==============
Key Management
==============

Problem Space - What is key management, why is it needed // Introductions
We protect sensitive information in cloud deployments using various applications of cryptography. Simple examples are encrypting data at rest or signing an image to prove that it has not been tampered with. In all cases these cryptographic capabilities require some sort of *key material* in order to operate.

Key Management, often referred to as “secrets management” describes a group of technologies that are designed to protect key materials within a software system. Traditionally key management involves deployment of Hardware Security Modules (HSM). These are typically devices that have been physically hardened against tampering. Physical HSM are often expensive to deploy and difficult to scale.

As technology has advanced, the number of “secret things” that need to be protected has increased beyond key materials to include certificate pairs, API keys, system passwords, signing keys etc. This increase has created a need for a more scalable approach to key management and resulted in the creation of a number of software services that provide scalable dynamic key management. In this chapter we briefly describe the services that exist today and focus on those that are able to be integrated into OpenStack clouds.

Summary of existing technologies

Fixed/Hardcoded keys - It is known that some services have the option to specify keys in their configuration files. This is the least secure way to operate and is not recommended for any sort of production operation.
Hardware security modules - HSMs can come in multiple forms. The traditional device is a rack mounted appliance such as a `here
<https://vakwetu.wordpress.com/2015/11/30/barbican-and-dogtagipa/>`_.
Software Key Managers - [Vault, KeyWhiz, Barbican, Citadel, Confidant, Conjur, EJSON, Knox, Red October]

  KeyWhiz
  Vault
  Castellan
  Barbican

Integration with other projects
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Barbican is integrated with several OpenStack features, either directly or as a back end of Castellan.

Image signature verification

Verification of image signatures assures the user that uploaded them that the image has not been replaced or changed. The image signature verification feature uses Castellan. The image signature and certificate UUID is uploaded along with the image to the Glance image service. Glance can verify the signature after retrieving the certificate from the key manager. When
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

the image is booted, the Nova compute service can also verify the signature after it retrieves the certificate from the key manager. For more details, see the `Trusted Images documentation <https://docs.openstack.org/security-guide/instance-management/security-services-for-instances.html#trusted-images/>`__.

Volume encryption

The volume encryption feature provides encryption of data-at-rest using Castellan. When a user creates an encrypted volume type and creates a volume using that type, the Cinder block storage service requests the key manager creates a key to be associated with that volume. When the volume is attached to an instance, Nova compute retrieves the key.
  https://docs.openstack.org/security-guide/tenant-data/data-encryption.html

https://docs.openstack.org/ocata/config-reference/block-storage/volume-encryption.html

Ephemeral disk encryption

Sahara

Sahara generates and stores several passwords during the course of operation. To harden sahara's usage of passwords it can be instructed to use an external key manager for storage and retrieval of these secrets. To enable this feature there must first be an OpenStack Key Manager service deployed within the stack.

With a Key Manager service deployed on the stack, sahara must be configured to enable the external storage of secrets. Sahara uses the castellan library to interface with the OpenStack Key Manager service. This library provides configurable access to a key manager. For detail see:

https://docs.openstack.org/developer/sahara/userdoc/advanced.configuration.guide.html#external-key-manager-usage

Magnum

Octavia/LBaaS

The LBaaS (Load Balancer as a Service) feature of Neutron and the Octavia Project need Certificates and their private keys to provide load balancing for TLS connections. They Can use Barbican to store this sensitive information.

https://wiki.openstack.org/wiki/Network/LBaaS/docs/how-to-create-tls-loadbalancer
https://docs.openstack.org/developer/octavia/guides/basic-cookbook.html#deploy-a-tls-terminated-https-load-balancer

Swift

Barbican Role Based Access Control

Secret Store Back-ends

The Key Manager service has a plugin architecture that allows the deployer to store secrets in one or more secret stores. Secret stores can be software-based, such as a software token, or hardware devices such as a hardware security module (HSM). This section describes the plugins that are currently available and discusses the security posture of each one. Plugins are enabled and configured with settings in the ``/etc/barbican/barbican.conf`` configuration File.

There are two types of plugins: crypto plugins and secret store plugins.

Crypto Plugins

Crypto plugins store secrets as encrypted blobs within the Barbican database. The plugin is invoked to encrypt the secret, on secret storage, and decrypt the secret, on secret retrieval. There are two flavors of storage plugins currently available: the Simple Crypto plugin and the PKCS#11 crypto plugin.

Simple Crypto Plugin

The simple crypto plugin is configured by default in barbican.conf. This plugin uses single symmetric key (kek - or 'key encryption key')- which is stored in plain text in the ``barbican.conf`` file to encrypt and decrypt all secrets. As such, this plugin is completely insecure and is only suitable for development and testing, it must not be used for production deployments.

PKCS#11 Crypto Plugin

The PKCS#11 crypto plugin can be used to interface with a Hardware Security Module (HSM) using the PKCS#11 protocol. Secrets are encrypted (and decrypted on retrieval) by a project specific Key Encryption Key (KEK) which resides in the HSM. Since a different KEK is used for each project, and since the KEKs are stored inside an HSM (instead of in plaintext in the configuration file) the PKCS#11 plugin is much more secure than the simple crypto plugin. It is the most popular back end amongst Barbican deployments.

Secret Store Plugins

Secret store plugins interface with secure storage systems to store the secrets within those systems. There are two types of secret store plugins: the KMIP plugin and the Dogtag plugin.

KMIP Plugin

The KMIP secret store plugin is used to communicate with a KMIP device, such as

a Hardware Security Module (HSM). The secret is securely stored in the KMIP device directly, rather than in the Barbican database. The Barbican database maintains a reference to the secret's location for later retrieval. The plugin can be configured to authenticate to the KMIP device using either a username and password, or using a client certificate. This information is stored in the Barbican configuration file.

Dogtag Plugin

The Dogtag secret store plugin is used to communicate with Dogtag. Dogtag is the upstream project corresponding to the Red Hat Certificate System, a Common Criteria/FIPS certified PKI solution that contains a Certificate Manager (CA) and a Key Recovery Authority (KRA) which is used to securely store secrets.The KRA stores secrets as encrypted blobs in its internal database, with the master encryption keys being stored either in a software-based NSS security database, or in a Hardware Security Module (HSM). The software-based NSS database configuration provides a secure option for deployments that do not wish to use a HSM. The KRA is a component of FreeIPA, therefore it is possible to configure the plugin with a FreeIPA server. More detailed instructions on how to set up Barbican with FreeIPA are provided `here <https://vakwetu.wordpress.com/2015/11/30/barbican-and-dogtagipa/>`_.

Architecture Considerations

Threat Analysis
~~~~~~~~~~~~~~~
The barbican team worked with the OpenStack Security Project to perform a security review of a best practise Barbican deployment. The objective of the security review is to identify weaknesses and defects in the design and architecture of services, and propose controls or fixes to resolve these issues.

The barbican threat analysis identified eight security findings and two recommendations to improve the security of a barbican deployment. These results can be reviewed in the security analysis repo, along with the barbican architecture diagram and architecture description page. <link>.

============
Case studies
============

Earlier in **:doc:**`../introduction/introduction-to-case-studies` we introduced the Alice and Bob case studies where Alice is deploying a private government cloud and Bob is deploying a public cloud each with different security requirements. Here we discuss how Alice and Bob would consider the appropriate key manager deployment decisions.

Alice's private cloud
~~~~~~~~~~~~~~~~~~~~~

Alice chooses the PKCS#11 crypto plugin for her barbican deployment.

She only has a small HSM that does not have enough capacity to store
all her cloud's secrets. She does not want to store the KEK in plaintext
in a configuration file, so the perfect compromise is the PKCS#11 plugin.

She further hardens her deployment by setting strict file permissions
on her barbican configuration files.

Bob's public cloud

Bob already has a deployment of Dogtag in his data center. He wants to
leverage this investment to ensure that all of his client's key are
securely protected. He chooses to use the Dogtag plugin to store his
cloud's secrets in Dogtag.

He also further hardens his deployment by setting strict file permissions
on her barbican configuration files.

.. _key_mgr_checklist:

Checklist

.. _check_key_mgr_01:

Check-Key-Manager-01: Is user/group ownership of config files set to root/barbican?

Configuration files contain critical parameters and information required
for smooth functioning of the component. If an unprivileged user, either
intentionally or accidentally, modifies or deletes any of the parameters or
the file itself then it would cause severe availability issues resulting in a
denial of service to the other end users. Thus user ownership of such critical
configuration files must be set to root and group ownership must be set to
barbican.

Run the following commands:

.. code:: console

```
$ stat -L -c "%U %G" /etc/barbican/barbican.conf | egrep "root barbican"
$ stat -L -c "%U %G" /etc/barbican/barbican-api-paste.ini | egrep "root barbican"
$ stat -L -c "%U %G" /etc/barbican/policy.json | egrep "root barbican"
```

Pass: If user and group ownership of all these config files is set
to root and barbican respectively. The above commands show output of root barbican.

Fail: If the above commands do not return any output as the user
and group ownership might have set to any user other than root or any group
other than barbican.

.. _check_key_mgr_02:

Check-Key-Manager-02: Are strict permissions set for configuration files?
~~~~~~

Similar to the previous check, it is recommended to set strict access
permissions for such configuration files.

Run the following commands:

.. code:: console

   $ stat -L -c "%a" /etc/barbican/barbican.conf
   $ stat -L -c "%a" /etc/barbican/barbican-api-paste.ini
   $ stat -L -c "%a" /etc/barbican/policy.json

Pass: If permissions are set to 640 or stricter. The permissions of 640
translates into owner r/w, group r, and no rights to others i.e. "u=rw,g=r,o=".
Note that with :ref:`check_key_mgr_01` and permissions set to 640, root has
read/write access and barbican has read access to these configuration files. The
access rights can also be validated using the following command. This command
will only be available on your system if it supports ACLs.

.. code:: console

   $ getfacl --tabular -a /etc/barbican/barbican.conf
   getfacl: Removing leading '/' from absolute path names
   # file: etc/barbican/barbican.conf
   USER   root rw-
   GROUP barbican r--
   mask   r--
   other   ---

Fail: If permissions are not set to at least 640.

.. _check_key_mgr_03:

Check-Key-Manager-03: Is OpenStack Identity used for authentication?
~~~~~~

OpenStack supports various authentication strategies like noauth and keystone.
If the '``noauth``' strategy is used then the users could interact with
OpenStack services without any authentication. This could be a potential risk

since an attacker might gain unauthorized access to the OpenStack components. Thus it is strongly recommended that all services must be authenticated with keystone using their service accounts.

Pass: If the parameter ``authtoken`` is listed under the ``pipeline:barbican_api`` section in ``barbican-api-paste.ini``.

Fail: If the parameter ``authtoken`` is missing under the ``pipeline:barbican_api`` section in ``barbican-api-paste.ini``.

.. _check_key_mgr_04:

Check-Key-Manager-04: Is TLS enabled for authentication?
~~~~~~

OpenStack components communicate with each other using various protocols and the communication might involve sensitive or confidential data. An attacker may try to eavesdrop on the channel in order to get access to sensitive information. Thus all the components must communicate with each other using a secured communication protocol.

Pass: If value of parameter ``auth_protocol`` under ``[keystone_authtoken]`` section in ``barbican.conf`` is set to ``https``, or if value of parameter ``identity_uri`` under ``[keystone_authtoken]`` section in ``barbican.conf`` is set to Identity API endpoint starting with ``https://`` and value of parameter ``insecure`` under the same ``[keystone_authtoken]`` section in the same ``barbican.conf`` is set to ``False``.

Fail: If value of parameter ``auth_protocol`` under ``[keystone_authtoken]`` section in ``barbican.conf`` is set to ``http``, or if value of parameter ``identity_uri`` under ``[keystone_authtoken]`` section in ``barbican.conf`` is not set to Identity API endpoint starting with ``https://`` or value of parameter ``insecure`` under the same ``[keystone_authtoken]`` section in the same ``barbican.conf`` is set to ``True``.
~~~~~~