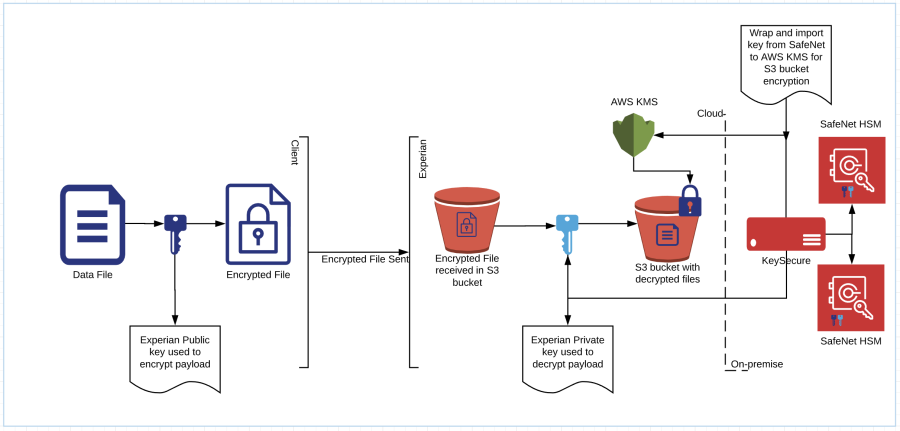
Data at rest:

Product use cases and policies

**Data transfer and disk-level encryption (PII data)**

This scenario is applicable in the scenario where PII data is part of the payload that is being transferred. The sequence of the data flow would include:

1. Using PGP public key to encrypt and transfer the payload to an Ascend landing zone (S3 bucket)
2. Decrypt the payload with the PGP private key
3. PII Data - Transfer and Storage Store the decrypted payload in an S3 bucket encrypted with Experian keys.



**Database encryption**

A company or any organization storing large data in on-premises or cloud need to be secured with effective database security. This is an essential and important for any businesses across the global to have smooth operations. Basically, this security doesn’t go beyond the basic access control mechanisms and managed services. This is not sufficient to protect the sensitive databases with basic security measures alone. With improper planning of sensitive data maintenance leads to data at high risk. Such that businesses need to maintain robust security measures for the sensitive data are smart to enable more safeguard mechanisms that can protect from external and internal threats: database encryption.

**Database encryption working flow**

A database encryption is a process of converting readable database into ciphertext of unreadable databases. By using key and algorithm, an user can decrypt and perform the operations like, retrieving and modifications. This is an essential when system was breached, the data would be in unreadable and only legitimates can decrypt with corresponding keys.

There are various ways for implementing database encryption standard with varied length of keys. Different databases we can find in the market- SQL, Access, Oracle, etc., each of these offers different database encryption standard for their customers to safeguards the stored data. To prevent the unauthorized access through the key computation, keys must be longer than usual keys.

That said, a longer key length can reduce the number of sessions per second, which can have a negative impact on throughput. Many developers hold off on database encryption precisely because they fear such performance degradations and potential system slowdowns. Additionally, encrypting a database will require more storage space than the original volume of data. While it’s true that database encryption adds some complexity ([making tasks like backup and recovery trickier](https://www.solarwindsmsp.com/products/backup/secure-backup-software)), it’s possible to ensure good performance by implementing a range of best practices. For instance, strategically implementing file-level encryption will have a much lower impact on performance than application-level or other more granular encryption methods.

**Database encryption for business security**

Whether large or small, customers of every size may very well be dealing with sensitive data—data that, in many cases, may be subject to regulations. This data might include credit cards, Social Security numbers, classified information, or medical records. Businesses deal with all manner of big data, and any kind of protected data, financial records, or personally identifiable information (PII) needs to be secured—not just as it’s used and analyzed, but while it’s in storage.

As an MSP, it’s your responsibility to assess whether a certain kind of data encryption is a useful or necessary security strategy for a client’s unique situation. For instance, encryption is typically a necessary step for businesses dealing with compliance regulations like SOX, HIPAA, PCI DSS, and GLBA. Many states penalize data breaches, and even if encryption isn’t legally mandated, businesses are often eager to prevent expensive and inconvenient losses of data.

In a sense, database encryption should be redundant, only becoming necessary if access controls and other security measures fail. However, hewing too closely to this mindset is a clear vulnerability, as these other security measures aren’t failsafe. MSPs can help businesses understand that databases need additional safeguard protections. For instance, don’t expect SSL to encrypt a database—that’s only for [data in motion](https://www.n-able.com/blog/ipsec-vs-ssl). Likewise, firewalls and VPNs are great protection, but can be breached, leaving data up for grabs. Further, internal bad actors may have to do little more than figure out a username and password to steal sensitive data.

When it comes to implementing encryption, it’s important not just to choose the right algorithm, but to manage encryption keys in a secure, organized fashion. For instance, keys shouldn’t be kept on the same server as the encrypted data. What’s more, don’t hesitate to implement database encryption for cloud storage, too—just be sure that the business itself, rather than the cloud provider, keeps track of the decryption keys.

**Common database encryption methods**

It’s possible to encrypt data at a number of levels, from the application to the database engine. For an MSP considering how to help a customer choose an encryption method, it’s important to be clear on the purposes and requirements of these different encryption methods:

* API Method: This is application-level encryption that is appropriate across any database product (Oracle, MSSQL, etc). Queries within the encrypted columns are modified within the application, requiring hands-on work. If a business has an abundance of data, this can be a time-consuming approach. Additionally, encryption that functions at the application level can lead to increased performance issues.
* Plug-In Method: In this case, you’ll attach an encryption module, or “package,” onto the database management system. This method works independently of the application, requires less code management and modification, and is more flexible—you can apply this to both commercial and open-source databases. With this option, you will typically use column-level encryption.
* TDE Method: Transparent data encryption (TDE) executes encryption and decryption within the database engine itself. This method doesn’t require code modification of the database or application and is easier for admins to manage. Since it’s a particularly popular method of database encryption, TDE is explored in further detail below.

**What is transparent database encryption?**

The term transparent data encryption, or “external encryption,” refers to encryption of an entire database, including backups. This is a method specifically for “data at rest” in tables and tablespaces—that is, inactive data that isn’t currently in use or in transit. Increasingly, transparent data encryption is a native function within database engines. It can also be handled through drive or OS encryption, meaning everything written to the disk is encrypted.

This type of encryption is “transparent” because it is invisible to users and applications that are drawing on the data and is easily used without making any application-level changes. It is decrypted for authorized users or applications when in use but remains protected at rest. Even if the physical media is compromised or the files stolen, the data as a whole remains unreadable—only authorized users can successfully read the data. This provides a disincentive for hackers to steal the data at all. When all is said and done, using TDE can help a business remain in compliance with a range of specific security regulations.

**Encryption levels**

When it comes to database encryption, it’s possible to protect data at a number of particular levels, from columns to blocks of files. All cells within these units would use the same password for access, so you can choose more specialized or generalized protection depending on your requirements. Be warned, however, that more granular encryption can dramatically reduce performance:

* Cell-Level: In this case, each individual cell of data has its own unique password—a configuration that comes with a high level of performance impact. Even so, this configuration may be appropriate in situations in which you need a highly granular level of protection. Managing the many associated keys requires careful organization.
* Column-Level: This is the most commonly known encryption level and is typically included by database vendors. Simply put, it works by encrypting columns within a database. This requires less processing than at cell-level, but could still impact performance, depending on the number of columns that are encrypted and actions like insertions, queries, and table scans. Similarly, it’s possible to implement row-level encryption in which each row of data is encrypted with its own key.
* Tablespace-Level: This method provides a different level of control over encryption, allowing encryption across tables, even if accessed by multiple columns. This method doesn’t have as much of an impact on performance but can cause issues if improperly implemented.
* File-Level: This approach works not by encrypting rows or columns, but by scrambling entire files. The files can be moved to reports, spreadsheets, or emails and still retain their protection, meaning fewer transformations or encryption mechanisms are required. This type of encryption holds the least potential for performance degradation.

**Which types of database encryption are most secure?**

Customers interested in encrypting their database will likely want to know their options. The following types of database encryption offer varying levels of protection that must be weighed against their associated performance impact in order to select a safeguard that is at once practical and effective.

* AES: The Advanced Encryption Standard is a symmetric algorithm and considered very secure. In fact, everyone from the U.S. government to software and hardware companies utilizes this algorithm. This method uses a block cipher rather than a bit-by-bit stream cipher. The block lengths are either 128, 192, or 256 bits. Users must share the key in order for others to access the data, which means they must also secure that key to prevent unauthorized access.
* RSA: Rivest-Shamir-Adleman is an asymmetric algorithm that uses a public key for encryption and a unique private key for decryption. This method is typically used for sharing data over an insecure network, which can include database encryption. The key size is between 1024 and 2048 bits, which provides higher security but a significantly slower pace than other methods.
* 3DES: Triple Data Encryption is another block cipher. It utilizes three 56-bit keys to encrypt data three times, resulting in a 168-bit key. This option is fairly secure, but also slower due to the multiple encryptions. While currently in place for a number of businesses, 3DES likely won’t last much longer as a standard.
* Twofish: Twofish is also a symmetric block cipher, with keys ranging from 128 bits to 256 bits. It’s a fairly flexible method, especially since it’s license-free. The number of encryption rounds is always 16, but you can choose whether you want key setup or encryption to be the quicker process.

**Ensuring optimal security with encryption**

If you’re still relying exclusively on firewalls or access controls to protect sensitive business data—or even if you have an old encryption method in place—it might be time to offer a better solution to your customers. Data encryption doesn’t have to lead to worse performance if you ensure that your encryption types and implementation follow best practices and strike an appropriate balance between security and usability. Interested in finding out more? [Speak with a member of our team](https://www.solarwindsmsp.com/contact) to see if your current encryption methods are implemented effectively.

* + Native
  + Transparent Data Encryption (TDE)
* Folder/network share encryption
* File level encryption
* Field or column level encryption