Encryption of Data at Rest

Report: Documentation of the methodology used and results obtained

Team: 0.1134-W0R1D

Members:
Bishwambhar Dahal
Rajat Dulal
Sirjana Bhatta
Suramya Pokharel

AES-256 Algorithm

AES_256, formally known as Advanced Encryption Standard with a 256-bit key, is a widely adopted symmetric encryption algorithm that safeguards sensitive data.
It's considered highly secure due to its robust key length and complex encryption process.

How It Works:

- Key Expansion:
  1. The 256-bit key undergoes a multi-step transformation to generate multiple round keys.
  2. These keys introduce variations in each encryption round, enhancing security.

- Encryption:
  The plaintext (unencrypted data) is divided into 128-bit blocks, arranged in a 4x4 matrix.
  Each block undergoes multiple rounds of operations:
  1. *SubBytes*: Substitutes each byte with a value from a predefined table (S-box).
  2. *ShiftRows*: Rearrange bytes within rows of the matrix.
  3. *MixColumns*: Mixes data across columns using mathematical operations.
  4. *AddRoundKey*: Combines the block with a round key using XOR operation.
  5. The number of rounds depends on the key size: 14 rounds for AES_256.

- Decryption:
  The ciphertext (encrypted data) goes through the same rounds in reverse order, using inverse operations and round keys in reverse order.

- Key Features:
  1. Symmetric Encryption: Identical key used for both encryption and decryption.
  2. Block Cipher: Encrypts fixed-size blocks of data (128 bits).
  3. Key Lengths: Supports 128, 192, or 256 bits, with AES_256 offering the strongest security.
  4. Efficiency: Can be implemented efficiently in hardware and software.
  5. Resistance to Attacks: Has proven resilient against various attack methods.

- Common Uses:
  1. Secure communication protocols (HTTPS, VPNs)
  2. Data encryption at rest (hard drives, databases)
  3. Password protection
  4. File encryption software
  5. Wireless security standards

- Strengths:
  1. Strong security due to long key length and complex encryption process

2. Widely adopted and standardized
3. Efficient implementation
4. Resistant to various known attacks

- Considerations:
  1. Secure key management is crucial as the same key is used for both encryption and decryption.
  2. Potential vulnerabilities to side-channel attacks that target implementation weaknesses rather than the algorithm itself.

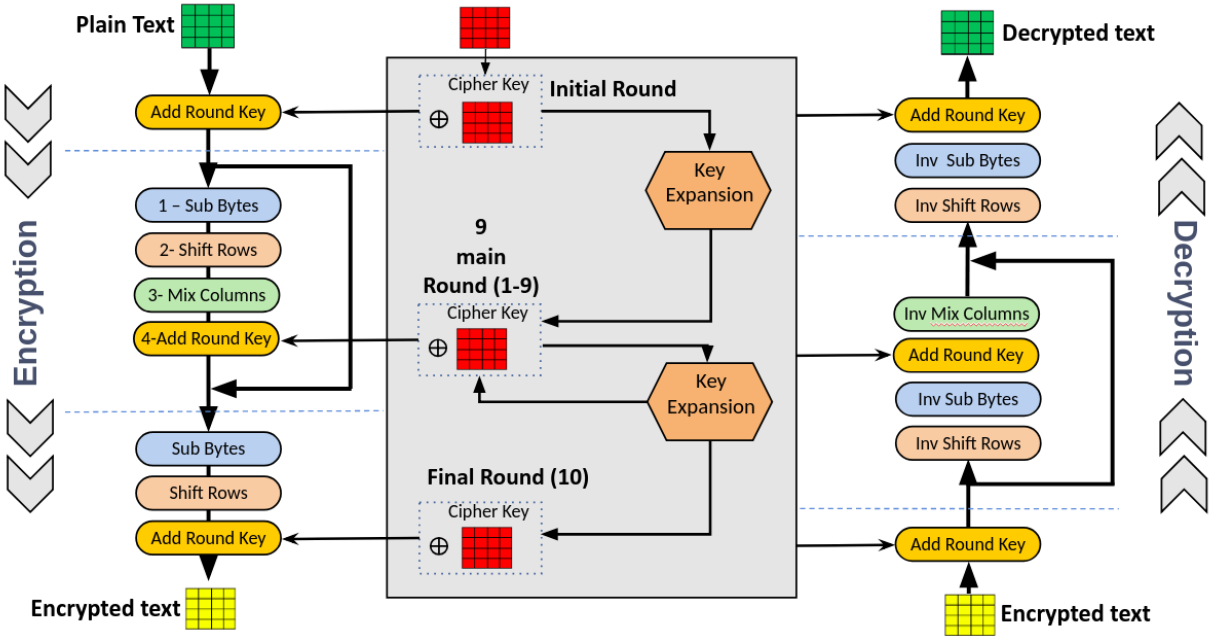

Fig: Overall Architecture of AES

Transparent Data Encryption (TDE):

Transparent Data Encryption (TDE) is a sophisticated and robust security feature designed to enhance the protection of sensitive data within a relational database management system (RDBMS). It is specifically engineered to safeguard the confidentiality and integrity of data by encrypting the entire database, rendering it unreadable and indecipherable to unauthorized users or entities. TDE operates seamlessly, integrating with the underlying database engine without necessitating modifications to existing applications or queries, thereby maintaining a transparent user experience while fortifying data security.

The primary objective of Transparent Data Encryption is to shield sensitive information stored in a database from potential breaches or unauthorized access. It achieves this by employing advanced cryptographic algorithms to encrypt the database files, including both data and log files, rendering them inaccessible to anyone lacking the appropriate cryptographic keys. This encryption process is executed in real-time, ensuring that data remains protected throughout its lifecycle, including storage, retrieval, and transmission.

One of the fundamental aspects of TDE is its ability to encrypt data at rest, meaning that even if physical storage media, such as disks or backup files, were to be compromised, the encrypted data would remain incomprehensible without the corresponding decryption keys. This feature is particularly crucial in scenarios where databases are stored on portable devices or external media, as it mitigates the risks associated with data exposure in case of loss or theft.

TDE operates at the file level, encrypting the entire database rather than specific columns or tables. This holistic approach simplifies the implementation process and ensures comprehensive protection for all sensitive data within the database. Additionally, TDE does not introduce any discernible performance overhead, allowing organizations to bolster their data security without compromising the system's responsiveness.

The management of Transparent Data Encryption typically involves the generation, storage, and protection of cryptographic keys. These keys, essential for both encryption and decryption processes, must be secured to prevent unauthorized access. The separation of duties and the implementation of robust key management practices are critical to maintaining the effectiveness of TDE. Key rotation and periodic key updates are also recommended to enhance the overall security posture.

In conclusion, Transparent Data Encryption stands as a formidable tool in the arsenal of database security measures, providing a transparent yet robust layer of protection for sensitive information. By seamlessly integrating encryption into the database infrastructure, TDE ensures that data remains confidential and secure, mitigating the risks associated with unauthorized access and data breaches. As the digital landscape continues to evolve, TDE remains a crucial component in the broader strategy to safeguard critical data assets and uphold the trust of users and stakeholders alike.

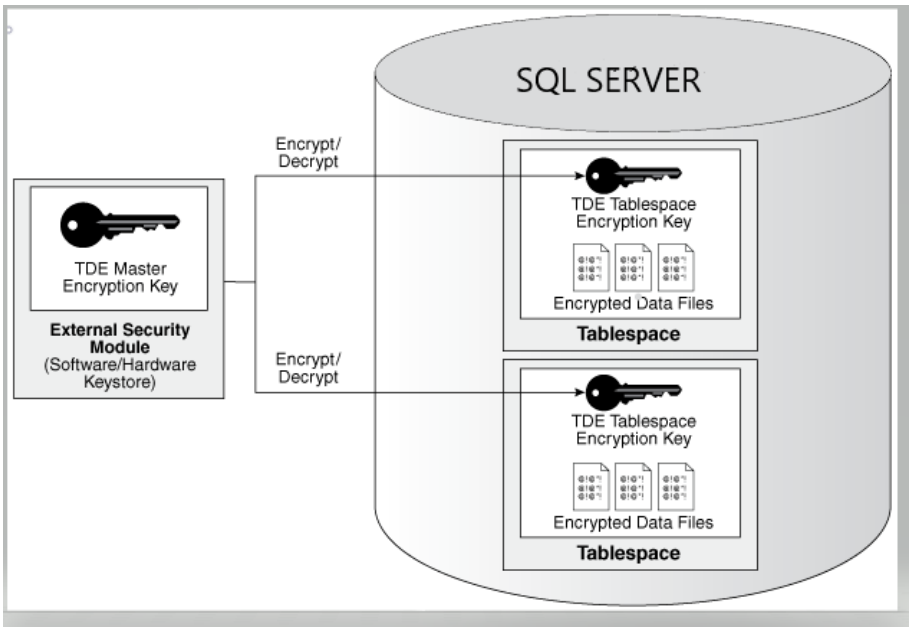

Fig: Overview of Transparent Data Encryption

Azure Key Vault:

Azure Key Vault is a cloud service provided by Microsoft Azure that allows us to securely store and manage sensitive information such as secrets, encryption keys, and certificates. It is designed to help us safeguard cryptographic keys and other secrets used by cloud applications and services. Here's an overview of key features and steps for storing a symmetric encryption key in Azure Key Vault:

Key Features of Azure Key Vault: Secure Storage: Azure Key Vault provides a secure and centralized location for storing sensitive information. All data is encrypted at rest and in transit, providing a high level of security.

- Key Management: You can create, import, and manage cryptographic keys used by cloud applications and services. Key Vault supports both software-protected keys and hardware-protected keys (HSM).

- Secrets Management: Besides keys, Key Vault allows you to store and manage other types of secrets, such as passwords and connection strings.

- Access Control: Azure Key Vault supports role-based access control (RBAC), allowing you to control who has access to specific keys and secrets.

- Audit Logging: Key Vault provides detailed logging and auditing capabilities, enabling you to monitor access and actions on keys and secrets.

Storing Symmetric Encryption Key in Azure Key Vault:

- Create an Azure Key Vault: In the Azure portal, create a new Key Vault instance.
- Set Access Policies: Configure access policies to define who can manage and use keys in the Key Vault.
- Generate or Import Symmetric Key: You can either generate a new symmetric key within the Key Vault or import an existing one. If importing, ensure that the key is securely transferred and only accessible by authorized personnel.
- Store the Symmetric Key: Use the Azure Key Vault SDK or the Azure Portal to store the symmetric key in the Key Vault.
- Retrieve the Symmetric Key: When your application needs to use the symmetric key, authenticate and authorize the application with Azure Key Vault. Retrieve the symmetric key securely using Key Vault SDK.
- Use the Symmetric Key in Your Application: Integrate the retrieved symmetric key into your application for encryption and decryption processes.

UI Designing:

In developing the user interface (UI) for a Django project, HTML and CSS were employed to structure and style the web pages. The HTML markup was organized into sections such as header, main content, and footer, facilitating a clean and modular structure. CSS styles were applied to enhance the visual aesthetics, employing responsive design principles for optimal display across diverse devices. The chosen Django framework facilitated seamless integration, with Django templates utilized to dynamically generate HTML content and ensure a consistent look and feel throughout the application. This UI development focused on user-friendly design, incorporating responsive layouts and thoughtful styling choices to enhance the overall user experience. The resulting interface aligns with the project's objectives, providing an intuitive and visually appealing platform for users interacting with the Django-based application.

Fig:- User Interface Design

Approach:

Given the requirement to protect sensitive data, particularly credit card details, and the focus on encrypting data at rest, a combination of Transparent Data Encryption (TDE) and column-level encryption with Advanced Encryption Standard (AES) could provide a robust solution. The approach follows the following method:

1. **Transparent Data Encryption (TDE):**
    - **Use Case:** TDE is an excellent choice for encrypting the entire database at rest. It operates at the file level, ensuring that data files, log files, and backup files are all encrypted.
    - **Advantages:**
        - Simplifies implementation as it encrypts the entire database, providing blanket protection against unauthorized access to the physical files.
        - Transparency to applications and users, making it easier to integrate into existing systems.

2. **Column-Level Encryption with AES:**
    - **Use Case:** For sensitive information like credit card details, you may want an additional layer of security, specifically encrypting individual columns. This is where column-level encryption using AES can be beneficial.
    - **Advantages:**
        - Allows for targeted encryption of specific columns, providing a more granular control over which data is encrypted.

        - Can be used to encrypt only the sensitive columns, reducing the overhead of encrypting the entire database.

3. **Combining TDE and Column-Level Encryption:**
    - **Strategy:** Use TDE to encrypt the entire database at rest. Additionally, implement column-level encryption with AES for specific columns containing sensitive information, such as credit card details.
    - **Advantages:**
        - Comprehensive protection: TDE safeguards the entire database, while column-level encryption adds an extra layer of security for specific sensitive columns.
        - Balances performance and security considerations by applying encryption selectively.

4. **Storage in Azure Key Vault:**
    1. **Certificates:**
        - **TDE Certificate:** The certificate used for Transparent Data Encryption (TDE) in SQL Server. It is used to protect the Database Encryption Key (DEK) during the encryption of the entire database at rest.
        - **Other Certificates:** Any additional certificates used in your application or infrastructure for secure communication, authentication, or other cryptographic operations.
    2. **Keys:**
        - **Symmetric Keys (DEK, CEK):** The symmetric keys used for Transparent Data Encryption (DEK) and column-level encryption (CEK). While the keys themselves are often stored in the database, you can store their backups or metadata in Azure Key Vault.
        - **Asymmetric Keys (CMK):** The asymmetric column-master key (CMK) generated during encryption process, you might store them in Azure Key Vault.

Why this approach?

TDE encrypts the entire database at rest, providing a holistic approach to data protection. This is crucial for safeguarding sensitive data like credit card details, ensuring that all database files, including data files, log files, and backups, are encrypted. TDE simplifies the implementation process by encrypting the entire database without requiring changes to applications or queries. It provides a foundational layer of security.

Column-based AES encryption allows for granular control over which specific columns are encrypted. For highly sensitive information like credit card details, this approach ensures that only the necessary data is encrypted. Unlike encrypting the entire database, column-level

encryption with AES introduces less overhead, making it efficient for selectively protecting specific columns.

TDE provides a baseline level of security by encrypting the entire database, while column-based AES encryption adds an additional layer of protection for specific sensitive columns. Together, they offer comprehensive security for both the entire database and targeted data elements. The combination allows for a balanced approach to security and performance. TDE secures the entire database efficiently, while column-based encryption minimizes the impact on performance by encrypting only essential columns.

Azure Key Vault provides a centralized and secure key management solution. Storing certificates and keys in Azure Key Vault ensures that cryptographic keys and secrets are protected in a dedicated and highly secure environment. Azure Key Vault is a cloud-based service that offers scalability and flexibility, making it suitable for managing keys and secrets across various services and applications. Using a dedicated key management service like Azure Key Vault aligns with best practices for security and compliance, reducing the risk of exposure or mishandling of cryptographic keys. In summary, the chosen approach leverages TDE and column-based AES encryption to achieve comprehensive and granular data protection for credit card details. The combination ensures a balanced trade-off between security and performance. Azure Key Vault is selected for centralized and secure key management, aligning with best practices and providing scalability in a cloud environment.

Results:

We have already looked at the designed UI above hence completing the functional requirements. Now for the non-functional part, the results can be seen below:

The database we created named “Rajat” has been TDE encrypted which we can see below:

	name	is_encrypted
1	master	0
2	tempdb	1
3	model	0
4	msdb	0
5	rajat	1

Fig: Statuses of database

We can see that the “Rajat” database has the value 1 for the “is_encrypted” attribute. This suggests that the database has been TDE encrypted as we desire.

The TDE certificate can be seen below:

	name	certificate_id	principal_id	pvt_key_encryption_type	pvt_key_encryption_type_desc	is_active_for_begin_dialog	issuer_name	cert_serial_number
1	##MS_SQLResourceSigningCertificate##	101	1	NA	NO_PRIVATE_KEY	0	MS_SQLResourceSigningCertificate	18 be c6 e4 d7 60 21 81 4
2	##MS_SQLReplicationSigningCertificate##	102	1	NA	NO_PRIVATE_KEY	0	MS_SQLResourceSigningCertificate	11 3a 02 0a 96 50 13 a9 4
3	##MS_SQLAuthenticatorCertificate##	103	1	NA	NO_PRIVATE_KEY	0	MS_SQLAuthenticatorCertificate	13 18 99 06 26 8e 14 a6 4
4	##MS_AgentSigningCertificate##	104	1	NA	NO_PRIVATE_KEY	1	MS_AgentSigningCertificate	49 f4 1c 93 6b 22 ef 97 49
5	##MS_PolicySigningCertificate##	105	1	NA	NO_PRIVATE_KEY	0	MS_PolicySigningCertificate	4b 11 f1 b4 cc 9c 28 82 4d
6	##MS_SmoExtendedSigningCertificate##	106	1	NA	NO_PRIVATE_KEY	0	MS_SmoExtendedSigningCertificate	74 97 80 2a 0b 32 88 ac 4
7	##MS_SchemaSigningCertificate49DAF39CDF8914F6AD71...	257	1	NA	NO_PRIVATE_KEY	1	MS_SchemaSigningCertificate49DAF39CDF8914F6AD714...	50 4a 37 8f de fb d2 bc 46
8	TDE_Cert	261	1	MK	ENCRYPTED_BY_MASTER_KEY	1	Database_Encryption	41 13 ae a2 ab d7 1a 9e 4

Fig: Certificates in the database

Among other certificates already existing with the SQL server, there is another certificate called "TDE_Cert" in the last row. That is the TDE certificate that is encrypting the DEK (Database Encryption Key) which is in turn encrypting the database.

Now, when sent in data as below:

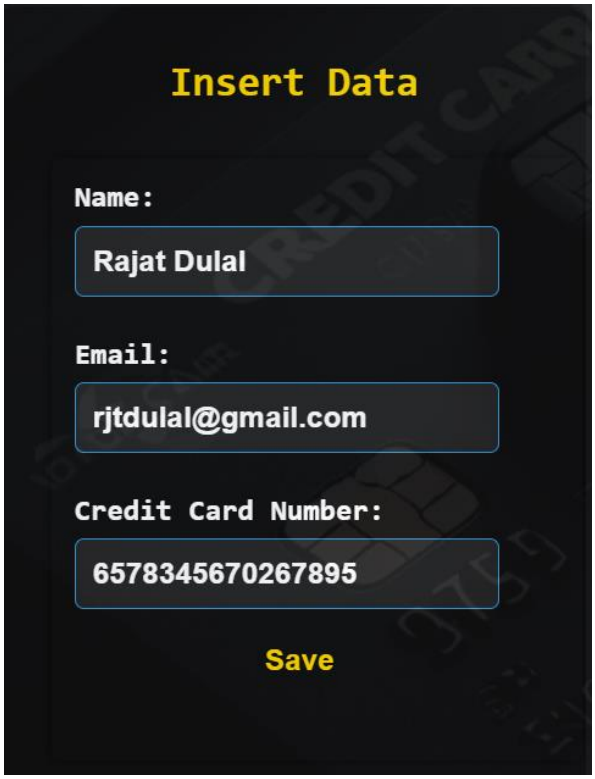

Fig: Data being inserted

The data is stored in the database as:

	fullname	email	ccn
1	Rajat Dulal	rjtdulal@gmail.com	KTlXTA2B8H/b9KE0dD0ysyGt1YlRZaGYyE2GQOct6TNd0lQDLhbWWCAiC61RqOC6

Fig: Data stored in database

We can see above that the name and email are in their normal form while the CCN column is column-wise encrypted using AES-256.

However, when the values are extracted when viewed by the unique key in email, they appear as:

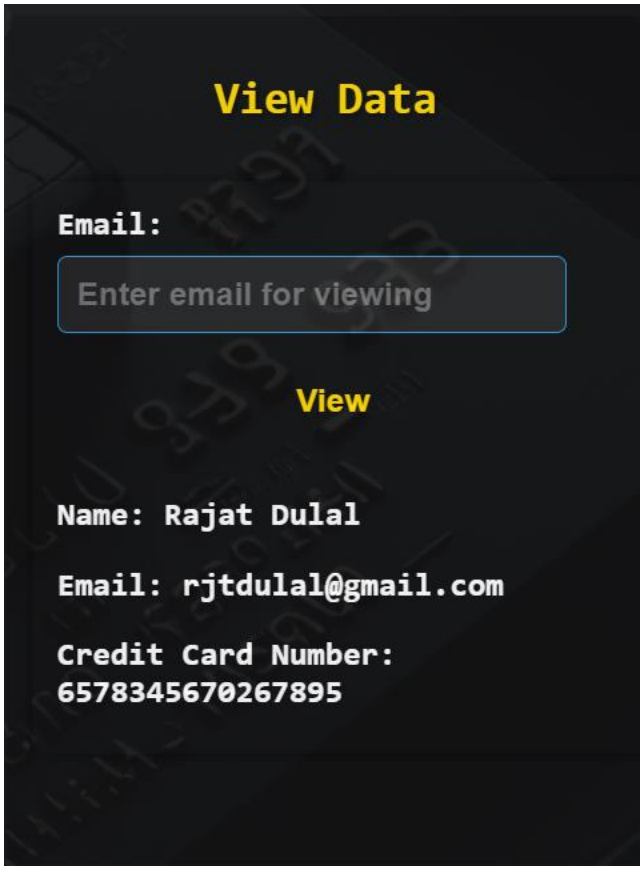

Fig: Data being viewed

We can see the encrypted credit card number is now decrypted and displayed.

Thus, this completes all the functional and non-functional requirements of our project.

How it can be implemented in larger scale?

1. **Cost Considerations:**
    - **TDE:** Transparent Data Encryption (TDE) is a feature available in Microsoft SQL Server, and its cost is often included in the licensing fees for SQL Server. There may be additional costs associated with managing and securing the keys used in TDE.

- **Column-Based AES Encryption:** The cost of implementing column-based AES encryption is primarily associated with the computational overhead for encryption and decryption. However, this cost is often justified by the granular control it provides and the reduced impact on overall system performance.
- **Azure Key Vault:** Azure Key Vault has its own pricing model based on usage, including operations performed (reads, writes) and the storage of keys and secrets. Costs can vary based on the number of applications or services utilizing the keys and the volume of key operations.

2. **Platform Independence:**
   - **TDE:** TDE is a feature native to Microsoft SQL Server, making it platform-dependent. It is designed to work seamlessly within the SQL Server environment.
   - **Column-Based AES Encryption:** AES encryption is a widely adopted standard and can be implemented across various platforms. However, the specifics of implementation may depend on the programming language and libraries available on the chosen platform.
   - **Azure Key Vault:** Azure Key Vault is a cloud service and is inherently platform-independent. It can be accessed by applications running on different platforms, including on-premises and in various cloud environments.

3. **Architecture and Scalability:**
   - **TDE:** TDE can scale horizontally as the size of the database grows. However, it's essential to monitor the performance impact, especially during encryption and decryption operations.
   - **Column-Based AES Encryption:** The implementation of column-based AES encryption allows for flexibility in scaling. It can be applied selectively to specific columns, reducing the impact on overall system performance. Consideration should be given to the choice of encryption algorithms and key management for scalability.
   - **Azure Key Vault:** Azure Key Vault is designed to scale with the demands of applications. It supports multiple applications and services, providing a centralized and scalable solution for key management.

4. **Integration with Azure Services:**
   - **TDE:** TDE integrates seamlessly with Microsoft Azure services, making it suitable for applications hosted in Azure environments.
   - **Column-Based AES Encryption:** The choice of encryption algorithm allows for integration with various Azure services and other cloud platforms.
   - **Azure Key Vault:** Azure Key Vault is specifically designed for integration with Azure services. Applications hosted in Azure can securely access keys and secrets from Azure Key Vault with proper authentication.

5. **Security and Compliance:**

- **TDE:** TDE helps meet security and compliance requirements by encrypting the entire database. It contributes to data protection and confidentiality.
- **Column-Based AES Encryption:** Selective column encryption enhances security by focusing on sensitive data elements. This is especially important for compliance with regulations that mandate protection for specific types of information, such as credit card details.
- **Azure Key Vault:** Storing keys and certificates in Azure Key Vault aligns with best practices for security and compliance. It provides centralized control and auditing capabilities, essential for meeting regulatory requirements.

6. **High Availability and Disaster Recovery:**
    - **TDE:** TDE supports high availability and disaster recovery scenarios, as long as the necessary keys are accessible during failover or recovery processes.
    - **Column-Based AES Encryption:** The implementation of column-based encryption should consider key management during high availability and disaster recovery events to ensure the availability of keys.
    - **Azure Key Vault:** Azure Key Vault itself is designed for high availability, and its features, such as geo-replication, contribute to disaster recovery planning.