Using Different Encryption Schemes for Secure Deletion While Supporting Queries

Maik Schott, Claus Vielhauer, Christian Krätzer

Department Informatics and Media
Brandenburg University of Applied Sciences
Magdeburger Str. 50
14770 Brandenburg an der Havel, Germany
schott@fh-brandenburg.de
claus.vielhauer@fh-brandenburg.de

Department of Computer Science
Otto-von-Guericke-University Magdeburg
Universitaetsplatz 2
39106 Magdeburg, Germany
kraetzer@cs.uni-magdeburg.de

Abstract: As more and more private and confidential data is stored in databases and in the wake of cloud computing services hosted by third parties, the privacy-aware and secure handling of such sensitive data is important. The security of such data needs not only be guaranteed during the actual life, but also at the point where they should be deleted. However, current common database management systems to not provide the means for secure deletion. As a consequence, in this paper we propose several means to tackle this challenge by means of encryption and how to handle the resulting shortcomings with regards to still allowing queries on encrypted data. We discuss a general approach on how to combine homomorphic encryption, order preserving encryption and partial encryption as means of de-personalization, as well as their use on client-side or server-side as system extensions.

1 Introduction and state of the art

With the increase of data in general stored in databases especially its outsourcing into cloud services, privacy-related informations are also becoming more and more prevalent. Therefore there is an increasing need of maintaining the privacy, confidentiality, and in general security of such data. Additionally privacy is required by several national laws, like the Family Educational Rights and Privacy Act and the Health Insurance Portability and Accountability Act of the United States, the Federal Data Protection Act (Bundesdatenschutzgesetz) of Germany, or the Data Protection Directive (Directive 95/46/EC) of the European Union. All these legal regulations require the timely and guaranteed – in the sense that it is impossible to reconstruct – removal of private information. Such removal is called forensic secure deletion.

Aside from the regular challenges of this issue, e.g. the behavior of magnetic media to partially retain the state of their previous magnetization – leaving traces of data later overwritten with other data – or the wear-levelling techniques of solid-state memory media [Gu96], RAM or swap memory copies, and remote backups, database systems have an additional complexity due to their nature to provide an efficient and fast access to data by the means of introducing several redundancies of data. A certain information is not only stored within its respective database table, but also in other locations, like indexes, logs, result caches, temporary relations or materialized views [SML07]. Deleted rows are often just flagged as deleted, without touching the actually stored data. Additionally, due to page-based storage mechanisms, changes to records on these pages, requiring a change to the layout of these pages will, not necessarily update only this very page but instead create a new copy in the unallocated parts of the file system updating with the new data, but still leaving the original page behind flagged as unallocated space. The same applies to any kind of deletion operations. Essentially, old data is being marked as deleted, however remains present and is not immediately or intentionally deleted. The later happens only occasionally, when this unallocated space is later overwritten by a new page.

An extensive study of this issue was done by Stahlberg et al. [SML07] who forensically investigated five different database storage engines - IBM DB2, InnoDB, MyISAM (both MySQL), PostgreSQL, and SQLite - with regards to traces left of deleted data within the table storage, transaction log, indexes. They found that even after applying 25,000 operations and vacuuming, a large amount of deleted records could still be found. Furthermore they investigated the cost of overwriting or encrypting (albeit using highly insecure algorithms) log entries for the InnoDB engine. Grebhahn et al. [GSKS13] especially focused on index structures and what kind and amount of traces of deleted records can be reconstructed from the structure of indexes. Albeit they didn't investigate a real database system but a mockup designed to thoroughly evaluate high-dimensional indexes, they achieved recovery rates from R trees of up to 60% in single cases.

As shown, although a forensic secure deletion is required in many cases, the actual realization of removing data once it has been ingested to a database is still a difficult or even unsolved challenge. Therefore the solution must be sought at an earlier point: the time the data first enters the database. As the concept of deletion of data basically means rendering this data unreadable/illegible it shares similarities to the encryption of data without having knowledge of the proper key, as introduced by [BL96] for backup systems. Encryption can therefore be seen as a "preventive deletion" scheme in a forensic secure way.

At the same time the illegibility of encrypted data also hinders the widespread use in database systems as most operations and therefore queries on them are not possible compared to their unencrypted state. Therefore, in this paper we discuss a general approach on how to use several encryption schemes to provide additional security, while still maintaining some of the advantageous properties of unencrypted data. A similar concept has been introduced in the context of CryptDB e.g. in [PRZ+11] and more recently in [GHH$^+$14].

2 Approach

Encryption schemes can generally be classified as symmetric and asymmetric, depending on if the encryption and decryption processes use the same shared key or different keys (public key/ and private key). As in asymmetric schemes the sender and receiver of a message use different keys, there is less trust required between both parties w.r.t. the key handling and secure storage; thus asymmetric schemes are more secure. However, this severely affects the performance and conversely symmetric schemes are several magnitudes faster than asymmetric algorithms.

In "traditional" (strong) cryptography, the goal is that the ciphertext does not reveal anything about the plaintext, e.g. any two given similar yet different plaintexts m_1 and m_2 the resulting ciphertexts are dissimilar and appear random. As such any operation which makes use of any property of the plaintext is not applicable to ciphertext. This is expressed by the cryptographic property is *ciphertext indistinguishability* introduced by [GM84] as *polynomial security*, i.e. given n plaintexts and their n ciphertexts, determining which ciphertexts refer to which plaintexts has the same probability as random guessing. Similar is the property non-malleability introduced by [DDN00] which states that given a ciphertext an attacker may not be able to modify this ciphertext in a way which would yield a related plaintext, i.e. a malleable scheme would fulfil: $E\ (m \oplus x) = E(m) \otimes x'$, with E being the encryption operation, m the plaintext, x a value to change the plaintext using the operation $\oplus$ and $x'/\otimes$ their counterparts in encrypted domain. However, there exist encryption schemes which intentionally give up on these security properties in exchange to provide for additional benefits, i.e. computational properties for mathematical operations in the encrypted domain.

The first one of these schemes is **Homomorphic Encryption** (HE), with its basic concept introduced by Rivest et al. [RAD78], as an encryption scheme allowing certain binary operations on the encrypted plaintexts in the ciphertext domain by a related homomorphic operation without any knowledge of the actual plaintext, i.e.: $E(m_1 \circ m_2) = E(m_1) \circledcirc E(m_2)$. However, most of the early homomorphic encryption schemes only allowed one operation (multiplication or addition) until Gentry [Ge09] introduced the first *Fully Homomorphic Encryption* (FHE) scheme, which allow additive as well as multiplicative arithmetic operations at the same time. As can easily be seen, homomorphic encryption does not fulfil the non-malleability criterion, since every adversary can combine two ciphertexts to create a valid new encrypted plaintext. The main drawback of this encryption scheme is that, although Gentry's approach opened up lots of interest in the scientific community, the scheme it is very intensive with regards to computational time and space requirements, with a plaintext to ciphertext expansion factor of thousands to millions [LH14], and the computation of complex/chained operations may take several seconds.

The second encryption scheme is **Order-Preserving Encryption** (OPE) introduced by Agrawal et al. [AKSX04] and cryptographically proofed by Boldyreva et al. [BCLO09], who also provided a cryptographic model based on hypergeometric distribution. The idea of order-preservation encryption is to provide an encryption scheme which maintains the property of order of plaintext within their encrypted counterparts, i.e. $\forall m_1 \geq m_2 : E(m_1) \geq E(m_2)$. It can be seen that this scheme does not fulfil the ciphertext indistinguishability criterion as by ordering the plaintexts and ciphertexts there is high probability of knowing which ciphertext can be mapped to which plaintext, except for equal plaintexts.

A third encryption scheme is **Partial Encryption,** differing from the aforementioned as it is not a specific new approach of encrypting, but a different application of existing common encryption schemes. The basic idea, as already discussed, e.g., in [MKD$^+$11] and [SSM$^+$11], is that in many complex data items only some parts are confidential. However, encrypting the complete data item may hinder the use of the data as even persons or processes who only want to access these non-confidential parts would need to get the data item decrypted in some way, e.g. by providing them with a higher security clearance, even though they should not need it. Using partial encryption, only the confidential parts of the data item are encrypted and associated with metadata specifying which parts. As the data is complex, and thus HE and OPE schemes would not be usable, partial encryption would use strong cryptography.

Combining these, our concept consists of two steps: The *first step* is to classify each data type with regards to its security level in the sense of maximum possible harm if the data gets misused or disclosed. This segmentation basically defines the amount of data to be protected as well as the type of security means needed to protect this data. Since the actual evaluation depends on the regulations of each organization, legislation, use-cases and so on, this is not the focus of this paper and thus will not be described in detail. The *second step* is to evaluate what kind of operations are commonly done to the data, in the sense of if the queries mostly consist of arithmetic operations, comparison operations, or other kinds of operations. If data is mainly used for arithmetic reasons, HE can be used to enable these kinds of operations on encrypted data. The same applies to data mainly used for comparison queries and OPE. If complex data types are present where only parts are sensitive and other parts contain useful information too, partial encryption can be used. It has to be stated here that this two-step concept is an extension of the basic approach discussed in [MKD$^+$11].However, as stated before, the HE and OPE schemes are weaker than strong encryption schemes. Therefore, based on the evaluated security level of step 1, highly confidential data which satisfy the criteria for either HE or OPE should still be encrypted using strong cryptography.

3 Realization in database systems

In this section we describe the implications of using our approach on the database management system by discussing if the encryption or parts thereof should be provided server-side or client-side, as well as necessary changes to queries. For two of the introduced schemes (HE and OPE) the integration into a query language is discussed only conceptually on query level, while for the third scheme (PE) a realization of the query language extension required is presented.

Our test MySQL database consists of actual forensic data acquired during the Digi-Dak[1] project consisting of fiber scans, synthetic fingerprints and metadata in the form of statistical, spectral and gradient features for fingerprints [MHF$^+$12] in 3.4 million tuples and diameter, length, perimeter, area, height and color as fiber features [AKV12].

3.1 Server-side vs client-side

An important notion is if the encryption E and decryption D functions should be provided client-side or server-side. On **client-side**, each client software would be required to implement both functions if it wants to create proper queries and interpret the results. However, this may be unfeasible for heterogeneous environments with many different types of client software, especially if future updates to the employed encryption are considered. The advantage of this approach is that the keys never leave the client.

If the crypto functions are centrally **server-based**, every client can make use of the encrypted data with only minor changes to the queries as described in the following sections. An important issue here is the acquisition of the keys. The keys may either be provided by the client or by the server. For *client-based key provision*, they would be part of the query and thus transmitted over the server-client connection. In this case this connection must be secured, e.g. by using SSL/TLS. For *server-based key provision* they may either be part of encryption or decryption functions themselves or are provided as part of a view as described in Section 3.4. The disadvantage *server-based key provision using views* approach is that the password needs to be stated in the view definition, and as such this encryption approach has at most the security level of the database access controls. As such it can protect confidential data against malicious clients or clients vulnerable to SQL injections and similar attacks, but not against attackers who have low-level (OS) access to the database. Therefore, client-based key provision has a higher security level, but as stated in Section 2 server-based key provision may also be feasible.

Approaches like [GSK13] propose an extension of the SQL syntax to permit special forensic tables that automatically handle secure deletion, and [SML07] proposes the use of an internal stream cipher to automatically encrypt every data. Both approaches need the extension/update of database systems on source code level which is in most cases not applicable to actual database systems in a productive environment. Therefore our approach consists of realizing secure deletion/encryption by making use of means provided by the database system, in our case external *User-Defined Functions* (UDF) – also called *call specification* in Oracle or *CLR Function* in MSSQL – from external libraries as complex cryptographic implementations are infeasible with stored procedures in SQL/PSM or related languages.

3.2 Order Preserving Encryption

Like for the other encryption schemes the data needs to be encrypted before it is stored in the database. This would be done by the client who transforms an input item *m* to

[1] http://omen.cs.uni-magdeburg.de/digi-dak/

$m' = E_{OPE}(m, key_E)$ which is then INSERTed. On a SELECT, the returned m' needs to be transformed back as $m = D_{OPE}(m', key_D)$. As described in the previous section, $E_{OPE}()$ and $D_{OPE}()$ may either be server- or client-side provided and the keys may either be explicitly provided or implicitly.

Assuming the columns of a tuple where c_2 contains confidential values mainly used for relational queries (for example with a fixed value v_1) a query in the form:

Query 1a: SELECT c_1, c_2 FROM t_1 WHERE $c_2<v_1$;

would be transformed into the following for OPE data:

Query 1b: SELECT c_1, $D_{OPE}(c_2, key_D)$ FROM t_1 WHERE $c_2<E_{OPE}(v_1, key_E)$;

In case the query result does not contain OPE data, the only overhead to non-encrypted data would be the encryption of the WHERE statement. Therefore the time complexity depends on the number and complexity of OPE statements in the SELECT statement multiplied by the result count and the number and complexity of OPE statements in the WHERE statement.

3.3 Homomorphic Encryption

For homomorphic encryption basically the same applies. Assuming the columns of a tuple where c_1 and c_2 contain confidential values mainly used for arithmetic operations queries to retrieve a value for a client or generate a value within the database:

Query 2a: SELECT $c_1 + c_2$ FROM t_1 WHERE ...;

Query 3a: INSERT INTO t_2 (c_3) SELECT $c_1 * c_2$ FROM t_1 WHERE ...;

would be transformed into the following for HE data:

Query 2b: SELECT $D_{HE}(ADD_{HE}(c_1, c_2), key_D)$ FROM t_1 WHERE ...;

Query 3b: INSERT INTO t_2 (c_3) SELECT $MUL_{HE}(c_1, c_2)$ FROM t_1 WHERE ...;

However depending on the homomorphic operations and if the query result is stored in the database or returned to the client the call to an additional function may not be needed. For example in the Paillier cryptosystem [Pa99] the addition of two plaintexts is expressed by a multiplication of the ciphertexts: $m_1 + m_2 \bmod n = m_1' * m_2' \bmod n^2$.

3.4 Partial Encryption

In our implementation the actual library and user-defined functions were written in C#, as its runtime library provides a large amount of conveniently usable image processing and cryptography functions. As C# libraries use different export signatures the Unmanaged Exports tool (MIT license) by Robert Giesecke[2] to automatically create proper C style exports and unmarshalling.

The pseudo-code of the decrypt UDF called D_{PE} is as follows:

```
INPUT: blob, password
OUTPUT: image
```

[2] https://sites.google.com/site/robertgiesecke/Home/uploads/unmanagedexports

```
(image, regions, cryptalg, cryptparams, blocksize) ← unpack(blob)
if password = Ø then return image endif
buffer ← ARRAY BYTE[1..blocksize], coord ← ARRAY POINT[1..blocksize]
b ← 0, h ← height(image), w ← width(image)
for y = 1 to h do
for x = 1 to w do
   if (x, y) in regions then
      buffer[b] ← image[x, y], coord[b] ← (x, y)
      b ← b+1
   endif
   if b = blocksize OR (y=h AND x=w)) then
      buffer' ← decrypt(buffer, cryptalg, cryptparams, password)
      for i = 1 to blocksize do image[coord[i]] = buffer'[i] endfor
      b ← 0
   endif endfor endfor
return image
```

In our scenario partial encryption is used for fingerprint scans. At a crime scene there are cases where latent fingerprints – sensitive data – may be superimposed with other non-sensitive evidence like fiber traces. Depending on the investigation goal (fingerprint of fiber analysis) it may thus become necessary to make areas containing sensitive data inaccessible. Therefore in our approach, only the fingerprint parts of the scans are encrypted by using AES, packed in ZIP container along the encryption metadata (algorithm, bit size, initialization vectors, password salt) and the outline of partially encrypted regions as shown in Figure 1.

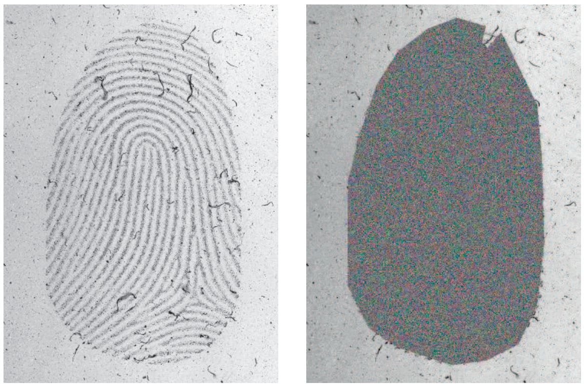
Figure 1: (Synthetic) original[3] and partially encrypted fingerprint

[3] Example from the Public Printed Fingerprint Data Set – Chromatic White Light Sensor - Basic Set V1.0. The image was acquired using sensory from the Digi-Dak research project (http://http://omen.cs.uni-magdeburg.de/digi-dak/, 2013) sponsored by the German Federal Ministry of Education and Research, see publication: Hildebrandt, M, Sturm, J., Dittmann, J., and Vielhauer, C.: Creation of a Public Corpus of Contact-Less Acquired Latent Fingerprints without Privacy Implications. Proc. CMS 2013, Springer, LNCS 8099, 2013, pp. 204–206; it uses privacy implication free fingerprint patterns generated with SFinGe, published in Cappelli, R.: Synthetic fingerprint generation, Maltoni, D., Maio, D., Jain, A.K., and Prabhakar, S. (Eds.): Handbook of Fingerprint Recognition, 2nd edn., Springer London, 2009.

The database itself only stores the container. Additionally for *server-based key provision using views* there would be two views to provide standardized means of access on the database query language level: a) an “anonymized view” for general users who do not have the proper security clearance to access unencrypted fingerprints, returning the encrypted fingerprint scan from the container (Figure 1 right side):

Query 4: CREATE VIEW fingerprints_anon AS SELECT id, filename, D_{PE}(scan) AS scan FROM fingerprints;

And b) a “deanonymized view” for privileged users like forensic fingerprint experts who do have the access rights to the actual fingerprints not only unpack the image but also try to decrypt it with the provided key (Figure 1 left side):

Query 5: CREATE VIEW fingerprints_deanon AS SELECT id, filename, D_{PE}(scan, *key*) AS scan FROM fingerprints;

Regarding the performance on our test system (Intel Core i7-4610QM @ 2.3GHz, 8 GB RAM), we observe in first experiments that querying 1000 tuples of containers takes 0.607s, unpacking the encrypted scans (Query 4) 29.110s, and returning the deanonymized scans (Query 5) 99.016s. However, it should be noted that for the Query 5 task, computation power for image parsing is included in the given figure and makes up for the main part of execution time for this query.

4 Conclusion and future work

In this paper we showed a general concept on how to use different encryption schemes for the sake of secure deletion and de-personalization, also taking into account that the data still at least partially remains usable for queries. The concept includes of the classification of data by its security level and later main usage which decides the appropriate encryption scheme. We also showed how these encryption schemes can be used in common database systems without the need to directly modify the system, but using its existing capabilities of a selected database system.

In future work a more thorough research on key management for *server-based key provision* needs to be done, as well the applicability to other database systems. Furthermore, the actual performance impact of this general concept to practical systems has to be evaluated with large-scale evaluations for relevant application scenarios, like e.g. larger forensic databases and/or biometric authentication systems where such a scheme could prevent information leakage as well as inter-system traceability.

5 Acknowledgements

The work in this paper has been funded by the Digi-Dak project by the German Federal Ministry of Education and Science (BMBF) through the research program under the contract no. FKZ: 13N10816.

References

[AKSX04] Agrawal, R.; Kiernan, J.; Srikant, R.; Xu, Y.: Order-preserving encryption for numeric data. In: SIGMOD, 2004; pages 563–574.

[AKV12] Arndt, C.; Kraetzer C.; Vielhauer, C.: First approach for a computer-aided textile fiber type determination based on template matching using a 3D laser scanning microscope. In: Proc. 14th ACM Workshop on Multimedia and Security, 2012.

[BCLO09] Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-preserving symmetric encryption. In: EUROCRYPT, 2009, pages 224–241.

[BL96] Boneh, D.; Lipton, R. J.: A revocable backup system. In: USENIX Security Symposium, 1996; pages 91–96.

[DDN00] Dolev, D.; Dwork, C.; Naor, M.: Nonmalleable Cryptography. In: SIAM Journal on Computing 30 (2), 2000; pages 391–437.

[Ge09] Gentry, C: Fully homomorphic encryption using ideal lattices. In: ACM symposium on Theory of computing, STOC '09, New York, 2009; pages 169–178.

[GM84] Goldwasser, S.; Micali, S.: Probabilistic encryption. In: Journal of Computer and System Sciences. 28 (2), 1984; pages 270–299.

[GSK13] Grebhahn, A.; Schäler, M.; Köppen, V.: Secure Deletion: Towards Tailor-Made Privacy in Database Systems. In: Workshop on Databases in Biometrics, Forensics and Security Applications (DBforBFS), BTW, Köllen-Verlag, 2013; pages 99–113.

[GSKS13] Grebhahn, A.; Schäler, M.; Köppen, V.; Saake, G.: Privacy-Aware Multidimensional Indexing. In: BTW 2013; pages 133–147.

[GHH$^+$14] Grofig, P.; Hang, I.; Härterich, M.; Kerschbaum, F.; Kohler, M.; Schaad, A.; Schröpfer, A.; Tighzert, W.: Privacy by Encrypted Databases. Preneel, B.; Ikonomou, D. (eds.): Privacy Technologies and Policy. Springer International Publishing, Lecture Notes in Computer Science, 8450, ISBN: 978-3-319-06748-3, pp. 56-69, 2014.

[Gu96] Gutmann, P.: Secure Deletion of Data from Magnetic and Solid-State Memory. In: USENIX Security Symposium, 1996.

[MHF$^+$12] Makrushin, A.; Hildebrandt, M.; Fischer, R.; Kiertscher, T.; Dittmann, J.; Vielhauer, C.: Advanced techniques for latent fingerprint detection and validation using a CWL device. In: Proc. SPIE 8436, 2012; pages 84360V.

[MKD$^+$11] Merkel, R.; Kraetzer, C.; Dittmann, J.; Vielhauer, C.: Reversible watermarking with digital signature chaining for privacy protection of optical contactless captured biometric fingerprints - a capacity study for forensic approaches. Proc. 17th International Conference on Digital Signal Processing (DSP), 2011.

[LN14] Lepoint, T.; Nehring, A.: A Comparison of the Homomorphic Encryption Schemes FV and YASHE. Africacrypt 2014.

[Pa99] Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: Eurocrypt 99, Springer Verlag, 1999; pages 223–238.

[PRZ$^+$11] Popa, R. A.; Redfield, C. M. S.; Zeldovich, N.; Balakrishnan, H.: CryptDB: Protecting Confidentiality with Encrypted Query Processing. Proc. 23rd ACM Symposium on Operating Systems Principles (SOSP), 2011.

[RAD78] Rivest, R. L.; Adleman, L.; Dertouzos, M. L.: On data banks and privacy homomorphisms. In: Foundations of Secure Computation, 1978.

[SSM$^+$11] Schäler, M.; Schulze, S.; Merkel, R.; Saake, G.; Dittmann, J.: Reliable Provenance Information for Multimedia Data Using Invertible Fragile Watermarks. In Fernandes, A. A. A.; Gray, A. J. G.; Belhajjame, K. (eds.): Advances in Databases. Springer Berlin Heidelberg, Lecture Notes in Computer Science, 7051, pp. 3-17, ISBN: 978-3-642-24576-3, 2011.

[SML07] Stahlberg, P.; Miklau, G.; Levine, B.N.: Threats to Privacy in the Forensic Analysis of Database Systems. In: SIGMOD, New York 2007, ACM; pages 91–102.