Securing Data at Rest with Encryption

Ken Beer and Ryan Holland

November 2013

(Please consult **http://aws.amazon.com/whitepapers** for the latest version of this whitepaper)

Abstract

Organizational policies, or industry or government regulations, might require the use of encryption at rest to protect your data. The flexible nature of Amazon Web Services (AWS) allows you to choose from a variety of different options that meet your needs. This whitepaper provides an overview of different methods for encrypting your data at rest available today.

Introduction

Amazon Web Services (AWS) delivers a secure, scalable cloud computing platform with high availability, offering the flexibility for you to build a wide range of applications. If you require an additional layer of security for the data you store in the cloud, there are several options for encrypting data at rest—ranging from completely automated AWS encryption solutions to manual, client-side options. Choosing the right solutions depends on which AWS service you're using and your requirements for key management. This whitepaper provides an overview of various methods for encrypting data at rest in AWS. Links to additional resources are provided for a deeper understanding of how to actually implement the encryption methods discussed.

The Key to Encryption: Who Controls the Keys?

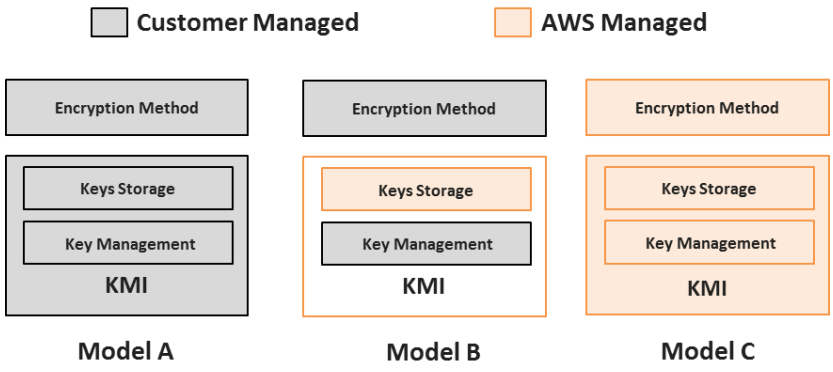
Encryption on any system requires three components: (i) data to encrypt; (ii) a method to encrypt the data using a cryptographic algorithm; and (iii) encryption keys to be used in conjunction with the data and the algorithm. Most modern programming languages provide libraries with a wide range of available cryptographic algorithms, like the Advanced Encryption Standard (AES). Choosing the right algorithm involves evaluating security, performance and compliance requirements specific to your application. While the selection of an encryption algorithm is important, protecting the keys from unauthorized access is critical. Managing the security of encryption keys is often performed using a key management infrastructure (KMI). A KMI is itself composed of two sub-components: the storage layer that protects the plaintext keys and the management layer that authorizes key usage. A common way to protect keys in a KMI is to use a hardware security module (HSM). An HSM is a dedicated storage and data processing device that performs cryptographic operations using keys on the device. An HSM typically provides tamper evidence or resistance to protect keys from unauthorized use. A software-based authorization layer controls who can administer the HSM and which users or applications can use which keys in the HSM.

As you deploy encryption for various data classifications in AWS, it is important to understand exactly who has access to your encryption keys or data and under what conditions. There are three different models for how you and/or AWS provide the encryption method and the KMI.

A. You control the encryption method and the entire KMI.

B. You control the encryption method; AWS provides the storage component of the KMI while you provide the management layer of the KMI.

C. AWS controls the encryption method and the entire KMI.



Model A: You control the encryption method and the entire KMI

In this model, you use your own KMI to generate, store and manage access to keys as well as control all encryption methods in your applications. This physical location of the KMI and the encryption method can be outside of AWS or in an Amazon Elastic Compute Cloud (Amazon EC2) instance you own. The encryption method can be a combination of open-source tools, AWS SDKs, or third-party software and/or hardware. The important security property of this model is that you have full control over the encryption keys and the execution environment that utilizes those keys in the encryption code. AWS has no access to your keys and cannot perform encryption or decryption on your behalf. You are responsible for the proper storage, management, and use of keys to ensure the confidentiality, integrity, and availability of your data. Data can be encrypted in AWS services as described in the following sections.

Amazon S3

You can encrypt data using any encryption method you want, and then upload the encrypted data using the Amazon Simple Storage Service (Amazon S3) APIs. Most common application languages include cryptographic libraries that allow you to perform encryption in your applications. Two commonly available open source tools are Bouncy Castle and OpenSSL. Once you have encrypted an object and safely stored the key in your KMI, the encrypted object can be uploaded to Amazon S3 directly with a PUT request. To decrypt this data, you issue the GET request in the Amazon S3 API and then pass the encrypted data to your local application for decryption.

AWS provides an alternative to these open source encryption tools with the Amazon S3 encryption client; itself an open source set of APIs embedded into the AWS SDKs. This client lets you supply a key from your KMI that can be used to encrypt or decrypt your data as part of the call to Amazon S3. The SDK leverages Java Cryptography Extensions (JCE) in your application to take your symmetric or asymmetric key as input and encrypt the object prior to uploading to Amazon S3. The process is reversed when the SDK is used to retrieve an object; the downloaded encrypted object from Amazon S3 is passed to the client along with the key from your KMI. The underlying JCE in your application decrypts the object. The Amazon S3 encryption client is integrated into the AWS SDKs for Java, Ruby, and .NET and provides a transparent drop-in replacement for any cryptographic code you may have used previously with your application that interacts with Amazon S3. While Amazon provides the encryption method, you control the security of your data because you control the keys for that engine to use. If you're using the Amazon S3 encryption client on-premises, AWS never has access to your keys or unencrypted data. If you're using the client in an application running in Amazon EC2, a best practice is to pass keys to the client using secure transport (e.g. SSL or SSH) from your KMI to help ensure confidentiality. For more information, see the AWS SDK for Java documentation and Using Client-Side Encryption in the *Amazon S3 Developer Guide*. Figure 1 shows how these two methods of client-side encryption work for Amazon S3 data.

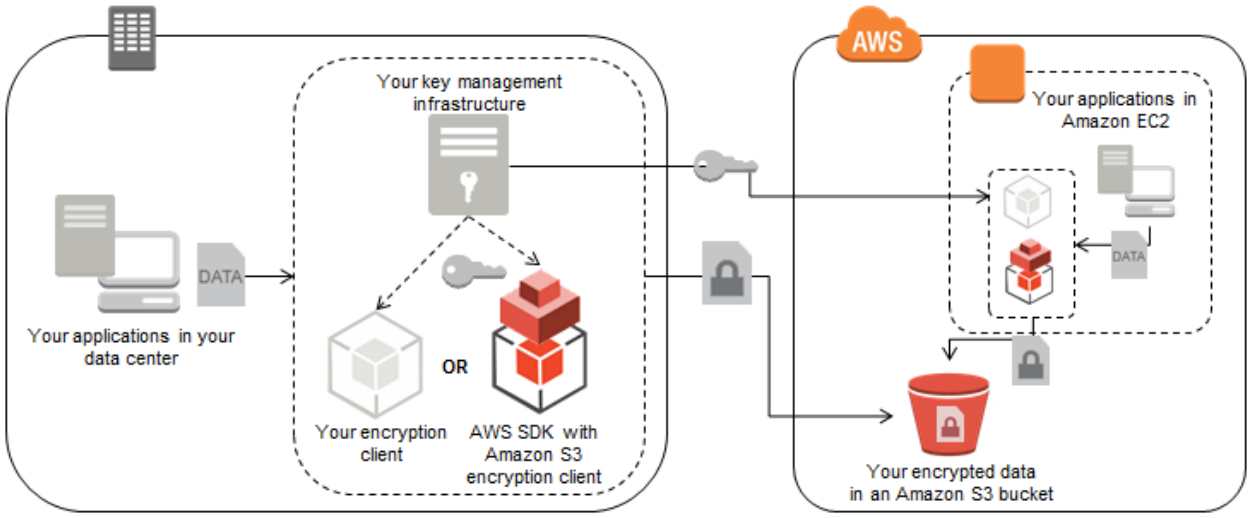


Figure 1: Amazon S3 client-side encryption from on-premises system or from within your Amazon EC2 application

There are third-party solutions available that can simplify the key management process when encrypting data to Amazon S3. CloudBerry Explorer PRO for Amazon S3 and CloudBerry Backup both offer a client-side encryption option that applies a user-defined password to the encryption scheme to protect files stored on Amazon S3. For programmatic encryption needs, SafeNet ProtectApp for Java integrates with the SafeNet KeySecure KMI to provide client-side encryption in your application. The KeySecure KMI provides secure key storage and policy enforcement for keys that are passed to the ProtectApp Java client compatible with the AWS SDK. The KeySecure KMI can run as an on-premises appliance or as a virtual appliance in Amazon EC2. Figure 2 shows how the SafeNet solution can be used to encrypt data stored on Amazon S3.

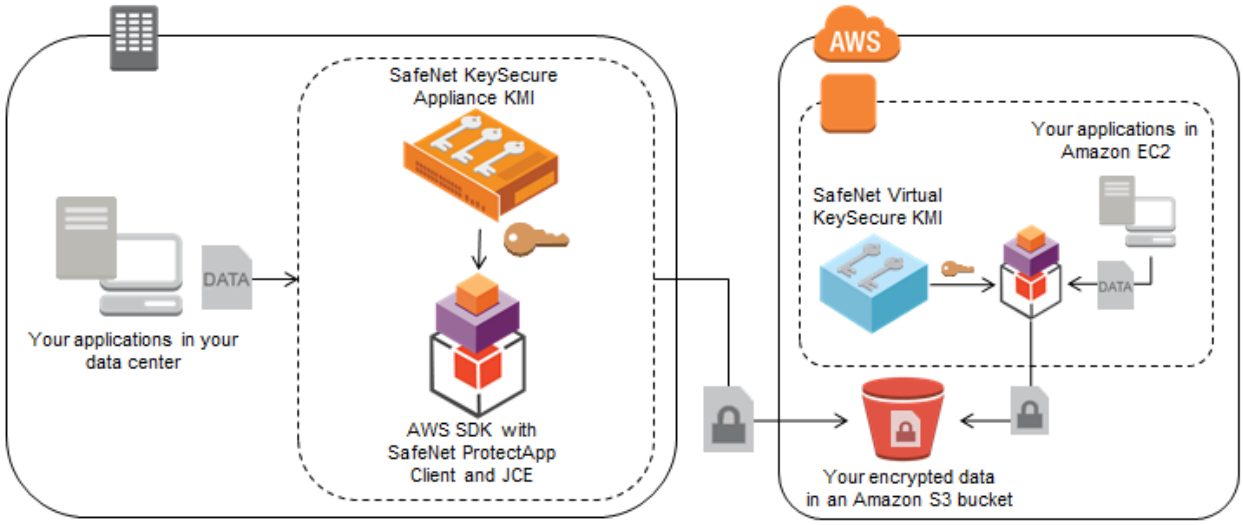


Figure 2: Amazon S3 client-side encryption from on-premises system or from within your application in Amazon EC2 using SafeNet ProtectApp and SafeNet KeySecure KMI

Amazon EBS

Amazon Elastic Block Store (Amazon EBS) provides block-level storage volumes for use with Amazon EC2 instances. Amazon EBS volumes are network-attached, and persist independently from the life of an instance. Because Amazon EBS volumes are presented to an instance as a block device, you can leverage most standard encryption tools for file system-level or block-level encryption. Some common block-level open source encryption solutions for Linux are *Loop-AES*, *dm-crypt* (with or without) *LUKS*, and *TrueCrypt*. Each of these operates below the file system layer using kernel space device drivers to perform encryption and decryption of data. These tools are useful when you want all data written to a volume to be encrypted regardless of what directory the data is stored in. Another option would be to use file system-level encryption, which works by stacking an encrypted file system on top of an existing file system. This method is typically used to encrypt a specific directory. *eCryptfs* and *EncFs* are two Linux-based open source examples of file system-level encryption tools. These solutions all require you to provide keys, either manually or from your KMI. An important caveat with both block-level and file system-level encryption tools is that they can only be used to encrypt data volumes that are not Amazon EBS boot volumes. This is because these tools don't allow you to automatically make a trusted key available to the boot volume at startup.

Encrypting Amazon EBS volumes attached to Windows instances can be done using *BitLocker* or *Encrypted File System (EFS)* as well as open source applications like TrueCrypt. In either case, you still need to provide keys to these encryption methods and you can only encrypt data volumes.

There are AWS partner solutions that can help automate the process of encrypting Amazon EBS volumes as well as supplying and protecting the necessary keys. Both Trend Micro SecureCloud and SafeNet ProtectV are two such partner products that encrypt Amazon EBS volumes and include a KMI. Both products are able to encrypt boot volumes in addition to data volumes. These solutions also support use-cases where Amazon EBS volumes attach to auto-scaled Amazon EC2 instances. Figure 3 shows how the SafeNet and Trend Micro solutions can be used to encrypt data stored on Amazon EBS using keys managed on-premises, via SaaS, or in software running on EC2.

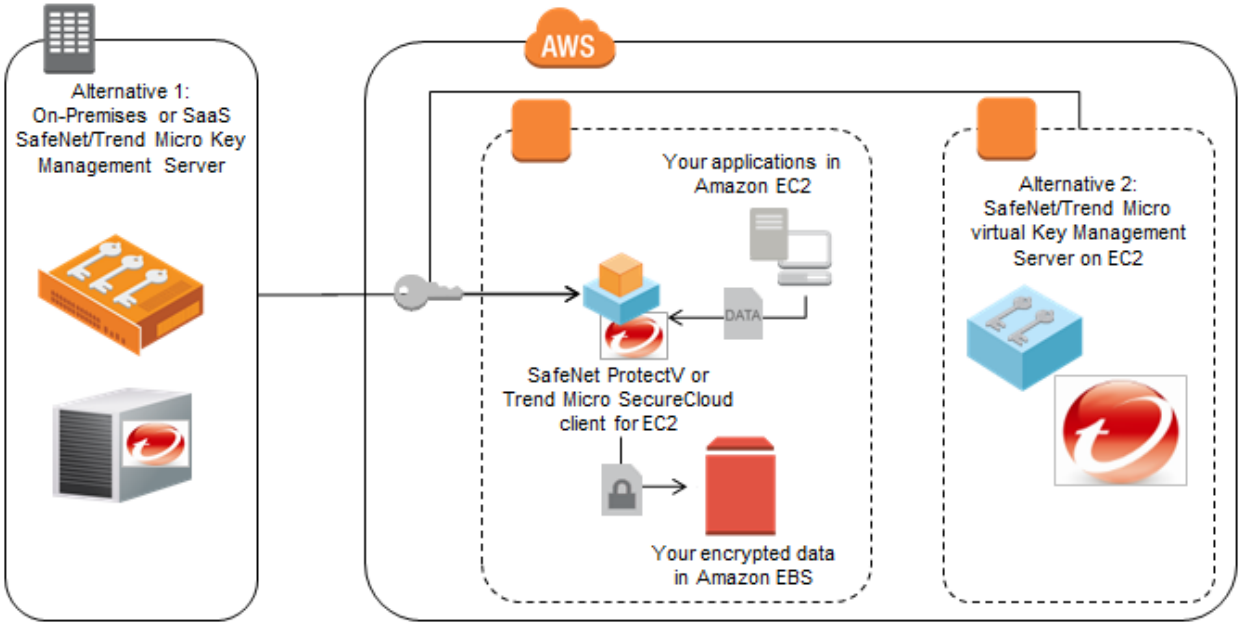


Figure 3: Encryption in Amazon EBS using SafeNet ProtectV or Trend Micro SecureCloud

AWS Storage Gateway

AWS Storage Gateway is a service connecting an on-premises software appliance with Amazon S3. It can be exposed to your network as an iSCSI disk to facilitate copying data from other sources. Data on disk volumes attached to the AWS Storage Gateway will be automatically uploaded to Amazon S3 based on policy. You can encrypt source data on the disk volumes using any of the file encryption methods described previously (e.g. Bouncy Castle or OpenSSL) before it reaches the disk. You can also use a block-level encryption tool (e.g. BitLocker or dm-crypt/LUKS) on the iSCSI endpoint that AWS Storage Gateway exposes to encrypt all data on the disk volume. Alternatively, two AWS partner solutions, Trend Micro SecureCloud and SafeNet StorageSecure, can perform both the encryption and key management for the iSCSI disk volume exposed by AWS Storage Gateway. These partners provide an easy, check box solution to both encrypt data and manage the necessary keys that is similar in design to how their Amazon EBS encryption solutions work.

Amazon RDS

Encryption of data in Amazon Relational Database Service (Amazon RDS) using client-side technology requires you to consider how you want data queries to work. Because Amazon RDS doesn't expose the attached disk it uses for data storage, transparent disk encryption using techniques described in the previous Amazon EBS section are not available to you. However, selective encryption of database fields in your application can be done using any of the standard encryption libraries mentioned previously (e.g. Bouncy Castle, OpenSSL) in your application before the data is passed to your Amazon RDS instance. While this specific field data would not easily support range queries in the database, queries based on unencrypted fields can still return useful results. The encrypted fields of the returned results can be decrypted by your local application for presentation. To support more efficient querying of encrypted data, you can store a keyed-hash message authentication code (HMAC) of an encrypted field in your schema as well as supplying a key for the hash function. Subsequent queries of protected fields that contain the HMAC of the data being sought would not disclose the plaintext values in the query. This allows the database to perform a query against the encrypted data in your database without disclosing the plaintext values in the query. Any of the encryption methods you choose must be performed on your own application instance before data is sent to the Amazon RDS instance.

CipherCloud and Voltage Secure are two AWS partners with solutions that simplify protecting the confidentiality of data in Amazon RDS. Both vendors have the ability to encrypt data using format-preserving encryption (FPE) that allows ciphertext to be inserted into the database without breaking the schema. They also support tokenization options with integrated lookup tables. In either case your data is encrypted or tokenized in your application before being written to the Amazon RDS instance. These partners provide options to index and search against databases with encrypted or tokenized fields. The unencrypted or untokenized data can be read from the database by other applications without needing to distribute keys or mapping tables to those applications to unlock the encrypted or tokenized fields. For example, you could move data from Amazon RDS to the Amazon Redshift data warehousing solution and run queries against the non-sensitive fields while keeping sensitive fields encrypted or tokenized. Figure 4 shows how the Voltage solution can be used within Amazon EC2 to encrypt data before being written to the Amazon RDS instance. The encryption keys are pulled from the Voltage KMI located in your data center by the Voltage client running on your applications on Amazon EC2.

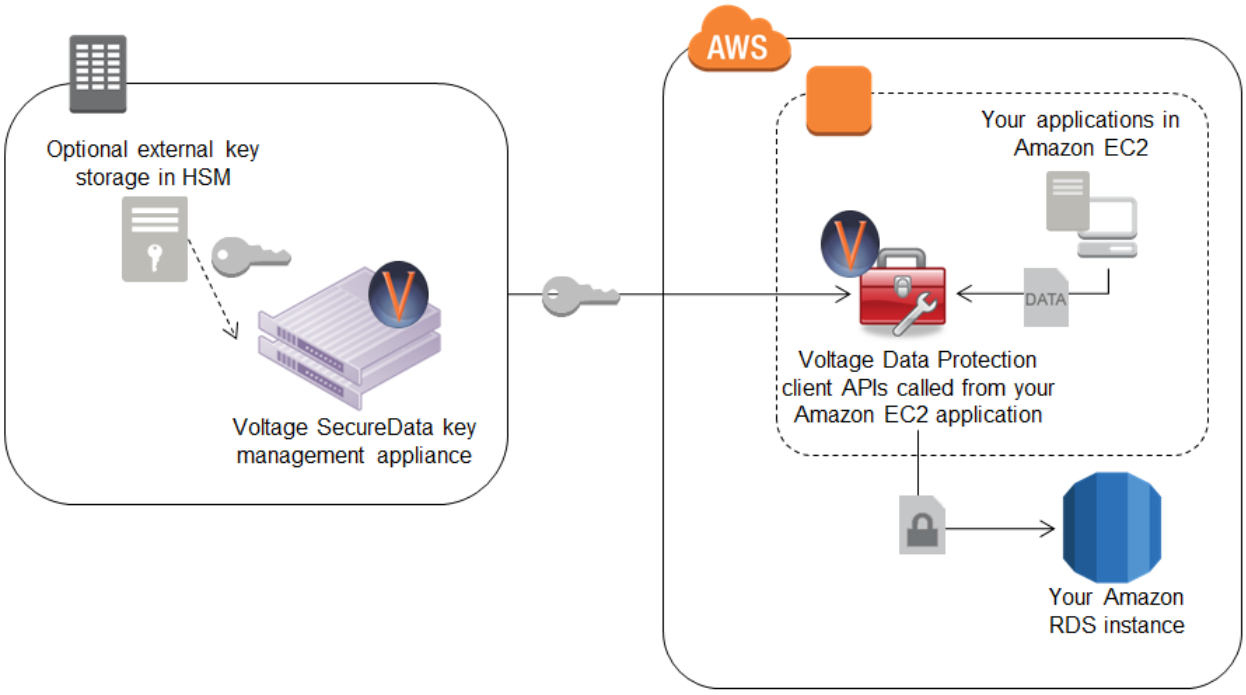


Figure 4: Encrypting data in your Amazon EC2 applications before writing to Amazon RDS using Voltage SecureData

CipherCloud for Amazon Web Services is a solution that works similar to the way the Voltage client does for applications running in Amazon EC2 that need to send encrypted data to and from Amazon RDS. CipherCloud provides a JDBC driver that can be installed on the application, regardless of whether it's running in EC2 or in your datacenter. In addition, the CipherCloud for Any App solution can be deployed as an inline gateway to intercept data as it is being sent to and from your Amazon RDS instance. Figure 5 shows how the CipherCloud solution can be deployed this way to encrypt or tokenize data leaving your data center before being written to the Amazon RDS instance.

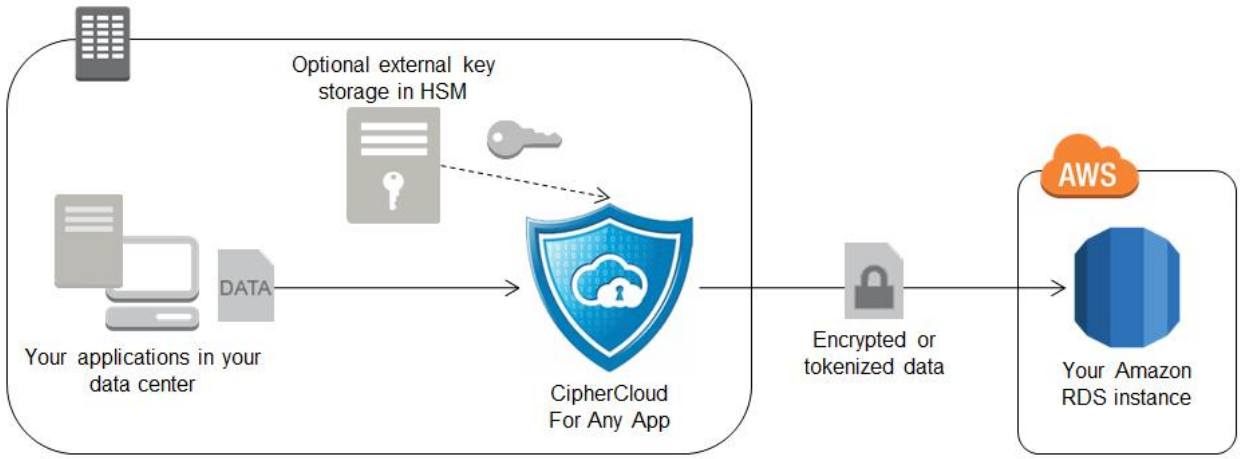


Figure 5: Encrypting data in your data center before writing to Amazon RDS using CipherCloud Encryption Gateway

Amazon EMR

Amazon Elastic MapReduce (Amazon EMR) provides an easy-to-use Hadoop implementation running on Amazon EC2. Performing encryption throughout the MapReduce operation involves encryption and key management at four distinct points:

1. The source data
2. Hadoop Distributed File System (HDFS)
3. Shuffle phase
4. Output data

If the source data is not encrypted, then this step can be skipped and SSL can be used to help protect data in transit to the Amazon EMR cluster. If the source data is encrypted, then your MapReduce job will need to be able to decrypt the data as it is ingested. If your job flow uses Java and the source data is in Amazon S3, you can use any of the client decryption methods described in the previous Amazon S3 sections.

The storage used for the HDFS mount point is the ephemeral storage of the cluster nodes. Depending upon the instance type there may be more than one mount. Encrypting these mount points requires the use of an Amazon EMR bootstrap script that will:

- stop the Hadoop service;
- install a file system encryption tool on the instance;
- create an encrypted directory to mount the encrypted file system on top of the existing mount points;
- restart the Hadoop service.

You could, for example, perform these steps using the open source eCryptfs package and an ephemeral key generated in your code on each of the HDFS mounts. You don't need to worry about persistent storage of this encryption key, as the data it encrypts does not persist beyond the life of the HDFS instance.

The shuffle phase involves passing data between cluster nodes before the reduce step. To encrypt this data in transit, you can enable SSL with a configure Hadoop bootstrap option when you create your cluster.

Finally, to enable encryption of the output data, your MapReduce job should encrypt the output using a key sourced from your KMI. This data can be sent to Amazon S3 for storage in encrypted form.

The AWS partner Gazzang simplifies managing the encryption and KMI to provide end-to-end encryption of data through an Amazon EMR job. Their Gazzang CloudEncrypt™ solution allows you to upload data to Amazon S3 using their integrated client-side encryption and then add a bootstrap script to your Amazon EMR cluster. CloudEncrypt will encrypt data used by all the HDFS mount points as well as handle key management both for input data retrieved from and output data written to Amazon S3. Figure 6 shows how the Gazzang solution can be used to perform end-to-end encryption of data used within Amazon EMR.

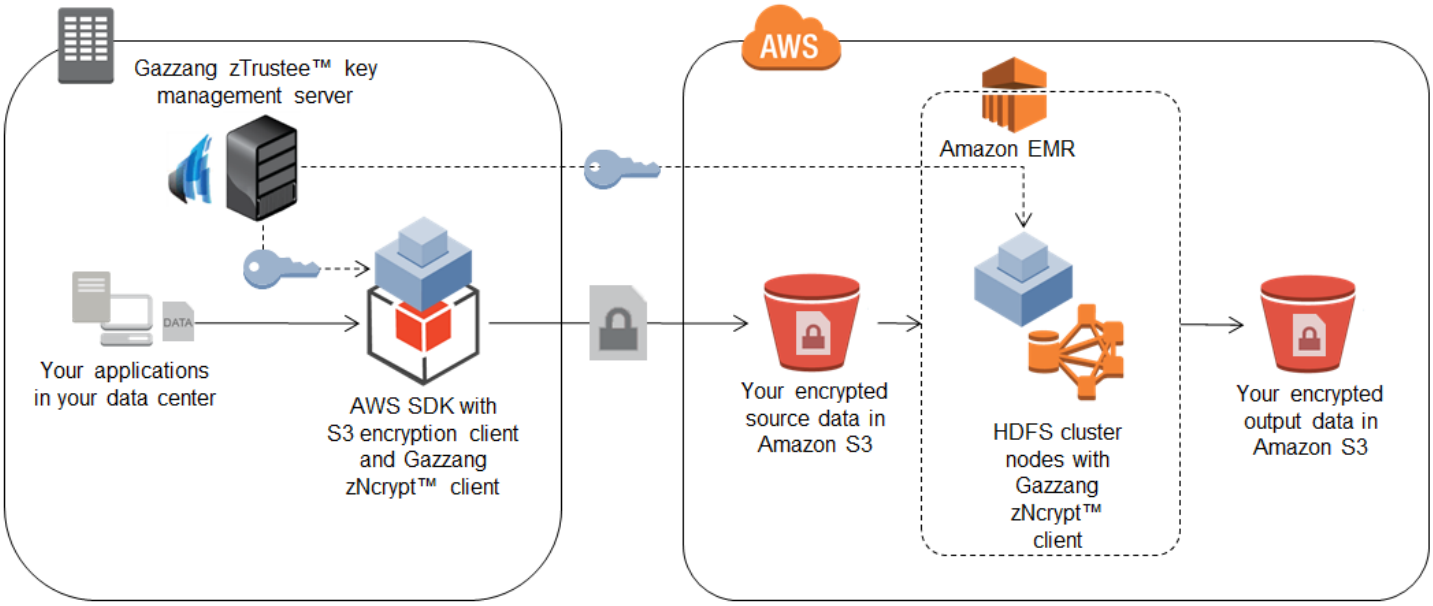


Figure 6: End-to-end encryption of data in Amazon EMR using Gazzang CloudEncrypt™

Model B: You control the encryption method; AWS provides the storage component of the KMI while you provide the management layer of the KMI

This model is similar to Model A in that you manage the encryption method, but differs in that the keys are stored in an AWS CloudHSM appliance rather than in a key storage system you manage on-premises. While the keys are stored in the AWS environment, they are inaccessible to any employee at AWS. This is because only you have access to the cryptographic partitions within the dedicated HSM to use the keys. The AWS CloudHSM appliance has both physical and logical tamper detection and response mechanisms that trigger zeroization of the appliance. Zeroization erases the HSM's volatile memory where any keys in the process of being decrypted were stored and destroys the key that encrypts stored objects, effectively causing all keys on the HSM to be inaccessible and unrecoverable.

When determining if using AWS CloudHSM is appropriate for your deployment, it is important to understand the role that an HSM plays in encrypting data. An HSM can be used to generate and store key material and can perform encryption and decryption operations, but it does not perform any key lifecycle management functions (e.g. access control policy, key rotation). This means that a compatible KMI may be needed in addition to the AWS CloudHSM appliance before deploying your application. The KMI you provide can be deployed either on-premises or within Amazon EC2 and can communicate to the AWS CloudHSM instance securely over SSL to help protect data and encryption keys. Because the AWS CloudHSM service uses SafeNet Luna appliances, any key management server that supports the SafeNet Luna platform can also be used with AWS CloudHSM. Any of the encryption options described for AWS services in Model A can work with AWS CloudHSM as long as the solution supports the SafeNet Luna platform. This allows you to run your KMI within the AWS compute environment while maintaining a root of trust in a hardware appliance to which only you have access.

Applications must be able to access your AWS CloudHSM appliance in an Amazon Virtual Private Cloud (Amazon VPC). The AWS CloudHSM client, provided by SafeNet, interacts with the AWS CloudHSM appliance to encrypt data from your application. Encrypted data can then be sent to any AWS service for storage. Database, disk volume, and file encryption applications can all be supported with AWS CloudHSM and your custom application. Figure 7 shows how the AWS CloudHSM solution works with your applications in an Amazon VPC, Amazon EC2 instance.

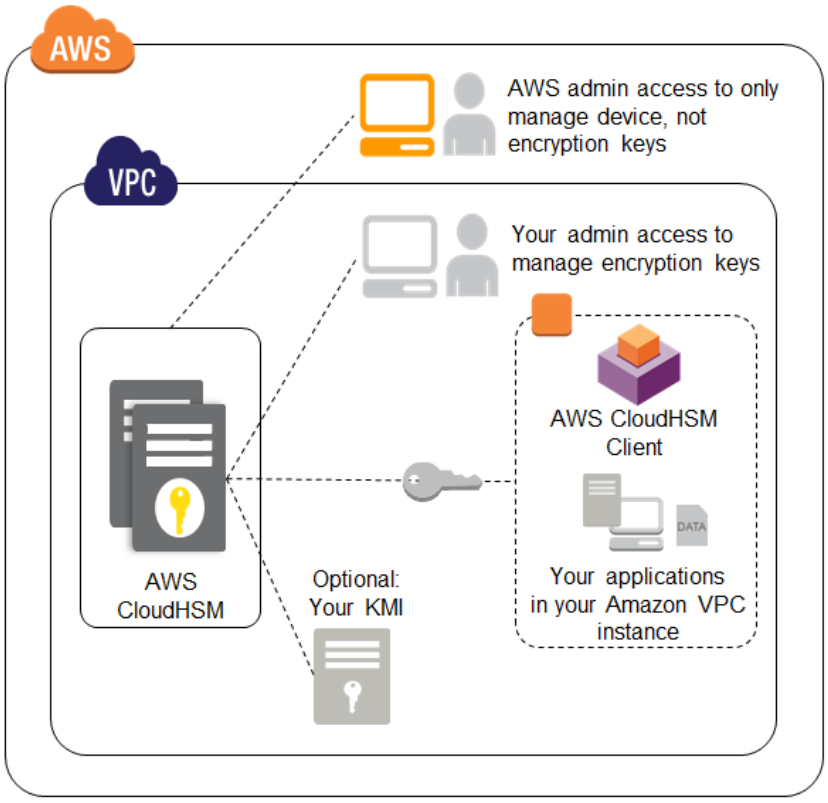


Figure 7: AWS CloudHSM deployed in Amazon VPC

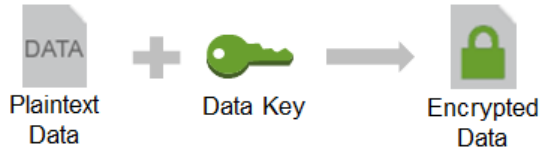
In order to achieve the highest availability and durability of keys in your AWS CloudHSM appliance, we recommend deploying multiple AWS CloudHSM applications across Availability Zones or in conjunction with an on-premises SafeNet Luna appliance that you manage. The SafeNet Luna solution support secure replication of keying material across appliances. For more information, see AWS CloudHSM on the AWS website.

Model C: AWS controls the encryption method and the entire KMI

In this model, AWS provides server-side encryption of your data, transparently managing the encryption method and the keys. Envelope encryption is used in AWS for server-side encryption. Figure 8 describes envelope encryption.

1. A data key is generated by the AWS service at the time you request your data to be encrypted.

2. Data key is used to encrypt your data.



3. The data key is then encrypted with a key-encrypting key unique to the service storing your data.

4. The encrypted data key and the encrypted data are then stored by the AWS storage service on your behalf.

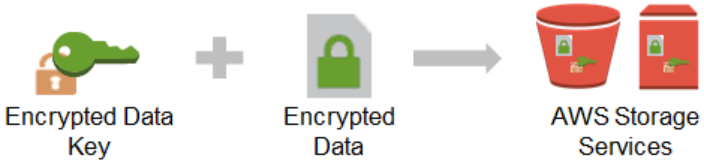


Figure 8: Envelope encryption

The key-encrypting keys used to encrypt data keys are securely stored and managed separately from the data and the data keys. Strict access controls are placed on the encryption keys designed to prevent unauthorized use by AWS employees. When you need access to your plaintext data, this process is reversed. The encrypted data key is decrypted using the key-encrypting key; the data key is then used to decrypt your data.

The following AWS services offer server-side encryption:

Amazon S3

You can set an API flag or check a box in the AWS Management Console to have data encrypted before it is written to disk in Amazon S3. Each object is encrypted with a unique key. As an additional safeguard, this key itself is encrypted with a periodically rotated master key unique to Amazon S3 that is securely stored in separate systems under AWS control. Amazon S3 server-side encryption uses 256-bit Advanced Encryption Standard (AES) keys for both object and master keys. This feature is offered at no additional cost beyond what you pay for using Amazon S3.

Amazon Glacier

Data is always automatically encrypted before it's written to disk using 256-bit AES keys unique to the Amazon Glacier service that are securely stored in separate systems under AWS control. This feature is offered at no additional cost beyond what you pay for using Amazon Glacier.

AWS Storage Gateway

The AWS Storage Gateway securely transfers your data to AWS over SSL and stores data encrypted at rest in Amazon S3 or Amazon Glacier using their respective server side encryption schemes.

Amazon EMR

S3DistCp is an Amazon EMR feature that moves large amounts of data from Amazon S3 into HDFS, from HDFS to Amazon S3, and between Amazon S3 buckets. S3DistCp supports the ability to request Amazon S3 to use server-side encryption when it writes EMR data to an Amazon S3 bucket you manage. This feature is offered at no additional cost beyond what you pay for using Amazon S3 to store your Amazon EMR data.

Oracle on Amazon RDS

You can choose to license the Oracle Advanced Security option for Oracle on Amazon RDS to leverage the native Transparent Data Encryption (TDE) and Native Network Encryption (NNE) features. The Oracle encryption module creates data and key-encrypting keys to encrypt the database. The key-encrypting keys specific to your Oracle instance on Amazon RDS are themselves encrypted by a periodically-rotated 256-bit AES master key. This master key is unique to the Amazon RDS service and is securely stored in separate systems under AWS control.

Microsoft SQL Server on Amazon RDS

You can choose to provision Transparent Data Encryption (TDE) for Microsoft SQL Server on Amazon RDS. The SQL Server encryption module creates data and key-encrypting keys to encrypt the database. The key-encrypting keys specific to your SQL Server instance on Amazon RDS are themselves encrypted by a periodically-rotated, regional 256-bit AES master key. This master key is unique to the Amazon RDS service and is securely stored in separate systems under AWS control. . This feature is offered at no additional cost beyond what you pay for using Microsoft SQL Server on Amazon RDS.

Amazon Redshift

When creating an Amazon Redshift cluster you can optionally choose to encrypt all data in user-created tables. There are two options to choose from for server-side encryption of an Amazon Redshift cluster.

1. In the first option, data blocks (included backups) are encrypted using random 256-bit AES keys. These keys are themselves encrypted using a random 256-bit AES database key. This database key is itself encrypted by a 256-bit AES cluster master key that is unique to your cluster. The cluster master key is encrypted with a periodically-rotated regional master key unique to the Amazon Redshift service that is securely stored in separate systems under AWS control. This feature is offered at no additional cost beyond what you pay for using Amazon Redshift.
2. With the second option, the 256-bit AES cluster master key used to encrypt your database keys is generated in your AWS CloudHSM or using a SafeNet Luna HSM appliance on-premises. This cluster master key is then encrypted by a master key that never leaves your HSM. When the Amazon Redshift cluster starts up, the cluster master key is decrypted in your HSM and used to decrypt the database key, which is sent securely to the Amazon Redshift hosts to reside only in memory for the life of the cluster. If the cluster ever restarts, the cluster master key is again retrieved securely from your HSM—it is never stored on disk in plaintext. This option lets you more tightly control the hierarchy and lifecycle of the keys used to encrypt your data. This feature is offered at

no additional cost beyond what you pay for using Amazon Redshift (and AWS CloudHSM if you choose that option for storing keys).

In addition to encrypting data generated within your Amazon Redshift cluster, you can also load encrypted data into Amazon Redshift from Amazon S3 that was previously encrypted using the Amazon S3 encryption client and keys you provide. Amazon Redshift supports the decryption and re-encryption of data going between Amazon S3 and Amazon Redshift to protect the full lifecycle of your data.

These server-side encryption features across multiple AWS services enable you to easily encrypt your data simply by making a configuration setting in the AWS Management Console, CLI, or API request for the given AWS service. The authorized use of encryption keys is automatically and securely managed by AWS. Because unauthorized access to those keys may lead to the disclosure of your data, we have built systems and processes that minimize the chance of unauthorized access.

Conclusion

We have presented three different models for how encryption keys are managed and where they are used. If you take all responsibility for the encryption method and the KMI, you can have granular control over how your applications encrypt data. However, that granular control comes at a cost—both in terms of deployment effort and an inability to have AWS services tightly integrate with your applications' encryption methods. At the other end of the spectrum AWS offers fully-integrated check box encryption for several services that store your data.

Table 1 below summarizes the available options for encrypting data at rest across AWS. We recommend you determine which encryption and key management model is most appropriate for your data classifications in the context of the AWS service you are using.

	Encryption Method and KMI			
	Model A		Model B	Model C
AWS Service	Client-Side Solutions Using Customer-Managed Keys	Client-Side Partner Solutions with KMI for Customer-Managed Keys	Client-Side Solutions for Customer-Managed Keys in AWS CloudHSM	Server-Side Encryption Using AWS-Managed Keys
Amazon S3	Bouncy Castle, OpenSSL, Amazon S3 encryption client in the AWS SDK for Java	SafeNet ProtectApp for Java	Custom Amazon VPC-EC2 application integrated with AWS CloudHSM client	Amazon S3 server-side encryption
Amazon Glacier	N/A	N/A	Custom Amazon VPC-EC2 application integrated with AWS CloudHSM client	All data is automatically encrypted using server-side encryption
AWS Storage Gateway	Linux Block Level: - Loop-AES, dm-crypt (with or without LUKS), and TrueCrypt Linux File System: - eCryptfs and EncFs Windows Block Level: -TrueCrypt Windows File System: - BitLocker	Trend Micro SecureCloud, SafeNet StorageSecure	N/A	Amazon S3 server-side encryption
Amazon EBS	Linux Block Level: - Loop-AES, dm-crypt+LUKS and TrueCrypt Linux File System: - eCryptfs and EncFs Windows Block Level: -TrueCrypt Windows File System: - BitLocker, EFS	Trend Micro SecureCloud, SafeNet ProtectV	Custom Amazon VPC-EC2 application integrated with AWS CloudHSM client	N/A
Oracle on Amazon RDS	Bouncy Castle, OpenSSL	CipherCloud Database Gateway and Voltage SecureData	Custom Amazon VPC-EC2 application integrated with AWS CloudHSM client	Transparent Data Encryption (TDE) and Native Network Encryption (NNE) with optional Oracle Advanced Security license TDE for Microsoft SQL Server
Microsoft SQL Server on Amazon RDS	Bouncy Castle, OpenSSL	CipherCloud Database Gateway and Voltage SecureData	Custom Amazon VPC-EC2 application integrated with AWS CloudHSM client	N/A
Amazon Redshift	N/A	N/A	Encrypted Amazon Redshift clusters with your master key managed in AWS CloudHSM or on-premises Safenet Luna HSM	Encrypted Amazon Redshift clusters with AWS-managed master key
Amazon EMR	eCryptfs	Gazzang CloudEncrypt	Custom Amazon VPC-EC2 application integrated with AWS CloudHSM client	S3DistCp using Amazon S3 server-side encryption to protect persistently stored data

Table 1: Summary of Data at Rest Encryption Options

References and Further Reading

AWS Resources

- Client-Side Data Encryption with the AWS SDK for Java and Amazon S3
  http://aws.amazon.com/articles/2850096021478074
- AWS CloudHSM
  https://aws.amazon.com/cloudhsm/
- Amazon EMR S3DistCp to encrypt data in Amazon S3
  http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/UsingEMR_s3distcp.html
- Transparent Data Encryption for Oracle on Amazon RDS
  http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Appendix.Oracle.Options.html#Appendix.Oracle.Options.AdvSecurity
- Transparent Data Encryption for Microsoft SQL Server on Amazon RDS
  http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_SQLServer.html#SQLServer.Concepts.General.Options
- Amazon Redshift encryption
  http://aws.amazon.com/redshift/faqs/#0210
- AWS Security Blog
  http://blogs.aws.amazon.com/security

Partner Resources

- Bouncy Castle Java crypto library
  http://www.bouncycastle.org/
- OpenSSL crypto library
  http://www.openssl.org/
- CloudBerry Explorer PRO for Amazon S3 encryption
  http://www.cloudberrylab.com/amazon-s3-explorer-pro-cloudfront-IAM.aspx
- SafeNet encryption products for Amazon S3, Amazon EBS, and AWS CloudHSM
  http://www.safenet-inc.com/
- Trend Micro SecureCloud
  http://www.trendmicro.com/us/enterprise/cloud-solutions/secure-cloud/index.html
- CipherCloud for AWS and CipherCloud for Any App
  http://www.ciphercloud.com/
- Voltage Security SecureData Enterprise
  http://www.voltage.com/products/securedata-enterprise/
- Gazzang CloudEncrypt
  http://www.gazzang.com/products/cloudencrypt