CryptZip: Squeezing out the Redundancy in Homomorphically Encrypted Backup Data

Yiwen Shao, Sa Wang, Yungang Bao
SKL Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China
{shaoyiwen,wangsa,baoyg}@ict.ac.cn

ABSTRACT

Ideal homomorphic encryption is theoretically achievable but impractical in reality due to tremendous computing overhead. Homomorphically encrypted databases, such as CryptDB, leverage replication with partially homomorphic encryption schemes to support different SQL queries over encrypted data directly. These databases reach a balance between security and efficiency, but incur considerable storage overhead, especially when making backups. Unfortunately, general data compression techniques exhibit inefficiency on encrypted data. We present CryptZip, a backup and recovery system that could highly reduce the backup storage cost of encrypted databases. The key idea is to leverage the metadata information of encryption schemes and selectively backup one or several columns among semantically redundant columns. The experimental results show that CryptZip could reduce up to 90.5% backup storage cost on TPC-C benchmark.

ACM Reference format:
Yiwen Shao, Sa Wang, Yungang Bao. 2018. CryptZip: Squeezing out the Redundancy in Homomorphically Encrypted Backup Data. In *Proceedings of 9th Asia-Pacific Workshop on Systems, Jeju Island, Republic of Korea, August 27–28, 2018 (APSys '18),* 8 pages.
https://doi.org/10.1145/3265723.3265726

1 INTRODUCTION

Applications on cloud today subject to much more threats than ever. Users' confidential data in back-end databases on cloud could be unauthorizedly accessed by hackers, adversaries and even malicious database administrators in various ways. To prevent data leaks, an intuitive alternative is to encrypt sensitive data in source databases. However, it will disable SQL queries executing on databases. Fortunately, with the help of homomorphic encryption schemes, CryptDB [24] was proposed to support SQL-queries on encrypted data directly and achieve both efficiency and confidentiality. CryptDB attracted much attention of industry. Many related systems have been implemented since then by companies like Google, SAP, Microsoft, and some startups [1]. In academia, many CryptDB-like systems have also been proposed [10, 11, 17–19, 22, 29].

Fully homomorphic encryption allows arbitrary computation on encrypted data, but incurs tremendous computing overhead. CryptDB instead leverages a collection of efficient *partially* homomorphic encryption schemes, that each encryption scheme can only support one kind of operations, such as equality checks, order comparisons, aggregates and etc.. To enable more operations, CryptDB stores multiple ciphertexts of each data item encrypted by these schemes in databases. Thus, each column is transformed into several replicas columns of ciphertexts in CryptDB.

Such data layout is essentially complementary to partially homomorphic encryption schemes to support arbitrary queries over encrypted data, but incurs prohibitive storage cost at the same time. As illustrated in Table 1, the storage cost will expand to 21.3x when making backups for all the encrypted TPC-C tables using MySQLDump. Today's database systems usually need to backup their data in DBMSes periodically and recover from crash immediately. For important data, they even have to maintain several copies of backup data in distributed data centers in case of unexpected disasters, which will further amplify the storage overhead incurred by CryptDB-like systems.

Unfortunately, it is quite inefficient to eliminate redundant encrypted data using data compression techniques (only 15% encrypted backup data of TPC-C are reduced by Gzip as illustrated in Table 1). For the sake of security, it is nearly unacceptable to decrypt and compress the plaintexts in databases, which will make the backup process vulnerable to attack.

APSys '18, August 27–28, 2018, Jeju Island, Republic of Korea
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6006-7/18/08…$15.00
https://doi.org/10.1145/3265723.3265726

TPC-C Benchmark	MySQLDump (Plaintext)	MySQLDump (Encrypted Data)	MySQLDump (Compressing)	CryptZip (Column Dedup)	CryptZip (Dedup & Compressing)
Storage Size	79M	1680M	1432M	160M	79M

Table 1: MySQLDump produces 79MB backup files for plaintext data and 1680MB for encrypted data, introducing 21.3x storage cost. Using Gzip to compress the encrypted data only reduces the storage cost to 1432MB. CryptZip is able to extract 160MB data for backup, which can be further compressed to 79MB.

Thus the ideal alternative is to apply the compression techniques on the encrypted data directly. However, the efficiency of compression techniques heavily rely on data similarity, whereas cryptography aims to make ciphertexts indistinguishable from random data. Thus, the efficiency of compressing ciphertexts decreases heavily.

Therefore in this paper we ask the following question: *can we significantly reduce the storage overhead of encrypted backup data?* Meanwhile, since we still have to recover all the duplicated data from crash, the duration of recovery is also important to DBMSes. Intuitively, more redundant data are reduced, longer duration of recovery is needed. Thus, the backup strategy need to be carefully designed to trade off between storage cost and recovery time.

We present **CryptZip**, an efficient backup and recovery system for homomorphically encrypted databases that leverages metadata information to significantly reduce the backup storage of encrypted data. We observe that CryptDB-like systems usually maintain metadata tables to record the encryption schemes of each duplicated column. The metadata information will indicate which encrypted columns essentially store the same data items. Intuitively, with metadata information, we can selectively backup one or a few columns among these semantically redundant columns without decryption and highly reduce the storage overhead.

However, since CryptDB-like systems incorporate a collection of partially homomorphic encryption schemes (DET, OPE, HOM and etc. in §2.2), the key challenge becomes how to choose the *right* encrypted columns encrypted by different schemes for backup and recovery. There are at least three aspects that should be considered: 1) the original table could be recovered from the selected encrypted columns correctly and completely. 2) The storage overhead incurred by these encrypted columns should be relatively small. 3) The time consumed by recovering from these encrypted columns should be relatively short.

Considering the above requirements, we first study the characteristics of state-of-the-art partially homomorphic encryption schemes on computation and storage overhead as well as encrypted data integrity. Then we design a min-space model to help operators explore the backup strategy with minimum storage cost under certain recovery time requirement. We further propose three backup strategies (space-optimal strategy, time-optimal strategy and balanced strategy) with state-of-the-art encryption schemes. To evaluate the efficiency of CryptZip, we conduct experiments towards two micro-benchmarks and TPC-C benchmark. The experimental results show that the space-optimal strategy could reduce 70%~95% storage cost on micro-benchmarks and 83%~90.5% on TPC-C benchmark. The balanced strategy makes a tradeoff between storage cost and recovery time that reduces 18%~67% storage cost on micro-benchmarks with extra recovery time of less than 3 minutes, reduces 55% storage cost on TPC-C benchmark with extra recovery time of only 5 minutes .

2 BACKGROUND

2.1 Computing On Encrypted data

Homomorphic encryption is a technique allowing computing on encrypted data. This notion was proposed by Ron Rivest [26] in 1978. The first fully homomorphic encryption(FHE) construction was proposed by Gentry [12]. Fully homomorphic encryption allows arbitrary computation over encrypted data but the efficiency is low. The evaluation of AES using FHE was reported to be more than six orders of magnitude than that in plaintext [23]. Therefore, FHE is far from practical.

In contrast, partially homomorphic encryption allows limited number of operations on ciphertexts with higher efficiency. RSA [27] and El Gamal [9] support multiplication on ciphertexts. Pailliar [21] supports addition on ciphertexts. Order preserving encryption [4, 6, 7] preserves the order of plaintexts, thus enabling comparing order directly on ciphertexts. Searching on encrypted strings is also supported by several encryption schemes [8, 15, 28]. Those algorithms are able to be adopted in production due to their high efficiency.

2.2 CryptDB

CryptDB is the first practical database system that uses a collection of partially homomorphic encryption schemes to support executing SQL queries directly on encrypted data. CryptDB leverages multi-onions encryption schemes to encrypt relational tables and a trusted proxy for transparent encryption and decryption.

Architecture. Figure 1(❶) shows the architecture of CryptDB. CryptDB puts a proxy between the web server and the DBMS. The proxy server maintains the metadata and encryption keys of the DBMS. Metadata records the encryption details of tables in the DBMS. The web server interacts with the proxy as if querying normal databases and the database

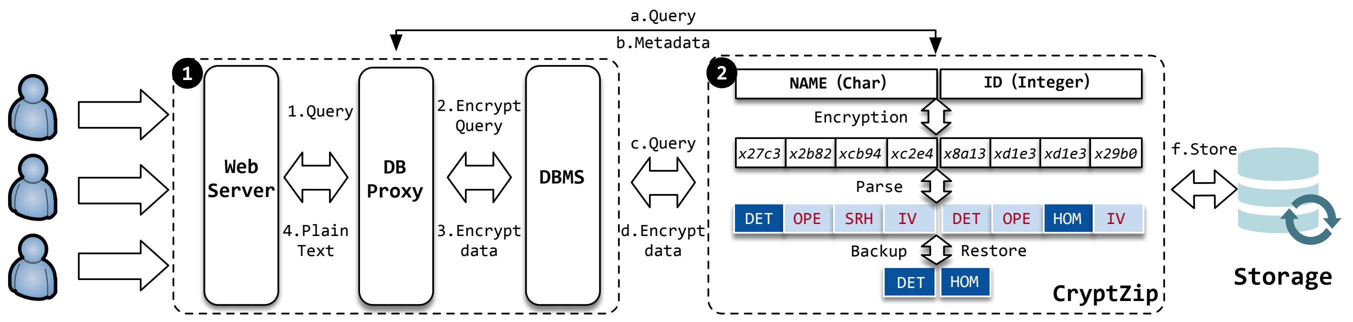


Figure 1: ❶The architecture of CryptDB-like system with three layers: a web server, a database proxy and a DBMS. They put a proxy server between the web server and the DBMS server which intercepts all SQL queries and rewrite them to execute on encrypted data. The results are sent back to the proxy server first for decryption, and then plaintexts will be sent to web server. ❷ The backup and recovery of CryptZip that leverages a backup client to get metadata from proxy servers, parses data, backup and recovery.

executes encrypted queries on encrypted data directly. Both the web server and the database do not need to be changed.

Multi-Onions Encryption. Each partially homomorphic encryption scheme supports one kind of operations. For example, Pailliar supports addition but disallows order comparison. Similarly, order preserving encryption schemes do not support addition instead. If both addition and order comparison are needed for the data, neither scheme alone is able to provide both. CryptDB addresses this problem by adopting data replication and multi-onions encryption. The original plaintext is replicated several times and each replica is encrypted using different onions. Onion **DET** [25] supports equality comparison; Onion **OPE** [6] allows order comparison; **SRH** [28] allows searching on ciphertexts; **HOM** [21] and **ASHE** [22] allows addition on ciphertexts. When a specific operation on ciphertexts is needed, the corresponding encrypted replica supporting that operation is queried. As shown in Figure 1(❷), each column in the plaintext table corresponds to three different encrypted replicas of onions and an IV column.

Each onion consists of one or more layers to improve security. For example, onion OPE uses order preserving encryption scheme which is considered insecure. Adding another random encryption layer on this onion improves security level [24]. To enable random encryption, the extra column IV containing random numbers are added. The random layer does not support order comparing any more, therefore, this layer is decrypted when order comparing is needed for the corresponding column.

2.3 Backup

Typical database backup falls into two categories based on the format of the backup files [2]: logical backup and physical backup.

Physical backup consists of raw copies of directories and files that store the database contents. In fact, simple commands like *cp* can be used as physical backup method. Popular physical backup tools includes ibbackup and XtraBackup [3]. MySQL stores data and metadata in a set of files in a user configured directory.

Logical backup saves information in the form of SQL queries. For Logical backup, it's easy to control the backup granularity, and it's highly portable since the backup is in text files containing data or SQL queries. MySQLDump and MySQLdumper are examples of Logical backup tools. We can also use SELECT .. INTO OUTFILE to create delimited-text files for logical backup. The basic process of logical backup is first to use select SQL queries to pull data from the MySQL-Server, and then save those data into text files.

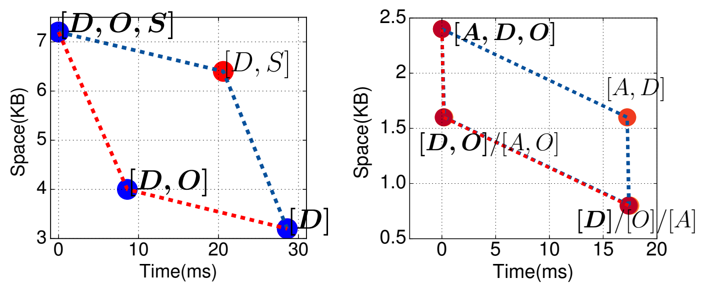
In this paper, we choose logical backup due to following considerations: 1) The portability of logical backup can make our method applied to a wide range of systems supporting SQL. 2) Logical backup enables us to find duplicated columns directly using SQL queries and further reduces storage overhead.

3 CRYPTZIP DESIGN

3.1 Overview

In CryptDB, each column is replicated several times and each replica of the original column is encrypted its corresponding onion. Therefore, the intuitive idea is to store only one encrypted replica (onion) and remove the others. To achieve this goal, we propose CryptZip's backup and recovery mechanism, as depicted in Figure 1(❷).

The backup process consists of two steps: parsing metadata and storing encrypted columns. Parsing metadata enables CryptZip to find the mapping from each plaintext column to its encrypted replicas. For example, the table in Figure 1(❷) consists of two columns, NAME and ID. Each column is replicated and encrypted using multiple onions, each



(a) Strings (b) Integers

Figure 2: The time and space overhead of different backup strategies: (a) strings with (D,O,S) and (b) integers with (D,O,A).

Table 2: Properities of onion encryption schemes. Onion DET and OPE for strings have properities different from that of integers. Onions that are not able to be decrypted can not be used as recovery onion.

Encryption Scheme	Data Type	Recovery Onion	Encrypt Time(us)	Decrypt Time(us)	Data Size (Byte)	Encrypt Rate (Byte/us)
DET		YES	1.9	1.33	8	4.21
OPE	Integer	YES	172.5	167.6	8	0.046
HOM		YES	5690	493	256	0.045
ASHE		YES	0.35	0.36	8	22.8
DET		YES	0-595.5	0-611.3	0-65535	0.009
OPE	String	NO	200.6	N/A	8	0.39
SEARCH		NO	10.1	N/A	16	1.58
IV	Integer /String	NO	0.7	N/A	8	11.4

of which owns an encrypted column name. The mapping from each plaintext column's name to names of its encrypted replicas is stored as metadata in the proxy. By querying the proxy, the mapping is revealed. After that, CryptZip is able to select one encrypted replica for data backup. As Figure 1(❷) depicts, onion DET is selected for column NAME, and onion HOM for ID. Once the selection is made, each encrypted column is fetched from the database and stored in its own file. In addition, a file called CryptMeta is also created to record the names of onions not selected in backup. This information is used in restoring process.

The data restoring process is the reverse of backup process. For each plaintext column, CryptMeta is used to find which encrypted replica of onions are selected for backup and which onions are not. Then the onion selected for backup, which is called the recovery onion, is decrypted and the onions not selected are restored by applying onion encryption. As in the example in Figure 1(❷), to restore the onions for the column NAME, onion DET is decrypted and the other onions are restored by onion encryption. For column ID, the process is similar.

3.2 Min-Space Analysis and Strategies

Intuitively, storing only one encrypted replica for each plaintext column can definitely reduce the storage overhead as much as possible. However, this strategy means all the other replicas needs to be restored by encryption, which incurs great time overhead. In addition, as is shown in Table 2, some onions like OPE for strings are not able to be decrypted, which means only using those onions for backup will prevent us from restoring data. As a result, it is a great challenge to choose a subset of more than one replicas may be selected for each plaintext column. For each plaintext column, the challenge of how to select a subset of its encrypted replicas needs to be addressed. We call the selection of this subset backup strategy and maintain that given a specified period of time, the strategies need to produce as little space overhead as possible. This is called Min-Space target in our analysis.

Overhead and Notations. In the analysis, we use the first characters D,O,A,H,S to represent onion DET, OPE, ASHE, HOM, Search respectively. In CryptDB, each column can be encrypted using a set of onions, which is denoted as O_A. The onions selected for backup is O_S. O_A is presented using parentheses, and brackets for O_S. For example, if an integer column is encrypted using onion DET, OPE, and HOM, and onion DET is selected for backup, then O_A and O_S are (D,O,H) and $[D,O]$ respectively. We use T_E, T_D, and s to denote the average encryption time, average decryption time and average size of an onion. Besides, we use value d to indicate whether an onion is able to be decrypted. If so, the value of d is 1, otherwise 0.

Once O_S is determined, the space overhead S_F and time overhead $Ext(O_S)$ for a column is obtained by Formula 1 and Formula 2 respectively. Meanwhile, O_S needs to satisfy the constraint in Formula 3 since at least one onion in the backup needs to be decrypted.

$$S_F = \sum_{O \in O_S} s \tag{1}$$

$$Ext(O_S) = \min_{O \in O_S} T_D + \sum_{O \in (O_A - O_S)} T_E \tag{2}$$

$$(\sum_{O \in O_S} d) \geq 1 \tag{3}$$

Min-Space Analysis. In CryptDB, the default O_A can be (D,O,H) or (D,O,A) for integers and (D,O,S) for strings. The random encryption layer discussed earlier is often peeled off to support operations, therefore, we do not backup the random number column IV, which is used to support random encryption.

Table 3: Backup Strategies

Backup Strategies	String	Integer	
	(D, O, S)	(D, O, H)	(D, O, A)
Space-Optimal Strategy	$[D]$	$[D]$	$[D]$
Balanced Strategy	$[D, O]$	$[D, H]$	$[D, O]$
Time-Optimal Strategy	$[D, O, S]$	$[D, H, O]$	$[D, O, A]$

For strings, only onion DET is able to be decrypted. Therefore, in order to satisfy the constraint in Formula 3, onion DET must be preserved in a backup, leaving us four backup strategies(O_S): $[D]$, $[D, O]$, $[D, S]$, and $[D, O, S]$.

Figure 2a shows time and space overhead of different strategies for backup. The overhead is obtained using Formula 1 and Formula 2. In Figure 2a, we assume that each column of onion contains one million lines of data, and it is allowed to backup a portion of lines of a column of data. The time for the point $[D, O, S]$ is 0 since all the onions are preserved, which means no extra time is needed for recovery. The point $[D]$ requires maximum time for recovery among the four strategies since two columns of onions need to be restored. From point $[D, O, S]$, the full backup, if we add Δt, then the user is able to delete several lines of either onion OPE or onion Search. If onion Search is first deleted, then onion OPE, we can obtain the red line in Figure 2a. Otherwise, we can obtain the blue line in Figure 2a.

If we look up from the x axis, the first point we meet is the min-space under the specified time. Therefore, all points on the red line in Figure 2a satisfy the min-space requirement. The three strategies on the red line are the three levels of strategies proposed for this data type: space-optimal strategy $[D]$, balanced strategy $[D, O]$ and time-optimal strategy $[D, O, S]$.

For integers, the default O_A can be either (D, O, A) and (D, O, H). Different from strings, all the onions for integers are able to be decrypted. Therefore, there are 7 strategies exist. For example, if O_A is (D, O, A), seven strategies exists, namely $[D]$, $[O]$, $[A]$, $[D, O]$, $[D, A]$, $[O, A]$, and $[D, O, A]$. Figure 2b shows the time and space overhead of the strategies when O_A is (D, O, A). The three levels of strategies can be proposed using similar methods.

Therefore, backup strategies for two data types under different O_A are summarized in Table 3.

4 EVALUATION

This section evaluates the storage cost and recovery time of three strategies of CryptZip proposed in §3.2. Meanwhile, we further analyze the breakdown of storage cost and recovery time of each encryption scheme.

Environment Setup. Our evaluations are conducted on two machines both with 8 Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz cores, 64GB memory and 1GB/s network running Ubuntu 16.04. We use MySQL 5.7 and MySQL Proxy 0.8.5 for CryptDB and Gzip 1.6 for data compressing.

Workloads. Our evaluations mainly use two benchmarks: 1) micro-benchmarks containing *one integer table* that contains only one 4 bytes integer column with 1,000,000 random integers and *two string tables* that each contains only one string column (16 bytes and 128 bytes) with 1,000,000 random strings. 2) TPC-C benchmark.

4.1 Micro-Benchmarks

Integers. We evaluate integers encrypted by (D, O, H) and (D, O, A). As illustrated in Table 4, for strategies with HOM, the space-optimal strategy reduces the storage overhead to 5%, but it introduces extra recovery time of more than 5,800 seconds[1]. When the time-optimal strategy reduces the recovery time to only 36%, the space overhead increases to 86%. Either time or space optimal strategy will cause the cost of the other side to soar, which makes it difficult to make tradeoffs between time and space including the balanced strategy. For strategies with ASHE, the balanced strategy can seek one good combination that storage cost and recovery time are both reduced (storage cost was reduced by 50% and time overhead was reduced by 20%).

We further break down the storage and time overhead as in Figure 3 (Integer), find out that HOM encryption scheme not only has a large storage overhead, but also cause long encryption and decryption time. When HOM is chose as a backup column, since it does not need to be encrypted at the time of recovery, the time cost is drastically reduced. But the storage overhead is huge. When HOM is removed from the backup solution, the storage overhead is drastically reduced, but the time for encryption recovery is increased. Thus, it is quite difficult to explore a balanced strategy to both reduce the storage cost and recovery time in the strategies with HOM. However, ASHE scheme has relatively small storage overhead and encryption time overhead. Therefore, *we recommend ASHE instead of HOM to support integer addition to save more storage and recovery time.*

Strings. We evaluate strings encrypted by (D, O, S). As discussed in §3.2, the size of onion OPE for strings is 8 Bytes while the size of onion DET and SEARCH depends on the number of key words. Therefore, we choose 16 bytes and 128 bytes to evaluate how the size of onions change with string length, and how this change affects the overhead of the three strategies.

[1]The + symbol in each table means the number plus the recovery time of MysqlDump is the whole recovery time of that strategy.

Table 4: Storage Cost and Recovery Time of Strategies for Integers

Backup Strategy	Strategies with HOM				Strategies with ASHE			
	Storage Cost	Storage Ratio	Recovery Time	Time Ratio	Storage Cost	Storage Ratio	Recovery Time	Time Ratio
MySQLDump	345MB	-	90s	-	65.9MB	-	19.9s	-
Space-Optimal Strategy	20MB	5%	+5866s	65x	19.5MB	30%	+132.8s	6.81x
Time-Optimal Strategy	297MB	86%	+16s	0.36x	44.7MB	68%	+12s	0.89x
Balanced Strategy	282MB	82%	+143s	2.45x	34.1MB	52%	+14.2s	0.8x

Table 5: Storage Cost and Recovery Time of Strategies for Strings

Backup Strategy	Strategies for String-16				Strategies for String-128			
	Storage Cost	Storage Ratio	Recovery Time	Time Ratio	Storage Cost	Storage Ratio	Recovery Time	Time Ratio
MySQLDump	59MB	-	17.9s	-	495MB	-	124s	-
Space-Optimal Strategy	16.2MB	27%	+94.3s	5.09x	143MB	29%	+316s	3.8x
Time-Optimal Strategy	42MB	71%	+6.4s	0.42x	415MB	84%	+17s	0.25x
Balanced Strategy	25.8MB	44%	+19.7s	1.17x	163MB	33%	+122s	1.48x

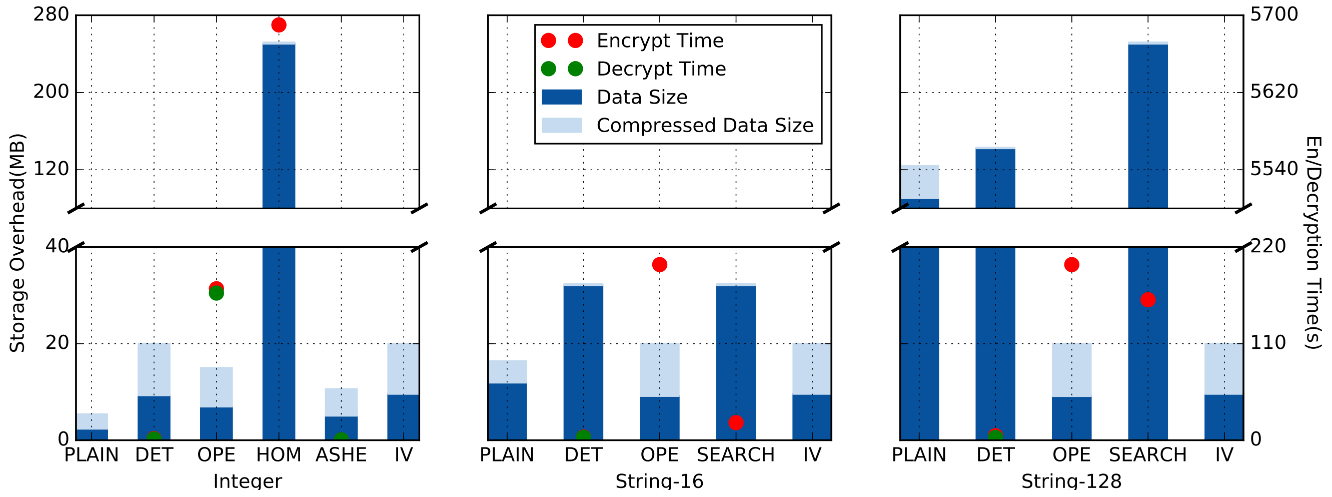


Figure 3: The four characteristics of each column encrypted using its onion: time for encryption, time for decryption, data size and compressed data size

Table 6: Storage Cost and Recovery Time of TPC-C benchmark

Backup Strategy	Strategies with HOM				Strategies with ASHE			
	Storage Cost	Storage Ratio	Recovery Time	Time Ratio	Storage Cost	Storage Ratio	Recovery Time	Time Ratio
MySQLDump	1680MB	-	244s	-	572MB	-	91s	-
Space-Optimal Strategy	160MB	9.5%	+26215s	107.4x	160MB	27%	+3556s	39.8x
Time-Optimal Strategy	1368MB	81.4%	+246s	1.008x	533MB	93%	+180s	1.97x
Balanced Strategy	1205MB	71.7%	+1031s	4.23x	259MB	45%	+327s	3.59x

As illustrated in Figure 3 (String-16 and String-128), the storage cost and encryption and decryption time of DET and SEARCH increases as the string length increases, except for OPE. The storage cost and recovery time of OPE stay the same whenever the string length grows. Therefore, *we recommend OPE should be preserved for long strings. Also, as strings grow longer, the relative storage cost becomes lower.*

Compression Ratio. We also illustrate the compression ratio of each encryption scheme in Figure 3. The light blue bar of each scheme is the original data size, and the dark blue is the data size after compressed by Gzip. For integers, except for HOM, the schemes all reach nearly 50% compression ratio. But for strings, except for OPE, the other two schemes both present small compression ratio.

4.2 TPC-C Benchmark

We evaluate the three backup strategies on encrypted TPC-C data. The backup of plaintext TPC-C data is 79MB. As illustrated in Table 1, for encryption schemes with HOM, the size of backup expands to 1680MB. Even with compression, the storage size is still 1432MB. In contrast, with space-optimal strategy, the storage size is reduced to 160MB. However, as discussed earlier, the corresponding time overhead drastically increase due to the encryption overhead of HOM. For TPC-C data encrypted with ASHE, the balanced strategy presents a better tradeoff between storage cost and recovery time, which reduces nearly 60% storage cost and introduces only 5 minutes recovery time.

5 RELATED WORK

Data compression. Data compression is an old technique to reduce storage size. It treats the data as byte stream and uses encoding algorithms to represent the data to be compressed [13]. Huffman Algorithm is one example. Others algorithms are also proposed in different areas, like Run-Length encoding and Lempel-Ziv encoding. The Gzip tool used in the experiments implements a variant of LZ77.

Data deduplication. Data deduplication [20] is another technique to reduce storage size in cloud environment. Those methods find either inter file or intra file redundancy and store only one copy of duplicated data. Both data compression and data deduplication are general purpose methods that are unaware of the underlying characteristics of data. In CryptDB, each replica of a column is encrypted using different algorithms and different encryption keys, which makes finding replicas difficult. Different from those methods, CryptZip leverages metadata to find replicas in higher level. Besides, each replica has different characteristics, which makes backup strategies necessary.

Encrypted deduplication. Message-locked encryption [5] and similar work are methods to solve the problem of secure deduplication. Those schemes are not the same as the partially homomorphic encryption schemes used in CryptDB, and therefore do not solve the problem in CryptDB.

6 CONCLUSION AND FUTURE WORK

CryptDB leverages data replication and onion encryption to support SQL queries on encrypted data. The replication and cryptographic expansion results in great space overhead. In this paper, we proposed CryptZip, which parses metadata of CryptDB to discover replicas encrypted using different onions. Using this extra information, we used min-space analysis to provide three levels of backup strategies. The experimental results show that CryptZip could reduce up to 90.5% backup storage cost on TPC-C benchmark.

In our future work, we plan to redesign the recovery process, which needs decrypt the backup data and encrypt the plaintext to restore the onions. This process will introduce extra security risks. We consider to leverage hardware encryption techniques like AMD Memory Encryption [16] and Intel SGX [14] to address this problem.

7 ACKNOWLEDGMENTS

First and foremost, we thank Professor Kai Li and Future Forum for inspiring us such an interesting idea. We would also like to thank Qun Huang, Dan Ding, Cheng Yang, Xusheng Zhan, Tianni Xu and the anonymous reviewers for their insightful suggestions. This work is supported in part by National Key R&D Program of China (No. 2016YFB1000201), and National Natural Science Foundation of China (No. 61420106013 and No. 61702480) and Youth Innovation Promotion Association of Chinese Academy of Sciences.

REFERENCES

[1] 2015. CryptDB. https://css.csail.mit.edu/cryptdb/. (2015).

[2] 2018. Backup and Recovery. (2018). https://dev.mysql.com/doc/refman/5.7/en/backup-and-recovery.html

[3] 2018. Xtrabackup. (2018). https://www.percona.com/software/mysql-database/percona-xtrabackup

[4] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. 2004. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. ACM, 563–574.

[5] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. 2013. Message-locked encryption and secure deduplication. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 296–312.

[6] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'neill. 2009. Order-preserving symmetric encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 224–241.

[7] Alexandra Boldyreva, Nathan Chenette, and Adam OâĂŹNeill. 2011. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *Annual Cryptology Conference*. Springer, 578–595.

[8] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. 2011. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security* 19, 5 (2011), 895–934.

[9] Taher ElGamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory* 31, 4 (1985), 469–472.

[10] Luca Ferretti, Michele Colajanni, and Mirco Marchetti. 2012. Supporting security and consistency for cloud database. In *Cyberspace Safety and Security*. Springer, 179–193.

[11] Luca Ferretti, Michele Colajanni, and Mirco Marchetti. 2014. Distributed, concurrent, and independent access to encrypted cloud databases. *IEEE transactions on parallel and distributed systems* 25, 2 (2014), 437–446.

[12] Craig Gentry et al. 2009. Fully homomorphic encryption using ideal lattices.. In *STOC*, Vol. 9. 169–178.

[13] Mohammad Hosseini. 2012. A survey of data compression algorithms and their applications. *Network Systems Laboratory, School of Computing Science, Simon Fraser University, BC, Canada* (2012).

[14] Simon Johnson, Vinnie Scarlata, Carlos Rozas, Ernie Brickell, and Frank Mckeen. 2016. Intel® Software Guard Extensions: EPID Provisioning and Attestation Services. *White Paper* 1 (2016), 1–10.

[15] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. 2012. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 965–976.

[16] David Kaplan, Jeremy Powell, and Tom Woller. 2016. AMD memory encryption. *White paper, Apr* (2016).

[17] Florian Kerschbaum, Patrick Grofig, Isabelle Hang, Martin Härterich, Mathias Kohler, Andreas Schaad, Axel Schröpfer, and Walter Tighzert. 2013. Adjustably encrypted in-memory column-store. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 1325–1328.

[18] Florian Kerschbaum, Martin Härterich, Patrick Grofig, Mathias Kohler, Andreas Schaad, Axel Schröpfer, and Walter Tighzert. 2013. Optimal re-encryption strategy for joins in encrypted databases. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 195–210.

[19] Florian Kerschbaum, Martin Härterich, Mathias Kohler, Isabelle Hang, Andreas Schaad, Axel Schröpfer, and Walter Tighzert. 2013. An encrypted in-memory column-store: The onion selection problem. In *International Conference on Information Systems Security*. Springer, 14–26.

[20] Nagapramod Mandagere, Pin Zhou, Mark A Smith, and Sandeep Uttamchandani. 2008. Demystifying data deduplication. In *Proceedings of the ACM/IFIP/USENIX Middleware'08 Conference Companion*. ACM, 12–17.

[21] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 223–238.

[22] Antonis Papadimitriou, Ranjita Bhagwan, Nishanth Chandran, Ramachandran Ramjee, Andreas Haeberlen, Harmeet Singh, Abhishek Modi, and Saikrishna Badrinarayanan. 2016. Big Data Analytics over Encrypted Datasets with Seabed.. In *OSDI*. 587–602.

[23] Raluca Ada Popa. 2014. *Building practical systems that compute on encrypted data*. Ph.D. Dissertation. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science.

[24] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 85–100.

[25] Vincent Rijmen and Joan Daemen. 2001. Advanced encryption standard. *Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology* (2001), 19–22.

[26] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. 1978. On data banks and privacy homomorphisms. *Foundations of secure computation* 4, 11 (1978), 169–180.

[27] Ronald L Rivest, Adi Shamir, and Leonard Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 21, 2 (1978), 120–126.

[28] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. 2000. Practical techniques for searches on encrypted data. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*. IEEE, 44–55.

[29] Stephen Tu, M Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. 2013. Processing analytical queries over encrypted data. In *Proceedings of the VLDB Endowment*, Vol. 6. VLDB Endowment, 289–300.