SwiftStack

Data-at-Rest Encryption

At-rest encryption in SwiftStack encrypts all object data, object etags (checksums), and any user metadata values set on objects. The feature is enabled by a cluster operator and is completely transparent to the end-user. Internally-encrypted data is never returned to clients via the API.

All security features start with defining the threat model (i.e. the circumstances you're trying to protect against). For the at-rest encryption feature in SwiftStack, the threat model is rather straightforward: 1) protect object data and user metadata from being exposed if a data drive leaves the cluster 2) inter-cluster data in flight is encrypted.

There are two common cases where data drives might leave a cluster. The first is by accident: an inventory error could misplace a drive taken out of a cluster and put it into a different server. If the drive isn't erased, the data could be exposed.

The second is when hard drives fail, which is a common reality. The drive vendor's "return merchandise authorization" (RMA) process could result in that user data being exposed to unauthorized parties. The at-rest encryption feature in Swift is designed to protect against this sort of data disclosure. When using encryption in SwiftStack, the cluster operator can confidently RMA drives.

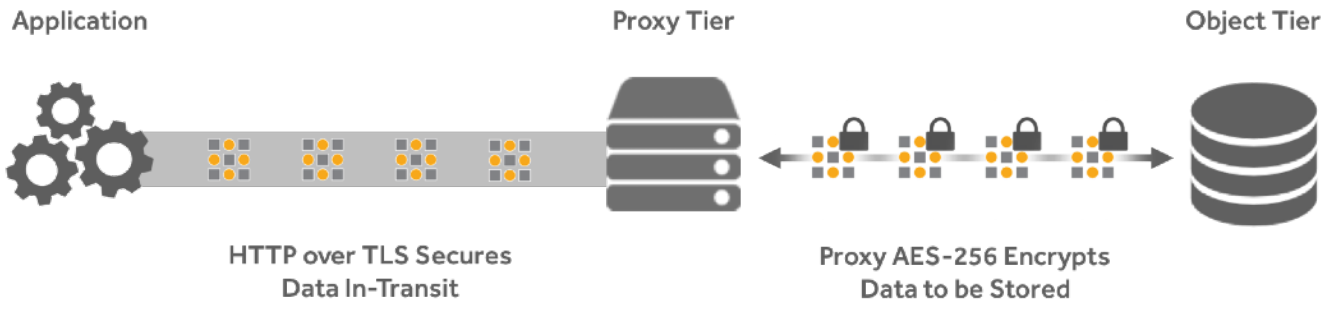
HIGHLIGHTS

- Encryption of object data and user metadata on disk
- AES-256 in counter mode
- Encryption option at a cluster level
- Encryption handled by the proxy role
- Inter-cluster data in flight is encrypted

USES

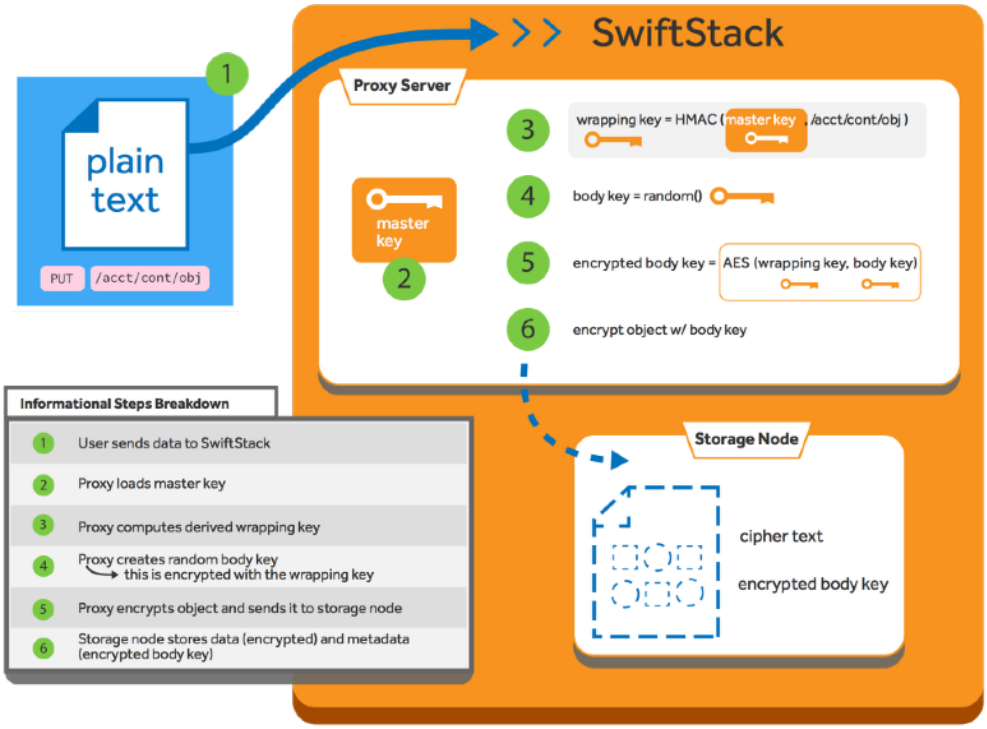
- Protects against data exposure if a drive physically leaves the cluster
- Allows drives to go through an RMA process without data exposure

Additionally, by encrypting data in the proxy tier, all inter-cluster communication is encrypted. While these are typically configured as private networks, this reduces the exposure if this network traffic is compromised. With the combination of client-facing TLS, this provides a mechanism for data to be encrypted at any point the data is on a network.



How it works

SwiftStack has a two-tier architecture: there's a proxy server that handles most of the API and coordinates requests to the storage nodes, and there's the storage nodes which actually persist the data. SwiftStack's encryption is implemented completely in the proxy server. If you're deploying separate physical proxy and storage servers, this design allows for all of the crypto knowledge to stay on the proxy server and not even be on the servers where the data is persisted at all.



Each object stored in SwiftStack is encrypted with it's own unique, randomly-chosen key. Internally, this is called the "body key". This randomly chosen body key is itself encrypted with the object's derived key. This is a technique called "key wrapping" where one key is encrypted with another key. The derived key is the HMAC (keyed-hash message authentication code) of the cluster's master key and the full path to the object. The cluster's master key is available to the proxy server. It's very important that this master key is protected from untrusted parties.

Since key wrapping is used, it means that any eventual support for rotating encryption keys will not require re-encrypting the entire object contents. A SwiftStack cluster will only have to re-encrypt the body key. Although key rotation is not supported yet, the foundation for it exists in this initial version.

SwiftStack uses AES in counter mode because it has the property that byte offsets in the plain text and the cipher text are the same. This allows range requests to encrypted data to be easily supported. One of the goals of at-rest encryption in SwiftStack is that it be completely transparent to the end user. All existing API functions should work in exactly the same way for both encrypted data and non-encrypted data.