S4: A New Secure Scheme for Enforcing Privacy in Cloud Data Warehouses

Somayeh Sobati Moghadam, Jérôme Darmont, Gérald Gavin

S4: A New Secure Scheme for Enforcing Privacy in Cloud Data Warehouses

Somayeh Sobati Moghadam, Jérôme Darmont, and Gérald Gavin

Université de Lyon, Lyon 2, Lyon 1, ERIC EA3083
5 avenue Pierre Mendès France – 69676 Bron Cedex – France
ssobati@eric.univ-lyon2.fr, jerome.darmont@univ-lyon2.fr,
gerald.gavin@univ-lyon1.fr

Abstract. Outsourcing data into the cloud becomes popular thanks to the pay-as-you-go paradigm. However, such practice raises privacy concerns. The conventional way to achieve data privacy is to encrypt sensitive data before outsourcing. When data are encrypted, a trade-off must be achieved between security and efficient query processing. Existing solutions that adopt multiple encryption schemes induce a heavy overhead in terms of data storage and query performance, and are not suited for cloud data warehouses. In this paper, we propose an efficient additive encryption scheme (S4) based on Shamir's secret sharing for securing data warehouses in the cloud. S4 addresses the shortcomings of existing approaches by reducing overhead while still enforcing good data privacy. Experimental results show the efficiency of S4 in terms of computation and storage overhead with respect to existing solutions.

1 Introduction

Data warehouses (DWs) provide a consolidated view of organizations and businesses' data, optimized for reporting and analysis. greatly enhance decision making. DWs consolidate historical data from different sources and allow on-line analytical processing (OLAP). Nowadays, data outsourcing scenarios tremendously grow with the advent of cloud computing that offers both cost savings and service benefits. One of the most notable cloud outsourcing services is Database-as-a-Service, where individuals and organizations outsource data storage and management to a Cloud Service Provider (CSP) [19]. Naturally, such services allow outsourcing a DW and running OLAP queries [1]. Yet, data outsourcing brings out privacy concerns since sensitive data are stored, maintained and processed by an external third party that may not be fully trusted.

A typical solution to preserve data privacy is encrypting data locally before sending them to an external server. Secure database management systems (SDBMSs) such as CryptDB [13] implement cryptographic schemes. Paillier's partially homomorphic encryption scheme [12] is notably used in CryptDB to provide high security. However, it induces a high storage and computation overhead. Hence, in this paper, we propose a new Secure Secret Splitting Scheme (S4) that aims at replacing Paillier's scheme in systems such as CryptDB. S4 is

based on the idea of secret sharing [16] and is efficient both in terms of storage and computing, without sacrificing privacy too much.

In the remainder of this paper, Section 2 discusses related works about SDBMSs, homomorphic encryption and secret sharing. Section 3 details and discusses S4. Section 4 provides an experimental validation of S4 against Paillier's scheme. Finally, section 5 concludes the paper and hints as future research.

2 Related Works

2.1 Secure Database Management Systems

CryptDB brings together powerful cryptographic tools to handle query processing on encrypted data without decryption [13]. Encryption in CryptDB is like onion layers that store multiple ciphertexts, i.e., encrypted data, within each other. Each onion layer enables certain kind of query processing and a given security level provided by one encryption scheme. For instance, order-preserving encryption (OPE) enables range queries and additive homomorphic encryption enables addition over encrypted data. Yet, CryptDB is not perfectly secure since schemes such as OPE reveal some statistical information about plaintext [11].

MONOMI builds upon CryptDB to allow the execution of analytical workloads over encrypted data outsourced to the cloud [18]. MONOMI aims at improving CryptDB's query processing capability and efficiency based on split client/server execution. A designer also optimizes physical data layout.

Eventually, using a local trusted hardware at the CSP's, such as TrustedDB [3] and CipherBase [2], is an alternative approach to query encrypted data. However, trusted hardware is limited in computation ability and memory capacity, and also very expensive.

2.2 Homomorphic Encryption

Fully homomorphic encryption (FHE) allows performing arbitrary arithmetic operations over encrypted data without decryption [7]. FHE provides semantic security, i.e., it is computationally impossible to distinguish two ciphertexts encrypted from the same plaintext. However, FHE requires so much computing power that it cannot be used in practice.

Partially homomorphic encryption (PHE) is more efficient than FHE. Paillier's [12] the most efficient additive FHE. With Paillier's scheme, multiplying the encryption of two values results in an encryption of the sum of the values, i.e., $Enc_k(x) \times Enc_k(y) = Enc_k(x + y)$, where the multiplication is performed modulo some public-key k [13]. Paillier's scheme is, however, still computationally intensive and induces as large ciphertext sizes as 2048 bits. Additionally, modular multiplications become computationally expensive on a large number of records, such as in the fact table of a DW [18, 17].

2.3 Secret Sharing

Secret sharing divides a secret piece of data into so-called shares that are stored at n participants'. A subset of $k \leq n$ participants is required to reconstruct the secret. In Shamir's, the first secret sharing scheme [16], to share a secret v_j, a random polynomial $P_{v_j}(x)$ of degree $k-1$ is first built. The owner of the secret chooses a prime $p > v_j$ and $k-1$ random numbers $a_1, a_2, ..., a_{k-1}$ from $\mathbb{F}_p$; and sets $a_0 = v_j$ (Equation 1). $P_{v_j}(x)$ passes through the point $(0, v_j)$.

$$P_{v_j}(x) = a_{k-1}x^{k-1} + ... + a_1 x + a_0 \qquad \text{mod } p \tag{1}$$

To build n points over $P_{v_j}(x)$, a set of n distinct elements in $\mathbb{F}_p$, $X = \{\mathtt{x_1}, \mathtt{x_2}, \ldots, \mathtt{x_n}\}$, is chosen such that $\mathtt{x_i} \neq 0 \ \forall i = 1, ..., n$. For each participant i, the corresponding share is $v_{i,j} = P_{v_j}(\mathtt{x_i})$. For each secret v_j, there are n points $(\mathtt{x_i}, v_{i,j})$ through which the polynomial $P_{v_j}(x)$ passes [8]. Any k shares form k points $(\mathtt{x_i}, v_{i,j})$ $i = 1, \ldots k$, from which polynomial $P_{v_j}(x)$ can be reconstructed using Lagrange interpolation [5] (Equation 2).

$$P_{v_j}(x) = \sum_{i=1}^{k} v_{i,j}\ell_i(x) \qquad \text{mod } p \tag{2}$$

$$\ell_i(x) = \prod_{1 \leq j \leq k, j \neq i} (x - \mathtt{x_j})(\mathtt{x_i} - \mathtt{x_j})^{-1} \qquad \text{mod } p$$

where $(\mathtt{x_i} - \mathtt{x_j})^{-1}$ is the multiplicative inverse of $(\mathtt{x_i} - \mathtt{x_j})$ modulo p [5]. Eventually, the secret is the constant term of the polynomial:

$$v_j = P_{v_j}(0) = \sum_{i=1}^{k} v_{i,j}\ell_i(0) \qquad \text{mod } p. \tag{3}$$

3 S4

S4's driving idea is based on secret sharing, but instead of sharing secrets to n participants' or CSPs', they are stored at one single CSP's. Thus, we avoid the high storage overhead of secret sharing. In S4, each secret v_j is divided into $n = k$ splits $v_{1,j}, ..., v_{k,j}$. $k-1$ splits, $v_{1,j}, ..., v_{k-1,j}$, are stored at the CSP's and $v_{k,j}$ is stored in a trusted machine, e.g., at the user's (Figure 1). In order to reduce storage overhead at the user's, $v_{k,j}$ is set to be the same for all secrets.

3.1 Splitting and Reconstruction Processes

First, $\mathtt{x_k}$ and v_k are randomly set up from $\mathbb{F}_p$, where p is a big prime number, i.e., greater than the greatest possible query answer. For any secret v_j, a random polynomial $P_{v_j}(x)$ is built that passes through $(0, v_j)$ and $(\mathtt{x_k}, v_k)$. To this end,

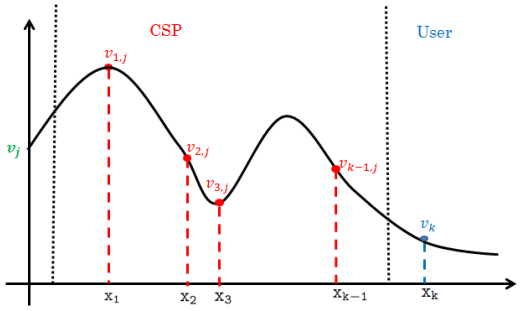

Fig. 1: S4 secret splitting

$k-2$ points $(a_i, b_i), i = 1, ..., k-2$ are chosen randomly from $\mathbb{F}_p$ such that $a_i \neq \mathtt{x_k}$ and $a_i \neq 0 \; \forall i = 1, ..., k-2$. Given k points $(a_1, b_1), (a_2, b_2), ..., (a_{k-2}, b_{k-2}), (0, v_j)$ and $(\mathtt{x_k}, v_k)$, polynomial $P_{v_j}(x)$ is built using Equation 2. Storing the $k-2$ random points is unnecessary because they are not needed for secret reconstruction.

To divide v_j into $k-1$ splits (since $(\mathtt{x_k}, v_k)$ is already fixed), a set of $k-1$ distinct elements $X = \{\mathtt{x_1}, \mathtt{x_2}, \ldots, \mathtt{x_{k-1}}\}$ is chosen from $\mathbb{F}_p$ such that $\mathtt{x_i} \neq 0$ and $\mathtt{x_i} \neq \mathtt{x_k} \; \forall i = 1, ..., k-1$. Then, splits are $v_{i,j} = P_{v_j}(\mathtt{x_i})$. $\mathcal{K}$=$(X, (\mathtt{x_k}, v_k))$ is considered as a *private key* for S4 and must be kept hidden from the CSP. To reconstruct secret v_j, its $k-1$ splits must be retrieved from the CSP. Given points $(\mathtt{x_i}, v_{i,j})$, $i = 1, ..., k-1$ and $(\mathtt{x_k}, v_k)$, which is stored at the user's, polynomial $P_{v_j}(x)$ can be reconstructed using Equation 2. Its constant term is v_j.

3.2 Summation Queries

Let a relational table T consist of one attribute A (additional attributes, if any, can be processed similarly). Suppose T has m records. We denote by v_j the j^{th} value of A. For attribute A in T, $k-1$ attributes A_i, $i = 1, ..., k-1$ are created in table T' at the CSP's, where each attribute A_i stores the i^{th} splits. Without loss of generality, we assume integer data type for A. Other data types can be transformed into integers before splitting. S4 allows summation queries to be computed directly at the CSP's. Consider a query that sums q values of A.

$$\mathtt{SUM=} \sum_{1 \leq j \leq q} v_j, \qquad v_j \in dom(A) \quad \forall j = 1, ..., q.$$

The CSP computes the sum of the splits stored in A_i as $\mathtt{SUM}_i \quad \forall i = 1, \ldots k-1$ such that

$$\mathtt{SUM}_i = \sum_{1 \leq j \leq q, 1 \leq i \leq k-1} v_{i,j} \qquad \text{mod } p.$$

Then, $\mathtt{SUM}_1$, $\mathtt{SUM}_2$,..., $\mathtt{SUM}_{k-1}$ are shipped back to the user and polynomial $P_{\mathtt{SUM}}(x)$ is built using Equation 2 using k points

$$(\mathtt{x_i}, \mathtt{SUM}_i)_{i=1,...,k-1}, \quad (\mathtt{x_k}, \sum_{j=1}^{q} v_k = q \times v_k).$$

$P_{\mathtt{SUM}}(x)$'s constant term is `SUM`. S4 does not alter the number of records. Hence, `COUNT` queries can be processed normally, thus also allowing `AVG` queries.

3.3 Security Analysis

Paillier's PHE is semantically secure, but it is too expensive in terms of ciphertext storage space and query response time. S4 proposes a classical trade-off with a lower level of security, but better storage and response time efficiency [16]. Let us consider a scenario where the CSP is said honest but curious, which is a widely used adversary model for cloud data outsourcing [15]. Such a CSP faithfully complies to any service-level agreement and, in our particular case, stores data, runs queries and provides results without alteration, malicious or otherwise. Yet, the CSP may access data and infer information from queries and results.

Privacy in S4 relies on the fact that a secret value is only retrievable by the user via private key $\mathcal{K}$. As in secret sharing, it is indeed guaranteed that at least k splits and X are necessary to reconstruct a secret, while the CSP has access to only $k-1$ splits. Both X and the k^{th} split, i.e., $\mathcal{K}$, are stored at the user's. However, the CSP still has access to linear combinations of splits (Equation 2), which provide some information. Still, the higher k is, the more difficult it is to interpret linear combinations of splits. Thus, k is the prime security parameter in S4. Experiments in Section 4 provide hints for choosing k.

Moreover, if some secrets are known by the CSP, e.g., through public communication of a company to its shareholders, solving Equation 3 becomes possible. For example, if the CSP knows secrets $v_1, ..., v_{k-1}$. Also knowing the corresponding splits $v_{1,j}, ..., v_{k-1,j}$ $\forall j \in [1, k-1]$, the CSP can recover the Lagrange basis polynomials $\ell_i(0)$ $\forall i \in [1, k]$ and solve Equation 3 to recover all secrets. However, the CSP must know at least $k-1$ secrets to do so. Moreover, we also propose leads to address this problem in Section 5.

4 Experimental Evaluation

4.1 Experimental Setup

We implement S4 in C using compiler gcc 4.8.2. S4's source code is freely available on-line[1]. Experiments related to Paillier's PHE exploit the libpaillier standard C library [4]. All mathematical computations use the GNU Multiple Precision Arithmetic Library (GMP) [6]. Eventually, we conduct our experiments on an Intel Core i7 3.10 GHz PC with 16 GB of RAM running Linux Ubuntu 15.05.

[1] http://eric.univ-lyon2.fr/download/libS4.zip

We compare S4 and Paillier's PHE using simple synthetic datasets, i.e., 32-bit unsigned integers generated uniformly at random from the integer range $[10^3, 10^4[$. We scale up the number of records m such that $m \in (10^3, 10^4, 10^5, 10^6)$, forming four distinct datasets.

In S4, we vary k from 8 to 64, higher values of k inducing too long execution times. Prime p must be greater than the greatest query answer, e.g., $p > \sum_{j=1}^{m} v_j$. In Paillier's PHE, we use a key size of 1024 bits, which induces ciphertexts of 2048 bits. Such key size is the absolute minimum to achieve security [9, 20].

4.2 Encryption and Decryption Time

Figure 2 plots the time of secret splitting in S4 and secret encryption in Paillier's scheme with respect to m. It shows that encryption time in S4 is lower than Paillier's when $k \leq 16$, and then becomes higher when $k \geq 16$. Secret splitting consists in building a random polynomial by randomly choosing $k-2$ points. Hence, splitting time increases with k. Figure 2 actually illustrates the tradeoff between S4's security and encryption efficiency with respect to Paillier's PHE.

Figure 3 plots the time of secret reconstruction in S4 and secret decryption in Paillier's scheme with respect to m. With the selected values of k, decryption is faster with S4 than with Paillier's PHE. This is mainly because Paillier's scheme needs m expensive modular multiplications of large, 2048-bit numbers for decryption, while secret reconstruction in S4 works by polynomial interpolation over k points and evaluating the polynomial in one single point.

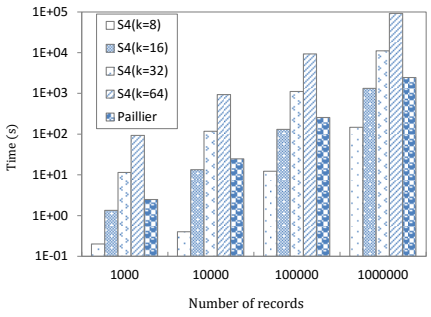

Fig. 2: Splitting/encryption time

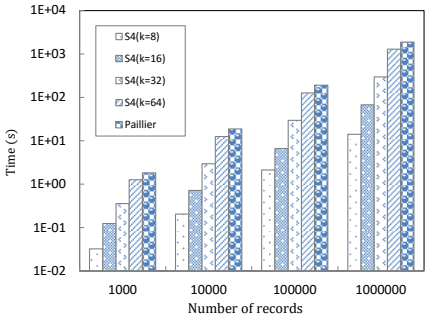

Fig. 3: Reconstruction/decryption time

4.3 Space Overhead

Figure 4 plots the storage required by S4 and Paillier's PHE with respect to m. With the selected values of k, S4's storage overhead is always much smaller than that of Paillier's PHE since Figure 4's y axis follows a logarithmic scale. Paillier's scheme indeed produces 2048-bit ciphertexts. Thus, its storage overhead is $m \times 2048$. With S4, each value is split into $k-1$ values. Thus, S4's storage overhead is $m \times (k-1)$ times plaintext size.

4.4 Query Processing Time

Figure 5 plots summation query processing times over all records in each dataset, for both S4 and Paillier's PHE, with respect to m. It shows that, with the selected values of k, query execution time in S4 is lower than that of Paillier's scheme. This is because Paillier's scheme requires m expensive modular multiplications to compute a sum, while S4 computes only $(k-1) \times m$ simple modular additions.

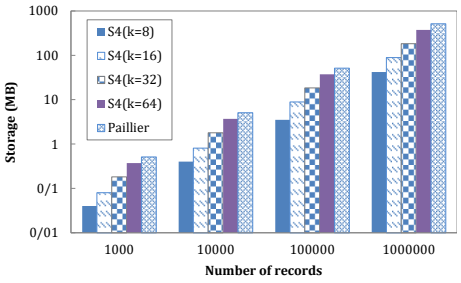

Fig. 4: Storage overhead

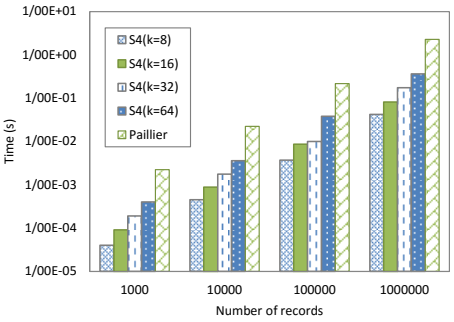

Fig. 5: Summation execution time

5 Conclusion

In this paper, we introduce S4, a new cryptographic scheme that supports summation queries in cloud-based OLAP. We experimentally show that S4 is much more efficient than Paillier's PHE in terms of query response time and space overhead. Thus, replacing Paillier's scheme with S4 in secure DBMSs such as CryptDB and MONOMI can improve analytical query processing in cloud DWs. Moreover, we also plan a variant of S4 for computing multiplications.

However, we achieve performance gains through a slight degradation of security, especially when an adversary has knowledge of secret values. Although it is definitely acceptable in some cloud DW and OLAP scenarios, e.g., public aggregate data might not actually yield secrets, i.e., fine-grained data, we will devote future research to strengthen S4 against such threats. More precisely, we plan to introduce noise, as in many cryptographic problems such as approximate-GCD [10] or LWE [14]. For instance, instead of sharing v_j, we could share $10^r \times v_j + noise$. By doing so, security is intuitively enhanced while the whole process remains correct, provided r is sufficiently large and $noise$ sufficiently small.

References

1. Amanatidis, G., Boldyreva, A., O'Neill, A.: Provably-secure schemes for basic query support in outsourced databases. In: 21st IFIP WG 11.3 Working Conference on Data and Applications Security, Redondo Beach, CA, USA. pp. 14–30 (2007)

2. Arasu, A., Eguro, K., Joglekar, M., Kaushik, R., Kossmann, D., Ramamurthy, R.: Transaction processing on confidential data using cipherbase. In: 31st IEEE International Conference on Data Engineering, ICDE, Seoul, South Korea. pp. 435–446 (2015)
3. Bajaj, S., Sion, R.: TrustedDB: a trusted hardware based database with privacy and data confidentiality. In: International Conference on Management of Data, SIGMOD, Athens, Greece. pp. 205–216 (2011)
4. Bethencourt, J.: Paillier library. http://acsc.cs.utexas.edu/libpaillier/ (last accessed: 2016)
5. Dautrich, J.L., Ravishankar, C.V.: Security limitations of using secret sharing for data outsourcing. In: 26th IFIP WG 11.3 Conference in Data and Applications Security and Privacy, Paris, France. pp. 145–160 (2012)
6. Free Software Foundation: GNU Multiple Precision Arithmetic library . https://gmplib.org/ (last accessed: 2016)
7. Gentry, C.: A fully homomorphic encryption scheme. Ph.D. thesis, Stanford University (2009)
8. Hadavi, M.A., Jalili, R., Damiani, E., Cimato, S.: Security and searchability in secret sharing-based data outsourcing. International Journal of Information Security 14(6), 513–529 (2015)
9. Jost, C., Lam, H., Maximov, A., Smeets, B.J.M.: Encryption Performance Improvements of the Paillier Cryptosystem. IACR Cryptology ePrint Archive 864 (2015), http://eprint.iacr.org/2015/864
10. Liu, D.: Practical Fully Homomorphic Encryption without Noise Reduction. IACR Cryptology ePrint Archive 468 (2015), http://eprint.iacr.org/2015/468
11. Naveed, M., Kamara, S., Wright, C.V.: Inference attacks on property-preserving encrypted databases. In: 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA. pp. 644–655 (2015)
12. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International Conference on the Theory and Application of Cryptographic Techniques, EUROCRYPT, Prague, Czech Republic. LNCS, vol. 1592, pp. 223–238 (1999)
13. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: CryptDB: protecting confidentiality with encrypted query processing. In: 23rd ACM Symposium on Operating Systems Principles, SOSP, Cascais, Portugal. pp. 85–100 (2011)
14. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM 56(6) (2009)
15. Sanamrad, T., Kossmann, D.: Query log attack on encrypted databases. In: 10th VLDB Workshop on Secure Data Management, SDM, Trento, Italy. pp. 95–107 (2013)
16. Shamir, A.: How to share a secret. Communications of the ACM 22(11), 612–613 (1979)
17. Sion, R.: Towards Secure Data Outsourcing, pp. 137–161. Springer (2008)
18. Tu, S., Kaashoek, M.F., Madden, S., Zeldovich, N.: Processing analytical queries over encrypted data. Proceedings of the VLDB Endowment 6(5), 289–300 (2013)
19. Xiong, L., Chitti, S., Liu, L.: Preserving data privacy in outsourcing data aggregation services. ACM Transactions on Internet Technology 7(3) (2007)
20. Yildizli, C., Pedersen, T.B., Saygin, Y., Savas, E., Levi, A.: Distributed privacy preserving clustering via homomorphic secret sharing and its application to (vertically) partitioned spatio-temporal data. International Journal of Data Warehousing and Mining 7(1), 46–66 (2011)