Hence, to prove the theorem, we need to prove that, for a graph $G = (V, E)$ and a query workload $\bigcup_{1 \leq i \leq k}\{Q_i\}$, for all PPT adversaries $\mathcal{A}$,

$$\begin{aligned}
&|\Pr[\mathcal{A}^{guessE}(data_{ext}) \in E] - p_E(data_{ext})| && \text{is negligible in } \lambda,\\
&|\Pr[\mathcal{A}^{guessQ}(data_{ext}) \in \bigcup_{1\leq i\leq k}\{Q_i\}] - p_Q(data_{ext})| && \text{is negligible in } \lambda,\\
&|\Pr[\mathcal{A}^{guessAns}(data_{ext}) \in \bigcup_{1\leq i\leq k}\{Ans(G, Q_i)\}] - p_{Ans}(data_{ext})| && \text{is negligible in } \lambda.
\end{aligned}$$

Each of the 3 statements can be proven separately by contradiction. We prove here only the first statement, the other two proofs being similar and omitted here due to space constraints, but available in Appendix A.

We assume, toward a contradiction, that there exists a PPT adversary $\mathcal{A}$ able from $data_{ext}$ to find a correct edge in E with a non-negligible advantage x:

$$|\Pr[\mathcal{A}^{guessE}(data_{ext}) \in E] - p_E(data_{ext})| = x + negl(\lambda).$$

If $data_{ext}$ does not correspond to an actual collection of encrypted messages as *ext* sees during an execution of GOOSE, then the advantage for such an input is naturally negligible.

We show that by using the adversary $\mathcal{A}$, we can construct an adversary $\mathcal{B}$ able to break the IND-CPA property of AES-CBC [1,5]. We build an IND-CPA game, in which $\mathcal{B}$ chooses two values m_0, m_1, and sends them to the challenger. The challenger randomly selects $b \in \{0, 1\}$ and answers with $\mathsf{Enc}_{\mathsf{DO\_QT}}(m_b)$. Adversary $\mathcal{B}$ wins the IND-CPA game if $\mathcal{B}$ guesses b with a non-negligible advantage.

To do so, $\mathcal{B}$ simulates a GOOSE execution i.e., $\mathcal{B}$ chooses a graph (over alphabet $\Sigma = \{a_1, \ldots, a_n\}$), a query workload, and the functions (σ_Σ and σ_V) used by DO during graph outsourcing; $\mathcal{B}$ does not know the keys shared among the participants of GOOSE. Let $data_{\mathcal{B}}$ be the set of encrypted messages exchanged during the GOOSE simulation (including $\mathsf{Enc}_{\mathsf{DO\_QT}}(\sigma_\Sigma)$, among others). As input for the IND-CPA game, $\mathcal{B}$ chooses $m_1 = \sigma_\Sigma$ and $m_0 = \sigma'_\Sigma$, where for $a \in \Sigma$, if $\sigma_\Sigma(a) \neq 0$, then $\sigma'_\Sigma(a) = \sigma_\Sigma(a)$, else $\sigma'_\Sigma(a) = -1$. Then, $\mathcal{B}$ sends m_0, m_1 to the challenger, and receives $\mathsf{Enc}_{\mathsf{DO\_QT}}(m_b)$. Next, $\mathcal{B}$ calls $\mathcal{A}^{guessE}(data_{\mathcal{B}} \setminus \{\mathsf{Enc}_{\mathsf{DO\_QT}}(\sigma_\Sigma)\} \cup \{\mathsf{Enc}_{\mathsf{DO\_QT}}(m_b)\})$. The strategy of $\mathcal{B}$ is as follows: if $\mathcal{A}$ returns a true edge having the label a for which $\sigma_\Sigma(a) = 0$, then $\mathcal{B}$ answers 1. Otherwise, $\mathcal{B}$ answers randomly. We next derive the probability of a correct answer by $\mathcal{B}$:

- If $b = 0$ (probability $\frac{1}{2}$), then $\mathcal{A}$ does not receive a correct simulation because the functions used during graph outsourcing to compute the pieces sent to QT and SE are not the same. According to our assumption, $\mathcal{A}$ does not give any advantage. $\mathcal{B}$ answers randomly and is right with probability $\frac{1}{2}$, hence the probability of success of this branch is $\frac{1}{4}$.
- If $b = 1$ (probability $\frac{1}{2}$), then $\mathcal{B}$ can leverage the advantage given by $\mathcal{A}$.
  - If $\mathcal{A}$ returns a true edge having the label a for which $\sigma_\Sigma(a) = 0$ (probability $p_E(data_\mathcal{B}) + x + negl(\lambda)$), then $\mathcal{B}$ correctly answers 1. The probability of success of this branch is $\frac{1}{2}(p_E(data_\mathcal{B}) + x + negl(\lambda))$.
  - Otherwise, (probability $1 - p_E(data_\mathcal{B}) - x - negl(\lambda)$), $\mathcal{B}$ answers randomly and is correct with probability $\frac{1}{2}$. This branch yields a probability of success of $\frac{1}{2}(1 - p_E(data_\mathcal{B}) - x - negl(\lambda))\frac{1}{2}$.

By aggregating these cases, the probability α of success of $\mathcal{B}$ is:

$$\begin{aligned}\alpha &= \frac{1}{4} + \frac{1}{2}(p_E(data_\mathcal{B}) + x + negl(\lambda)) + \frac{1}{2}(1 - p_E(data_\mathcal{B}) - x - negl(\lambda))\frac{1}{2}\\ &= \frac{1}{4} + \frac{p_E(data_\mathcal{B})}{2} + \frac{x}{2} + \frac{1}{4} - \frac{p_E(data_\mathcal{B})}{4} - \frac{x}{4} + negl(\lambda)\\ &= \frac{1}{2} + \frac{p_E(data_\mathcal{B})}{4} + \frac{x}{4} + negl(\lambda)\end{aligned}$$

Note that $p_E(data_\mathcal{B}) \geq 0$ (since it is a probability) and recall that x is non-negligible (by hypothesis). Hence, $\mathcal{B}$ has a non-negligible advantage of $\frac{p_E(data_\mathcal{B})}{4} + \frac{x}{4}$ in the IND-CPA game, which contradicts the fact that AES-CBC is IND-CPA secure. Hence, we conclude that there does not exist any PPT adversary $\mathcal{A}$ that violates the property stated in the theorem. □

3.5 Complexity of GOOSE

The number of cryptographic operations used by GOOSE is linear in the input's and output's size:

Theorem 4. *Given a graph $G = (V, E)$ over an alphabet Σ and a workload of k queries $Q_1, \ldots, Q_k$, the total size that* GOOSE *encrypts, as well as the total size that it decrypts, is*

$$|\Sigma| + |V| + |E| + 2 \sum_{1 \leq i \leq k} (|Q_i| + |Ans(G, Q_i)|).$$

Proof. This follows from the construction of GOOSE. During graph outsourcing, the size of encrypted data by DO is $|\Sigma| + |V| + |E|$, which is also the size of data decrypted by the cloud participants. For evaluating query $1 \leq i \leq k$ in the workload, the size of encrypted data is $|Q_i| + |Q_i| + |Ans(G, Q_i)| + |Ans(G, Q_i)|$, done sequentially by $\mathsf{U}, \mathsf{QT}, \mathsf{SE}, \mathsf{AT}$, and the size of decrypted data is the same, done sequentially by $\mathsf{QT}, \mathsf{SE}, \mathsf{AT}, \mathsf{U}$. By summing up the size for all queries in the workload, we obtain exactly the formula in the theorem statement. □

4 Experiments

We present a large-scale empirical evaluation devoted to showing the practical feasibility and scalability of GOOSE, for both graph outsourcing and query evaluation. We also compare GOOSE query evaluation with standard SPARQL evaluation and we zoom on the running time shares of each GOOSE participant.

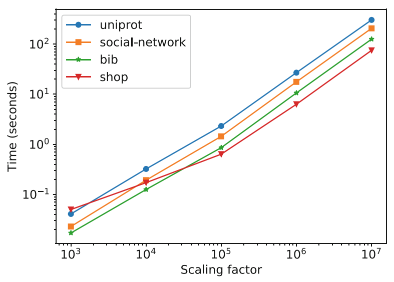
Implementation. We implemented GOOSE in Python 3. For AES-CBC we used keys of 256 bits with the *PyCryptodome* library[5]. As SPARQL engine, we used Apache Jena[6]. We carried out our experiments on a system with CPU Intel Xeon of 3 GHz and 755 GB of RAM, running CentOS Linux 7.

Open-Source Code. For reproducibility reasons, we make available on a public GitHub repository[7] our source code, together with scripts that install needed libraries, run the large-scale experiment, and generate the plots. This experiment took 8 days on our system and generated 46 GB of data (total size for graphs, queries, and query answers).

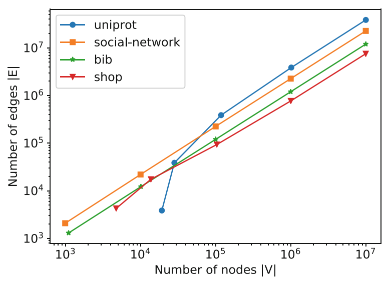
Datasets. We relied on gMark[8] [3,4], a schema-driven benchmark that allows generating synthetic graphs and queries with finely-tuned constraints. gMark provided us a large quantity of diverse data and queries to stress-test GOOSE as we used all 4 use cases that we found on the gMark repository: *uniprot* (biological data where proteins interact with other proteins, are encoded on genes, etc.), *shop* (online shop selling different types of products to users, etc.), *social-network* (social network where persons know other persons, work in companies, etc.), and *bib* (bibliographical data about researchers that author papers published in journals or conferences, etc.). Each use case encodes different types of constraints, which make the generated graphs and queries have different characteristics, that we detail when necessary to explain experimental results.

Scalability of Graph Outsourcing. For each of the 4 use cases, we consider 5 scaling factors, from 10^3 to 10^7, where a scaling factor n means that gMark should generate a graph with n nodes. For each combination (use case, scaling factor), we report the GOOSE graph outsourcing time, averaged over 10 graphs, each of them outsourced 3 times. We show the result of this experiment in Fig. 5(a), where we observe a smooth, linear-time behavior. We next explain the running times difference between the use cases by detailing their characteristics. In particular, the number of generated nodes for a scaling factor depends on how large is n and what constraints are specified in the use case. This is why, to help understanding the behavior in Fig. 5(a), we also plot in Fig. 5(b) the size (# of nodes vs # of edges) for the generated graphs. To simulate realistic graph constraints, each use case specifies how the number of nodes of some type increases: there are types of nodes whose number increases when the graph size

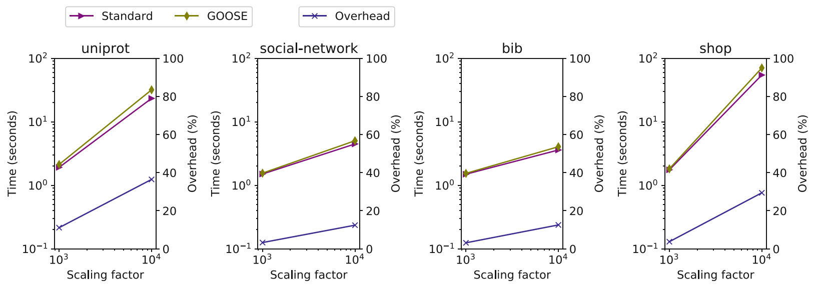
[5] https://pycryptodome.readthedocs.io/en/latest/src/cipher/classic.html.
[6] https://jena.apache.org/.
[7] https://github.com/radu1/goose.
[8] https://github.com/graphMark/gmark.



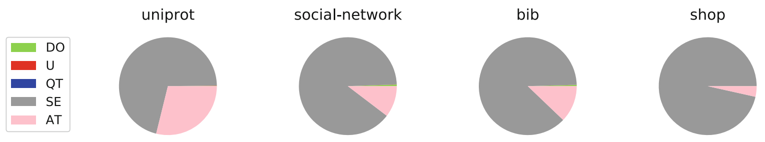
(a) Scalability of graph outsourcing.



(b) Size of graphs in dataset.



(c) Scalability of query evaluation, and comparison standard vs GOOSE.



(d) Zoom on end-to-end solution i.e., graph outsourcing and query evaluation for a workload of 5 queries, for graphs of fixed scaling factor 10^4. The shares of participants DO, U, and QT are barely visible.

Fig. 5. Experimental results.

increases (e.g., users and purchases in *shop*), and types of nodes whose number is constant for all graph sizes (e.g., cities and countries in *shop*). When we take a small scaling factor and a use case with strong constraints on the types with constant number of occurrences, gMark may have to add nodes beyond the size specified by the scaling factor to satisfy the number of nodes for each type. This explains the behavior for small scaling factors in Fig. 5(b) for *shop* (12 types of constant node types), and *uniprot* (3 types of constant node types, among which one with 15K occurrences, hence for the scaling factor 10^3, the generated graphs have at least 15 times more nodes). For large scaling factors ($10^5, 10^6, 10^7$), the number of constant node types is dominated by the nodes with types that increase with the graph size, hence the hierarchy of the use cases in terms of size is clear and determined by the number of edges that should be generated in each use case. We conclude this experiment by observing that the GOOSE graph outsourcing time is strongly correlated to the graph size: if you take any two graphs A and B, if A has more edges than B (cf. Y axis in Fig. 5(b)), then the time to outsource A is larger than the time to outsource B (cf. Y axis in Fig. 5(a)). This is particularly visible for large scaling factors, where the hierarchy of the generated graph sizes (in terms of # of edges) is strictly followed by the hierarchy of use cases in terms of graph outsourcing times.

Scalability of Query Evaluation. For each of the 4 use cases, for each of the scaling factors 10^3 and 10^4, we generate with gMark 200 graphs and a workload of 5 queries coupled to each graph. Hence, we have run a total number of 8000 queries, having diverse properties specified in the gMark use cases. In particular, for each use case the generated queries are unary/binary, recursive/non-recursive (i.e., contain Kleene stars or not), linear/constant (i.e., return a number of answers that depend or not on the size of the graph), and have various shapes (chain, star, cycle, star-chain). Although we were able to easily scale the GOOSE graph outsourcing up to scaling factor 10^7, for the query evaluation experiment we evaluated queries only up to 10^4 because the bottleneck of this experiment is the standard SPARQL engine. Indeed, if we simply evaluate a generated query on a generated graph of scaling factor 10^4, it may happen that this takes already up to a minute, without any GOOSE security. This limitation of current SPARQL engines, in particular for evaluating recursive queries, has been already pointed out in the literature e.g., in [4]. Hence, we were able to benchmark GOOSE query evaluation vs standard query evaluation only on scaling factors 10^3 and 10^4, and we run 3 times each query with each system before averaging. We show our results in Fig. 5(c). We observe that the running times depend on the use case in the sense that if a graph has more nodes, it is more likely that a query has more results hence it may take more time to enumerate all results. This is why *uniprot* and *shop* take more time than the others. If we compare the relative performance of standard SPARQL evaluation vs GOOSE query evaluation, we observe that the overhead due to cryptographic primitives in GOOSE is dominated by the time taken by the GOOSE SPARQL engine. We also plot the relative overhead, which obviously increases when there are more query answers to encrypt and decrypt during steps 3 and 4 in GOOSE. Hence, a large overhead in this experiment is correlated to a large share of the answers translator in the next one.

Zoom of End-to-End Solution. In this last experiment, we see GOOSE as an end-to-end solution consisting of outsourcing a graph and then evaluating several queries on it. In Fig. 5(d), we show the time shares taken by each GOOSE participant, for each of the 4 use cases, for fixed scaling factor 10^4, after summing up the times needed for graph outsourcing (cf. the first experiment) and for evaluating all 5 queries in the workload (cf. the second experiment). As expected, the SPARQL engine takes the lion's share. Moreover, the next most visible participant is the answers translator, which has to decrypt hidden answers received from the SPARQL engine, translate the answers, and re-encrypt the true answers before sending them to the user. Without surprise, the time shares taken by the two participants outside the cloud (data owner and user) are negligible, the bulk of the computation being outsourced to the cloud.

5 Conclusions and Future Work

We presented the design and implementation of the GOOSE secure framework for outsourcing graphs and querying them with SPARQL queries defined by UCRPQ. We formally proved that GOOSE enjoys desirable security properties and that its overhead due to cryptographic primitives is linear in the input's and output's size. Our large-scale experimental study confirms the scalability of GOOSE. As future work, we plan to extend GOOSE to support other practical SPARQL features such as aggregates and comparisons, for which we need to use cryptographic schemes such as Paillier [17] and order-preserving encryption [6].

A Appendix: Proof of Theorem 3 (Continued)

Recall that Theorem 3 states: *Given* $data_{ext}$*, then the graph* $G = (V, E)$*, any query* Q*, and any query answers* $Ans(G, Q)$ *are indistinguishable of random for an external network observer of* GOOSE *under the assumption that the symmetric encryption used is IND-CPA.*

In the main body of the paper (Sect. 3.4), we started the proof of Theorem 3, where we have first shown that proving the theorem boils down to proving that for a graph $G = (V, E)$ and a query workload $\bigcup_{1\le i\le k}\{Q_i\}$, for all PPT adversaries $\mathcal{A}$,

$$\begin{aligned}
&|\Pr[\mathcal{A}^{guessE}(data_{ext}) \in E] - p_E(data_{ext})| && \text{is negligible in } \lambda,\\
&|\Pr[\mathcal{A}^{guessQ}(data_{ext}) \in \bigcup_{1\le i\le k}\{Q_i\}] - p_Q(data_{ext})| && \text{is negligible in } \lambda,\\
&|\Pr[\mathcal{A}^{guessAns}(data_{ext}) \in \bigcup_{1\le i\le k}\{Ans(G, Q_i)\}] - p_{Ans}(data_{ext})| && \text{is negligible in } \lambda.
\end{aligned}$$

In the main body of the paper (Sect. 3.4), we have proven the first of the aforementioned statements, and we omitted the other two due to space constraints. We prove here the other two statements.

We assume, toward a contradiction, that there exists a PPT adversary $\mathcal{A}$ able from $data_{ext}$ to find a correct query Q_i with a non-negligible advantage x:

$$|\Pr[\mathcal{A}^{guessQ}(data_{ext}) \in \bigcup_{1\le i\le k}\{Q_i\}] - p_Q(data_{ext})| = x + negl(\lambda).$$

If $data_{ext}$ does not correspond to an actual collection of encrypted messages as *ext* sees during an execution of GOOSE, then the advantage for such an input is naturally negligible.

We next show that by using the adversary $\mathcal{A}$, we can construct an adversary $\mathcal{B}$ able to break the IND-CPA property of AES-CBC [1,5]. We build an IND-CPA game, in which $\mathcal{B}$ chooses two values m_0, m_1, and sends them to the challenger. The challenger randomly selects $b \in \{0,1\}$ and answers with $\mathsf{Enc}_{\mathsf{U\_QT}}(m_b)$. Adversary $\mathcal{B}$ wins the IND-CPA game if $\mathcal{B}$ guesses b with a non-negligible advantage.

To do so, $\mathcal{B}$ simulates a GOOSE execution i.e., $\mathcal{B}$ chooses a graph, a query workload consisting of a single query Q_1 that is $?x \leftarrow (?x, a+, ?y)$, such that $Ans(G, Q_1) \neq \emptyset$, and the functions used by DO during graph outsourcing; $\mathcal{B}$ does not know the keys shared among the participants of GOOSE. Let $data_{\mathcal{B}}$ be the set of encrypted messages seen by an external observer of the simulation of GOOSE done by $\mathcal{B}$, which includes, among others, $\mathsf{Enc}_{\mathsf{U\_QT}}(Q_1)$.

As input for the IND-CPA game, $\mathcal{B}$ chooses $m_1 = Q_1$ and $m_0 = Q_1'$ obtained by replacing a in Q_1 by a fresh label $a' \notin \Sigma$. Then, $\mathcal{B}$ sends m_0, m_1 to the challenger, and receives $\mathsf{Enc}_{\mathsf{U\_QT}}(m_b)$. Next, $\mathcal{B}$ calls $\mathcal{A}^{guessQ}(data_{\mathcal{B}} \setminus \{\mathsf{Enc}_{\mathsf{U\_QT}}(Q_1)\} \cup \{\mathsf{Enc}_{\mathsf{U\_QT}}(m_b)\})$. The strategy of $\mathcal{B}$ is as follows: if $\mathcal{A}$ returns Q_1, then $\mathcal{B}$ answers 1. Otherwise, $\mathcal{B}$ answers randomly. We next derive the probability of a correct answer by $\mathcal{B}$:

- If $b = 0$ (probability $\frac{1}{2}$), then $\mathcal{A}$ does not receive a correct simulation because $a' \notin \Sigma$, hence $Ans(G, Q_1') = \emptyset$ that is different from $Ans(G, Q_1) \neq \emptyset$ that also belongs to $data_{ext}$. According to our assumption, in such a case $\mathcal{A}$ does not give any advantage. $\mathcal{B}$ answers randomly and is right with probability $\frac{1}{2}$, hence the probability of success of this branch is $\frac{1}{4}$.
- If $b = 1$ (probability $\frac{1}{2}$), then $\mathcal{B}$ can leverage the advantage given by $\mathcal{A}$.
  - If $\mathcal{A}$ returns Q_1 (probability $p_Q(data_{\mathcal{B}}) + x + negl(\lambda)$), then $\mathcal{B}$ correctly answers 1. The probability of success of this branch is $\frac{1}{2}(p_Q(data_{\mathcal{B}}) + x + negl(\lambda))$.
  - Otherwise, (probability $1 - p_Q(data_{\mathcal{B}}) - x - negl(\lambda)$), $\mathcal{B}$ answers randomly and is correct with probability $\frac{1}{2}$. This branch yields a probability of success of $\frac{1}{2}(1 - p_Q(data_{\mathcal{B}}) - x - negl(\lambda))\frac{1}{2}$.

By aggregating these cases, the probability α of success of $\mathcal{B}$ is:

$$\begin{aligned}\alpha &= \frac{1}{4} + \frac{1}{2}(p_Q(data_{\mathcal{B}}) + x + negl(\lambda)) + \frac{1}{2}(1 - p_Q(data_{\mathcal{B}}) - x - negl(\lambda))\frac{1}{2} \\ &= \frac{1}{4} + \frac{p_Q(data_{\mathcal{B}})}{2} + \frac{x}{2} + \frac{1}{4} - \frac{p_Q(data_{\mathcal{B}})}{4} - \frac{x}{4} + negl(\lambda) \\ &= \frac{1}{2} + \frac{p_Q(data_{\mathcal{B}})}{4} + \frac{x}{4} + negl(\lambda)\end{aligned}$$

Note that $p_E(data_{\mathcal{B}}) \geq 0$ (since it is a probability) and recall that x is non-negligible (by hypothesis). Hence, $\mathcal{B}$ has a non-negligible advantage of $\frac{p_Q(data_{\mathcal{B}})}{4} + \frac{x}{4}$ in the IND-CPA game, which contradicts the fact that AES-CBC is IND-CPA secure. Hence, we conclude that there does not exist any PPT adversary $\mathcal{A}$ that violates the property stated in the theorem.

Next, we assume, toward a contradiction, that there exists a PPT adversary $\mathcal{A}$ able from $data_{ext}$ to find correct query answers $Ans(G, Q_i)$ with a non-negligible advantage x:

$$|\Pr[\mathcal{A}^{guessAns}(data_{ext}) \in \bigcup_{1 \leq i \leq k} \{Ans(G, Q_i)\}] - p_{Ans}(data_{ext})| = x + negl(\lambda).$$

Similarly to the previous statement, if $data_{ext}$ does not correspond to an actual collection of encrypted messages as *ext* sees during an execution of GOOSE, then the advantage for such an input is naturally negligible.

We next show that by using the adversary $\mathcal{A}$, we can construct an adversary $\mathcal{B}$ able to break the IND-CPA property of AES-CBC [1,5]. We build an IND-CPA game, in which $\mathcal{B}$ chooses two values m_0, m_1, and sends them to the challenger. The challenger randomly selects $b \in \{0,1\}$ and answers with $\mathsf{Enc}_{\mathsf{U\_AT}}(m_b)$. Adversary $\mathcal{B}$ wins the IND-CPA game if $\mathcal{B}$ guesses b with a non-negligible advantage.

To do so, $\mathcal{B}$ simulates a GOOSE execution i.e., $\mathcal{B}$ chooses a graph G, a query workload consisting of a single query Q_1 such that $Ans(G, Q_1) = \{(v_1)\}$, and the functions used by DO during graph outsourcing; $\mathcal{B}$ does not know the keys shared among the participants of GOOSE. Let $data_{\mathcal{B}}$ be the set of encrypted messages seen by an external observer of the simulation of GOOSE done by $\mathcal{B}$, which includes, among others, $\mathsf{Enc}_{\mathsf{U\_AT}}(Ans(G, Q_1))$.

As input for the IND-CPA game, $\mathcal{B}$ chooses $m_1 = Ans(G, Q_1)$ and $m_0 = \{(v'_1)\}$, where v'_1 is a fresh node $\notin V$. Then, $\mathcal{B}$ sends m_0, m_1 to the challenger, and receives $\mathsf{Enc}_{\mathsf{U\_AT}}(m_b)$. Next, $\mathcal{B}$ calls $\mathcal{A}^{guessAns}(data_{\mathcal{B}} \setminus \{\mathsf{Enc}_{\mathsf{U\_AT}}(Ans(G, Q_1))\} \cup \{\mathsf{Enc}_{\mathsf{U\_AT}}(m_b)\})$. The strategy of $\mathcal{B}$ is as follows: if $\mathcal{A}$ returns $Ans(G, Q_1)$, then $\mathcal{B}$ answers 1. Otherwise, $\mathcal{B}$ answers randomly. We next derive the probability of a correct answer by $\mathcal{B}$:

- If $b = 0$ (probability $\frac{1}{2}$), then $\mathcal{A}$ does not receive a correct simulation because $v' \notin V$, hence v' could not belong to the answer set of a query from the workload. According to our assumption, in such a case $\mathcal{A}$ does not give any advantage. $\mathcal{B}$ answers randomly and is right with probability $\frac{1}{2}$, hence the probability of success of this branch is $\frac{1}{4}$.
- If $b = 1$ (probability $\frac{1}{2}$), then $\mathcal{B}$ can leverage the advantage given by $\mathcal{A}$.
    - If $\mathcal{A}$ returns $Ans(G, Q_1)$ (probability $p_{Ans}(data_{\mathcal{B}}) + x + negl(\lambda)$), then $\mathcal{B}$ correctly answers 1. The probability of success of this branch is $\frac{1}{2}(p_{Ans}(data_{\mathcal{B}}) + x + negl(\lambda))$.
    - Otherwise, (probability $1 - p_{Ans}(data_{\mathcal{B}}) - x - negl(\lambda)$), $\mathcal{B}$ answers randomly and is correct with probability $\frac{1}{2}$. This branch yields a probability of success of $\frac{1}{2}(1 - p_{Ans}(data_{\mathcal{B}}) - x - negl(\lambda))\frac{1}{2}$.

By aggregating these cases, the probability α of success of $\mathcal{B}$ is:

$$\begin{aligned}\alpha &= \frac{1}{4} + \frac{1}{2}(p_{Ans}(data_{\mathcal{B}}) + x + negl(\lambda)) + \frac{1}{2}(1 - p_{Ans}(data_{\mathcal{B}}) - x - negl(\lambda))\frac{1}{2} \\ &= \frac{1}{4} + \frac{p_{Ans}(data_{\mathcal{B}})}{2} + \frac{x}{2} + \frac{1}{4} - \frac{p_{Ans}(data_{\mathcal{B}})}{4} - \frac{x}{4} + negl(\lambda) \\ &= \frac{1}{2} + \frac{p_{Ans}(data_{\mathcal{B}})}{4} + \frac{x}{4} + negl(\lambda)\end{aligned}$$

Note that $p_E(data_{\mathcal{B}}) \geq 0$ (since it is a probability) and recall that x is non-negligible (by hypothesis). Hence, $\mathcal{B}$ has a non-negligible advantage of $\frac{p_{Ans}(data_{\mathcal{B}})}{4} + \frac{x}{4}$ in the IND-CPA game, which contradicts the fact that AES-CBC is IND-CPA secure. Hence, we conclude that there does not exist any PPT adversary $\mathcal{A}$ that violates the property stated in the theorem. $\square$

References

1. Advanced Encryption Standard (AES), FIPS Publication 197 (2001). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf
2. Aburawi, N., Lisitsa, A., Coenen, F.: Querying encrypted graph databases. In: ICISSP, pp. 447–451 (2018)
3. Bagan, G., Bonifati, A., Ciucanu, R., Fletcher, G.H.L., Lemay, A., Advokaat, N.: Generating flexible workloads for graph databases. PVLDB **9**(13), 1457–1460 (2016)
4. Bagan, G., Bonifati, A., Ciucanu, R., Fletcher, G.H.L., Lemay, A., Advokaat, N.: gMark: schema-driven generation of graphs and queries. IEEE TKDE **29**(4), 856–869 (2017)
5. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: FOCS, pp. 394–403 (1997)
6. Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-preserving symmetric encryption. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 224–241. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_13
7. Bonifati, A., Martens, W., Timm, T.: An analytical study of large SPARQL query logs. VLDB J. **29**(2), 655–679 (2020)
8. Curino, C., et al.: Relational cloud: a database service for the cloud. In: CIDR, pp. 235–240 (2011)
9. Delanaux, R., Bonifati, A., Rousset, M.-C., Thion, R.: Query-based linked data anonymization. In: Vrandečić, D., et al. (eds.) ISWC 2018. LNCS, vol. 11136, pp. 530–546. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00671-6_31
10. Fernández, J.D., Kirrane, S., Polleres, A., Steyskal, S.: Self-enforcing access control for encrypted RDF. In: Blomqvist, E., Maynard, D., Gangemi, A., Hoekstra, R., Hitzler, P., Hartig, O. (eds.) ESWC 2017. LNCS, vol. 10249, pp. 607–622. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58068-5_37
11. Fernández, J., Kirrane, S., Polleres, A., Steyskal, S.: HDT$_{crypt}$: compression and encryption of RDF datasets. Semant. Web J. (2018)
12. Giereth, M.: On partial encryption of RDF-graphs. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 308–322. Springer, Heidelberg (2005). https://doi.org/10.1007/11574620_24
13. Goldreich, O.: The Foundations of Cryptography - Volume 2: Basic Applications. Cambridge University Press, Cambridge (2004)
14. Kasten, A., Scherp, A., Armknecht, F., Krause, M.: Towards search on encrypted graph data. In: PrivOn@ISWC (2013)
15. Kirrane, S., Abdelrahman, A., Mileo, A., Decker, S.: Secure manipulation of linked data. In: Alani, H., et al. (eds.) ISWC 2013. LNCS, vol. 8218, pp. 248–263. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41335-3_16
16. Kirrane, S., Villata, S., d'Aquin, M.: Privacy, security and policies: a review of problems and solutions with semantic web technologies. Semant. Web **9**(2), 153–161 (2018)
17. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_16
18. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: CryptDB: protecting confidentiality with encrypted query processing. In: SOSP, pp. 85–100 (2011)