Security and Encryption at Modern Databases

Martin Ocenas
Brno University of Technology
iocenas@fit.vutbr.cz

Ivan Homoliak
Brno University of Technology
ihomoliak@fit.vutbr.cz

Petr Hanacek
Brno University of Technology
hanacek@fit.vutbr.cz

Kamil Malinka
Brno University of Technology
malinka@fit.vutbr.cz

ABSTRACT

Data is a most valuable part of most of nowadays system. A lot of hackers and criminals are trying to steal this data all the time. Due to that data should also be the best protected part of every company's systems. We would like our systems to be impenetrable, but that is not possible. If we want to protect the data, in case our system is compromised, we need to use encryption.

This article describes traditional ways of database encryption, modern concept of securing data and some possible concepts how to secure the data using encryption. All of these approaches are discussed from high-level point of view to show their impact on security of entire system.

CCS Concepts

• **Theory of computation**→**Theory and algorithms for application domains** •**Database theory**→**Theory of database privacy and security**

Keywords

Database; DBMS; Encryption; Security; Threat; Performance; SQL

1. INTRODUCTION

In todays world, data are the most valuable part of IT systems. In enterprise sector it contains business plans, models, know how, information about customers, employees, other companies etc. All of these is a pretty attractive target for hackers and malicious users which can take advantage of them and sell or disclose them. Another threat is loss of the data, through ransomware infection or it's unauthorized modification. Loss or disclosure of internal data can be devastating for most of big companies, and companies does not want to take this risk. To avoid these threats we need to secure them. The best way is, of course, to secure our systems and make them impenetrable, but that is an ideal case which we can never completely achieve in real world. To make our data really secure, we need to protect them even in case, that attacker has some access to our system.

Way to protect a data, during security breach, is an encryption. Proper usage of encryption can make it impossible to understand the data content, even if attacker can access them but do not have encryption key. Encryption can also protect the data against modification, or at least let us know that the data has been modified. In modern systems there are a lot of ways how to use encryption to protect data. Each of this way works differently and can protect the data against different threat.

We distinguish two types of threats: insider threat [1] and outsider threat. The insider threat is divided to intentional and unintentional. Both types of insider threat present danger to database system but in different ways. Intentional insider threat represents an entity that wants to somehow harm the company -- steal the data, sabotage the system, or do a fraud [2]. Unintentional threats may be hardware malfunction, natural disasters, but also users who put system's data on the public web page or make some other mistake [3]. Outsider threat represent penetration made from outside by various kind of attackers that want to steal some private data or sabotage the company.

In this paper, we will describe data encryption and how to use it, to protect our data mainly from outsider and intentional insider threat.

2. Database encryption

Encryption can prevent disclosure of data, even if attacker can access them. Only problem is if he can break thought the encryption. So data security depends on entity which performs the encryption and what exactly is encrypted.

2.1 Where can we perform encryption

There are multiple levels where can perform encryption. Each of them have different possibilities of what can they encrypt and what are possible consequences of successful attack against entity performing it.

2.1.1 File system encryption

We can encrypt hard drive, partition or just file container. We may use tools on system level - such as Linux Unified Key Setup (LUKS), or at user space like VeraCrypt [4], BitLocker [5]. Also we can take advantage of security hardware, such as Hardware Security Modules (HSM) or Trusted platform module (TPM).

Advantage of this approach is, that it is completely transparent to the rest of system, so there is no need to modify the application. Unfortunately this advantage is also a disadvantage. In the case that attacker gain control of our system, this encryption will not stop him from reading the data, because the encrypted volumes will probably be unlocked.

2.1.2 Database management system

Another place to perform encryption in is the Database Management System (DBMS). There are multiple possibilities of how to encrypt data using DBMS, and what to encrypt. Such as encryption of databases, tables and specific columns in SQL databases. These possibilities are further described in subsection 2.3.

Encrypting data at DBMS level will help us in the case attacker gain access to the machine's OS, but cannot break into DBMS. Attacker may extract the key from DBMS process or directly from the system's memory dump, but due to the required technical skills is this unlikely to happen.

2.1.3 *Application*

In this scenario, the data would be in database encrypted, but the database itself would not be able to decrypt it.

Advantages of this approach contains the independence on database's ability to encrypt data. Also application can choose, which data will be encrypted.

Main disadvantage is that we need to implement it in the application. Which adds implementation overhead and add a considerable danger of inproper implementation which may lead to security flaw.

2.2 Key storage

Encryption itself is safe and useful only as long as encryption key is not compromised. Generally it is a good idea to store the encryption key separately from the encrypted data on some other place. This safe place might be different computer, TPM, HSM, some hardware token etc. Also it is possible to store the encryption key on same place, but encrypt the key itself, before it is stored.

2.2.1 *Key management for file system encryption*

We can distinguish two possibilities of when to get the encryption key. In **pre-boot** phase we get the before the operating system is booted. This way allow us to encrypt even the system partitions, but we are not able to use much of the system's functions. If retrieving the key in the **post-boot** phase, the situation is almost opposite. We are not able to encrypt system's partitions, only the data, but we are able to use full power of the operating system.

And where can we get the encryption key from?

First possibility is to manually enter the encryption password before each boot. This is a secure way, but may lead to delays because of need of manual actions.

Another possibility is to retrieve encryption key from the **TPM** (Trusted Platform Module). TPM has the ability to distinguish which operating system is running and asking for the encryption key. Because of that, it is unlikely that TPM will leak the key to someone else than rightful operating system.

Next possibility is to get they key from key-management server though the network. In that case, the booting system authenticates to the key-management server and gets the key. This case is suitable for the post-boot authentication.

2.2.2 *Key management for the DBMS*

In this case we assume that encryption keys are used by the DBMS and, at runtime, they are in DBMS's memory. To make this work, we need to pass the encryption key to the DBMS each time it is started.

Simplest way is to create a script, that will insert the encryption key, into the DBMS process, after it starts. To make it secure we store the key on different machine and insert it via push model. In this case, attacker cannot access the encryption key if he takes control over the database machine.

Another possibility is to use special software for key storing, such as Oracle wallet [6]. This software provides a secure place to store the keys and reveals them only to the DBMS process. This solution should be secure, but it requires cooperation between DBMS and wallet system, which means specific implementation for each DBMS.

2.2.3 *Key management for the application*

Passing keys to the application is, in principle, very similar like passing key to the DBMS. We can use the same or very similar techniques.

But if we assume that application is running on different machine than database, then we can use more straight approach and add the encryption key into the application's configuration.

2.3 Encryption in SQL databases

This section describes encryption possibilities specific to SQL databases. Level of encryption differs from use cases of what are we going to defend, and against what threat.

2.3.1 *Database system encryption*

In this case we are encrypting the entire database system. It is a similar solution to the encryption of file system. Also in this case are all the data encrypted by the same key. Mayor difference is, that in this case, the encryption is performed by the DBMS, so the storage is not opened for other processes.

Advantage is that it can protect the data, even if DBMS's operating system is compromised. Encryption keys should be only stored in DBMS's memory, so the attacker cannot easily access the data. Another advantage of this solution is, that it is simpler to implement, that solutions with more granular approach.

2.3.2 *Database encryption*

In this solution each database is encrypted by different key. The encryption keys themselves are encrypted by master key as show in the Figure 1. At this approach the master key is delivered to the DBMS at a start of it's process. Advantage against the approach of encrypting entire database with one key, are a more granular key management. That allows us to change the encryption key to one database with no need to re-encrypt entire DBMS's storage, but only the impacted database.

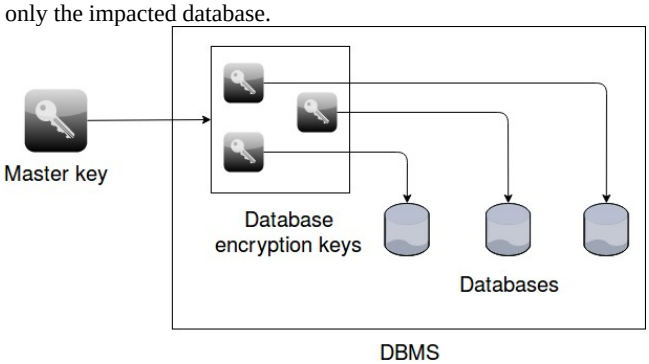

Figure 1: Database encryption keys

2.3.3 *Table encryption*

In this approach the system will distinguish the encryption on per table basis. This allows us to separate access into the tables, and we should use it only we want to protect only some tables.

Advantage of this solution is, that it is not mandatory to encrypt all the tables, which can save performance.

Disadvantage of this solution is, that structure of the database is not protected by the encryption. So if attacker gains access to the data-at-rest, then he can extract metadata from the database and it's tables. In that case, the data itself will not be leaked, but attacker will at least find out which data are stored in each table.

2.3.4 *Column encryption*

Encrypting only specific columns, we can hide only the sensitive data from each record. This can be useful to address law requirements, to protect personal data of customers or employees, so that only data-administrating employees have full access to it. Disadvantage is a rising complexity of key and system management.

3. Modern database possibilities

In time technology does change, and so does demands on it. Application needs to process big and flexible data, and be able to store it. This lead to creation of NoSQL databases. Also in nowadays the companies search for a ways to not administer their own IT infrastructure and outsource it, if possible.

This section describe technologies and approaches, usable for databases, that came up in recent years. It will briefly describe the technologies themselfs and discuss their impact on security.

3.1 Cloud

Nowadays we can hear the name cloud set on a lot of technologies, which sometimes have nothing in common. In this article by the word cloud we understand an environment which can run and scale the containers inside, across multiple physical machines. As a container we take a package of operating system, environment and application, such as docker container, which can be deployed at a cloud and run exactly the same functionality, no matter the host.

We need to distinguish types of cloud - public, private and on premise. Using the public cloud, the operator usually buys the computing resources from cloud provider, and deploys it's own containers with application, database or any other kind of service. This infrastructure is shared with other customers. Following this scheme, the operator has full control over software, running inside the containers, but has no control over the hardware nor over the hypervisor.

In private cloud, it is similar to public but the infrastructure is dedicated to one customer.

Using on premise cloud, the operator is also a cloud administrator and has control over both containers and environment. In that case the operator can take advantage of cloud's container system and still run it in private environment.

3.1.1 *Microservice architecture*

In cloud we often use a service (or microservice) architecture. This *design pattern* suggests to run separable parts of system in different runtimes. Database can be one of these services, as it is independent of application. Mayor advantage of microservice architecture is a good scalability and possible redundancy of services.

Because the communication between services is performed on the network layer, we need to also secure the communication. Even in cloud it is possible to hijack the traffic, or deploy malicious version of some service. For database it should not be a problem, because most of databases support TLS, which should be sufficient if trustworthy certificates are used.

3.1.2 *Database as a service*

Another approach is to not run in own container but use cloud possibility of Database as a service (DaaS or DbaaS). Difference is, that with DaaS the operator has no control over DBMS. Using DaaS also removes operator's possibility to control way how data is stored and secured.

Security of the data here relies entirely on the DaaS provider and it's ability to protect the data. While storing data in DaaS we must consider possibility of provider's failure to protect the data and also intentional data theft by the provider. Using business level clouds we can settle this in Service Level Agreement (SLA). Using free clouds makes a significant risk for storing the sensitive data in this way.

If we want to secure the data, no matter the provider, we need to encrypt them outside database, either by application encryption or techniques used for encrypted search.

3.1.3 *Security at cloud*

In public and private clouds infrastructure is run by provider, which has access to all data and containers in it. To protect our data we need to cover it's protection in SLA or other form of agreement. Provider will always have some way to access our data, which are placed in his cloud. Only other way is to manipulate only with encrypted data inside cloud.

Also in public clouds, environment is shared with multiple customers. This can be a security problem when bugs like L1 Terminal Fault appears. These bugs allow malicious customer to steal data from other containers running on same hypervisor. Due to that the public clouds are not sufficient for application which works with sensitive data.

Using on-premise cloud, all of the environment is under operator's control so there is no need for additional security.

Advantage of containers is an easy way to install and deploy security updates, which is a common source of security problems. Updates can be easily installed just by updating the impacted image layer and deploying the updated container. Due to the advantages of service architecture and scaling, this can be performed without outage.

3.2 Encrypted search

When encrypting data in database, we need to ask, how will it affect search of data? Will DBMS be able to determine which data we seek? And how encryption affect indexes, and thought them the search performance?

This is no problem with encryption that are transparent to the DBMS, such as file system encryption. Schemes where DBMS has access to the encryption key, and can decrypt the data should not be a problem. Problem arises in case, that encryption is performed in the application, or generally in data provider. In this case the DBMS does not know the stored data, but we would like to be able to search in them.

All the queries in the DBMS are build upon several base operations. SQL systems are mostly build upon relational algebra and it's operations like set union, difference, projection, selection, etc. NoSQL system are mostly build upon Associative arrays and it's operations like construction, find, addition. If the DBMS is able to perform all of these operations over encrypted data, it would be possible to perform even entire queries over encrypted data.

Solution which enables to do this is called *Homomorphic encryption*. Fully Homomorphic Encryption (FHE) is an encryption system, which allows to do arbitrary computation operation over encrypted data and get an encrypted result. Basic idea came in 1978 in [7]. By that time is was uncertain if it is even possible to create a FHE. A lot of partially homomorphic cryptosystems were created since that. These allowed to perform several operations over the encrypted data, but rest of the

operation must have been done over the plaintext. First construction of FHE was introduced in 2009 in [8]. Since that a lot of improvements and usage of these were made, e.g. [9], [10], [11]. Another approach to this problem were introduced in [12] and is called *Searchable Symmetric Encryption*. These approaches enables us to separate providing, administering and querying the data to different entities without security flaws. DBMS can receive the encrypted data from the provider, and administer them. By administering the data we understands storing them, search in them but also enforcing the access rules. With all that accomplished, DBMS is able to accept query from querier and return exactly the data, that was queried for. Decryption of the data is performed by querier. This approach do not disclose the data to the DBMS, and also do not disclose any additional data to the querier.

Encrypting data at the provider and storing them in the database only in encrypted form, also makes the data more secure in case the of attack. Even if attacker takes control over the DBMS, he is not able to decrypt the data. Disadvantage of this is a performance overhead, due to the [11] the overhead is between 30%-500% in the standard SQL databases.

3.3 SGX

For a long time there have been attempts to create a computation resource, which cannot be hacked and thus can be trusted. Result of such a work is called Trusted Computation Base (TCB). TCB is a hardware device, which can store data or perform operation and is by design resistant to any kind of malicious behavior. Another intention is to achieve a "Secure Remote computation" - ability to perform computation on remote machine and can prove, that the machine does not do something malicious.

One of the recent concept, designed and implemented by Intel in it's processors, is called Software Guard Extensions (SGX). SGX is a hardware inside the CPU and a set o CPU instructions to communicate with the HW. Concept and design of SGX is in more detail described in [13].

In shortcut SGX allows developer to create a container inside the CPU, this container is called *enclave*. Enclave stores data and code, and is able to execute the code in a secure environment. Enclaves are stored in special DRAM memory called PRM (Processor Reserved Memory). It is a memory placed in the CPU, and CPU protects the access to it, so no software, except the enclaves can access it, and even enforces the enclaves to be able to access only part of PRM assigned to them. Enclaves are once initiated by any process, and since that no process can directly access them, only way to communicate to the enclave is though predefined entry points.

Next part of the problem is communication with the TCB. Even we really trust SGX to be flawless and secure, we need to be able to securely communicate with it. To solve this problem, SGX can perform a *software attestation* - it is able to prove to the user, that he communicates with certain software inside trusted hardware. To achieve this, every SGX contains an asymmetric key pair, and a certificate signed by it's manufactured. Any remote machine may verify the certificate and use it to establish a secure channel directly with the SGX. This enables us to securely communicate with SGX even if SGX resides in a compromised machine.

SGX may solve the problem of untrusted environment, such as public clouds, where even a system administrator is a potential threat. There are already concepts of real data processing systems getting advantage of SGX, such as EnclaveDB [14] or VC3 [15]. These systems are designed to be secure, even if entire technology stack, except SGX, on server is compromised.

4. Related work

This section present articles, which covers multiple approaches of securing and encrypting the databases.

The article [16] describes database security model, treats to the database and security considerations to the databases. It also covers several ways of how to encrypt the database, but only of few. This article describes more ways of using the encryption end describes their impact on the security

The article [17] discuses a layers, where can we perform the encryption and impact of encryption on performance. Then it describes a problems and solutions with implementations of database encryption.

In difference to mentioned articles, this article provide an overview of classical and modern encryption possibilities and their impact on database security.

5. Conclusion

Encryption and security of data is an important task which every system developer and provider must take into account. We have described and discussed a traditional ways of securing databases through the encryption. These approaches are well supported in popular SQL databases.

Last chapters presented some modern possibilities and use cases for databases, such as cloud and containers, secure remote computation using SGX, homomorphic encryption etc. and discussed their impact on security. These technologies are still subjects of research and will take time to have them in production. But if they prove successful they can solve a lot of the security problems with DaaS, untrustworthy environments and hacked machines.

6. Acknowledgments

This work was supported by the BUT project FIT-S-17-4014 and the IT4IXS: IT4Innovations Excellence in Science project (LQ1602).

7. REFERENCES

[1] I. Homoliak, F. Toffalini, J. Guarnizo, Y. Elovici, and M. Ochoa, “Insight into insiders and it: A survey of insider threat taxonomies, analysis, modeling, and countermeasures,” ACM Computing Surveys (CSUR), vol. 52, no. 2, p. 30, 2019.

[2] D. M. Cappelli, A. P. Moore, and R. F. Trzeciak, The CERT guide to insider threats: how to prevent, detect, and respond to information technology crimes (Theft, Sabotage, Fraud). Addison-Wesley, 2012.

[3] F. L. Greitzer, J. R. Strozer, S. Cohen, A. P. Moore, D. Mundie, and J. Cowley, “Analysis of unintentional insider threats deriving from social engineering exploits,” in 2014 IEEE Security and Privacy Workshops, pp. 236–250, IEEE, 2014.

[4] “Veracrypt - free open source disk encryption with strong security for the paranoid.” https://www.veracrypt.fr/en/Home.html.

[5] “Bitlocker.” https://docs.microsoft.com/en-us/windows/security/information-protection/bitlocker/bitlocker-overview.

[6] “Using oracle wallet manager.” https://docs.oracle.com/cd/B28359 01/network.111/b28530/asowalet.htm. Accessed: 2018-05-25.

[7] R. L. Rivest, L. Adleman, and M. L. Dertouzos, “On data banks and privacy homomorphisms,” Foundations of secure computation, vol. 4, no. 11, pp. 169–180, 1978.

[8] C. Gentry, “Fully homomorphic encryption using ideal lattices,” in the 41st ACM Symposium on Theory of Computing (STOC), 2009.

[9] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, and M. Orrù, “Homomorphic secret sharing: optimizations and applications,” in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 2105–2122, ACM, 2017.

[10] H. Chen, K. Laine, and P. Rindal, “Fast private set intersection from homomorphic encryption,” in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 1243–1255, ACM, 2017.

[11] B. Fuller, M. Varia, A. Yerukhimovich, E. Shen, A. Hamlin, V. Gadepally, R. Shay, J. D. Mitchell, and R. K. Cunningham, “Sok: Cryptographically protected database search,” in Security and Privacy (SP), 2017 IEEE Symposium on, pp. 172–191, IEEE, 2017.

[12] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, “Searchable symmetric encryption: improved definitions and efficient constructions,” Journal of Computer Security, vol. 19, no. 5, pp. 895–934, 2011.

[13] V. Costan and S. Devadas, “Intel sgx explained.,” IACR Cryptology ePrint Archive, vol. 2016, p. 86, 2016.

[14] C. Priebe, K. Vaswani, and M. Costa, “Enclavedb: A secure database using sgx,” in EnclaveDB: A Secure Database using SGX, p. 0, IEEE, 2018.

[15] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, “Vc3: Trustworthy data analytics in the cloud using sgx,” in Security and Privacy (SP), 2015 IEEE Symposium on, pp. 38–54, IEEE, 2015.

[16] I. Basharat, F. Azam, and A. W. Muzaffar, “Database security and encryption: A survey study,” International Journal of Computer Applications, vol. 47, no. 12, 2012.

[17] E. Shmueli, R. Vaisenberg, E. Gudes, and Y. Elovici, “Implementing a database encryption solution, design and implementation issues,” Computers & security, vol. 44, pp. 33–50, 2014.

Authors’ background

*Title can be chosen from: master student, Phd candidate, assistant professor, lecture, senior lecture, associate professor, full professor

Your Name	Position*	Email	Research Field	Personal website
Martin Ocenas	Phd candidate	iocenas@fit.vutbr.cz	Computer security	
Ivan Homoliak	Assistant professor	ihomoliak@fit.vutbr.cz	Computer security	
Kamil Malinka	Assistant professor	malinka@fit.vutbr.cz	Computer security	
Petr Hanacek	Associate professor	hanacek@fit.vutbr.cz	Computer security	