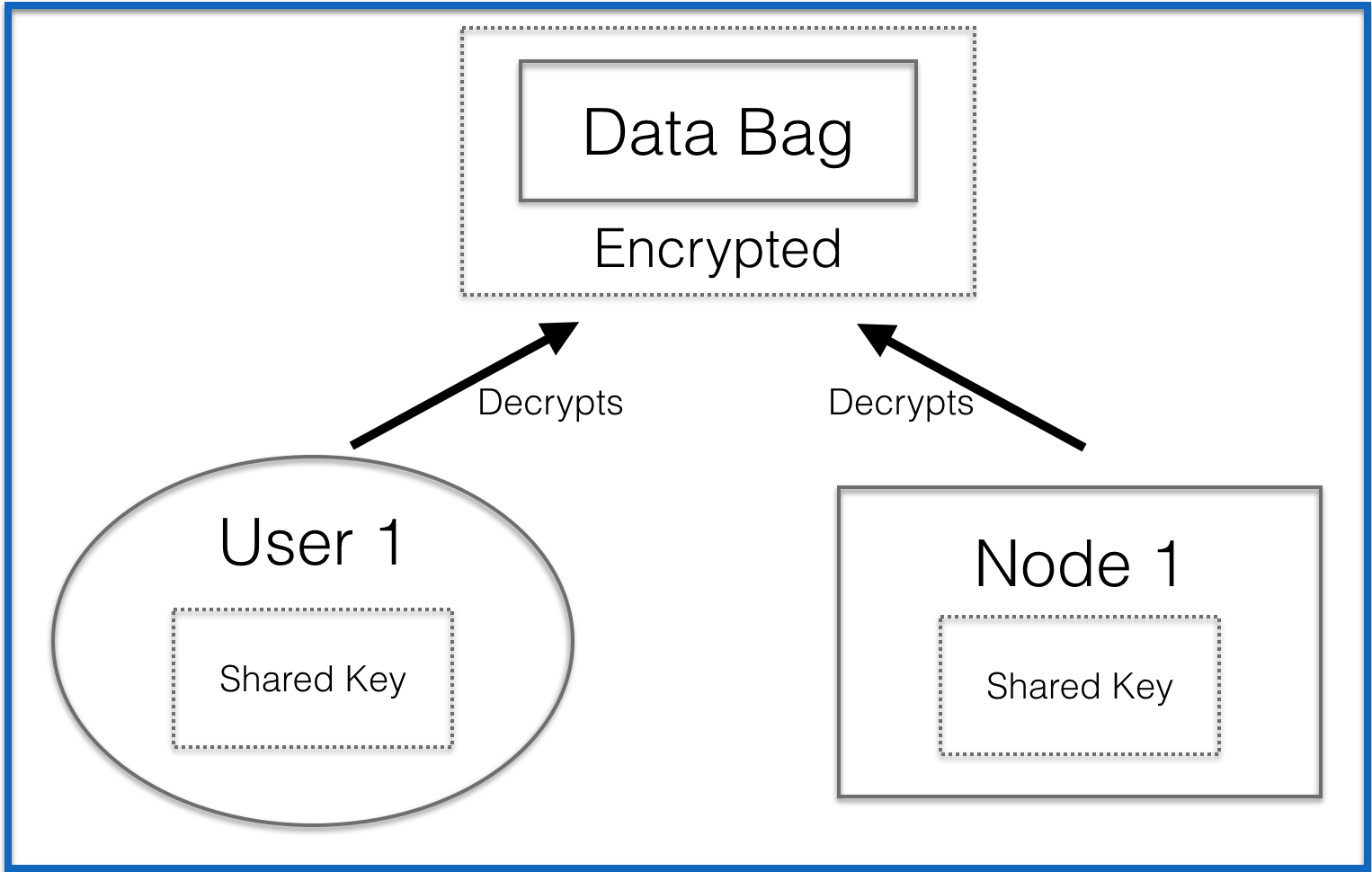
**Chef Vault(Security like Databag)**

to deal with secrets – i.e. API keys, database passwords, and more – within a configuration management framework? Although data bags are incredibly useful for storing global variables to use in your Chef recipes, they do not always provide adequate security for sensitive information

One solution is to use [encrypted databag items](https://docs.chef.io/data_bags.html#encrypt-a-data-bag-item). In this case you would encrypt a data bag item containing sensitive information with a shared key. Only those people or pieces of infrastructure with the shared key would be able to decrypt the data bag.



The problem with this approach is the use of the shared key –  
you would be responsible for distributing this key to the appropriate users and pieces of infrastructure. This invites security issues and is definitely not a best practice.

To avoid this security issues of a shared key, you can use Chef Vault.

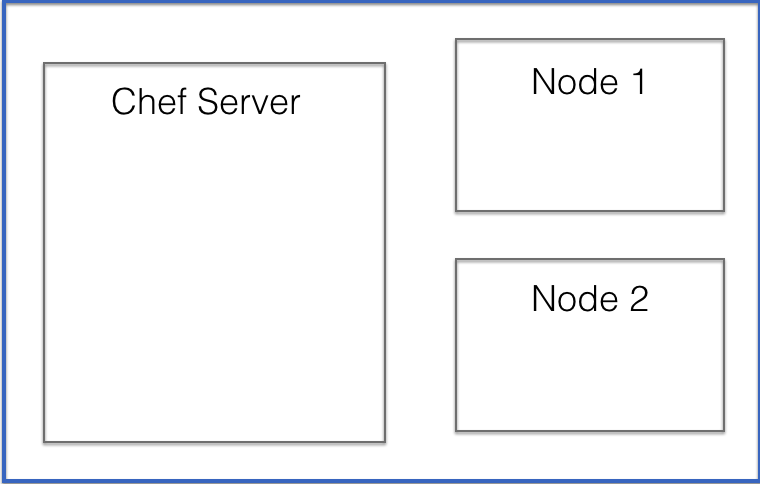
# What is Chef Vault?

Chef Vault is an alternative tool for managing secrets with Chef. It was developed as an open source project originally by Nordstrom, but transferred to Chef in 2015. Unlike encrypted data bags, it does not require the use of one shared key by all users and pieces of infrastructure.

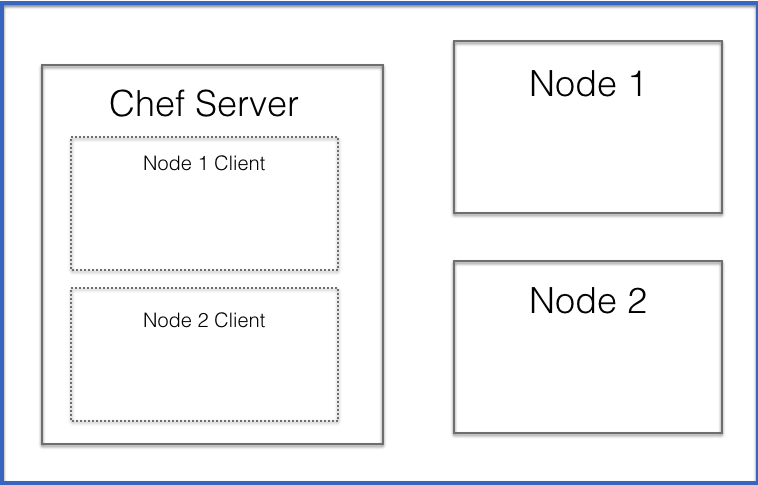
# How Does Chef Vault Work?

The [THEORY.md](https://github.com/chef/chef-vault/blob/master/THEORY.md) file in the Chef Vault Github repo has detailed information on the how and why Chef Vault works the way  
it does. Let’s go over some of this now.

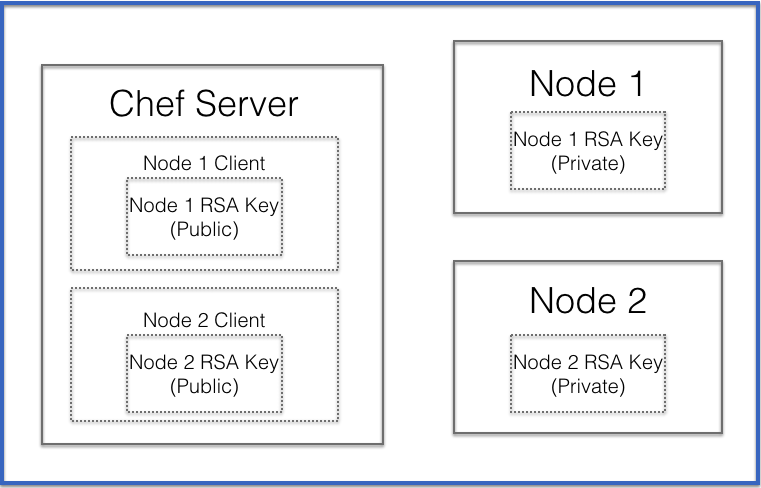
Let’s say you are managing a set of nodes through Chef Server



Every node managed through a Chef Server has an associated client object on the Chef Server. This is how Chef Server is aware of and managing the node.

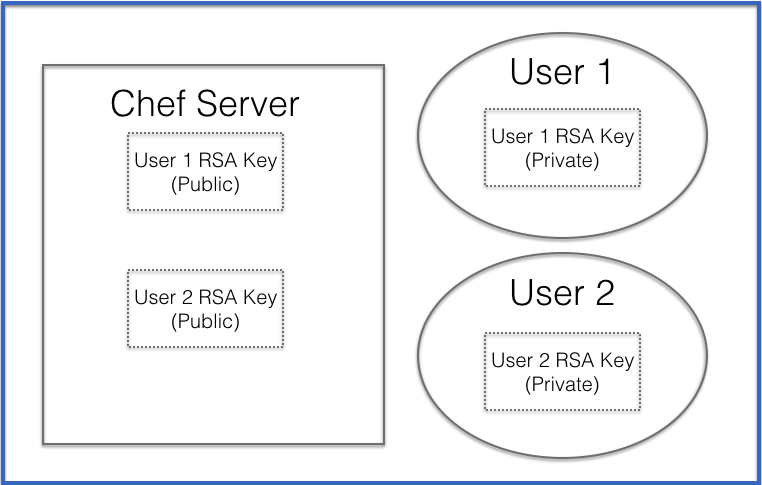


When a node is managed through Chef Server it uses an RSA keypair. The node retains the private half of the RSA keypair (usually it’s in /etc/chef/client.pem). The Chef Server  
holds the public half of the keypair.

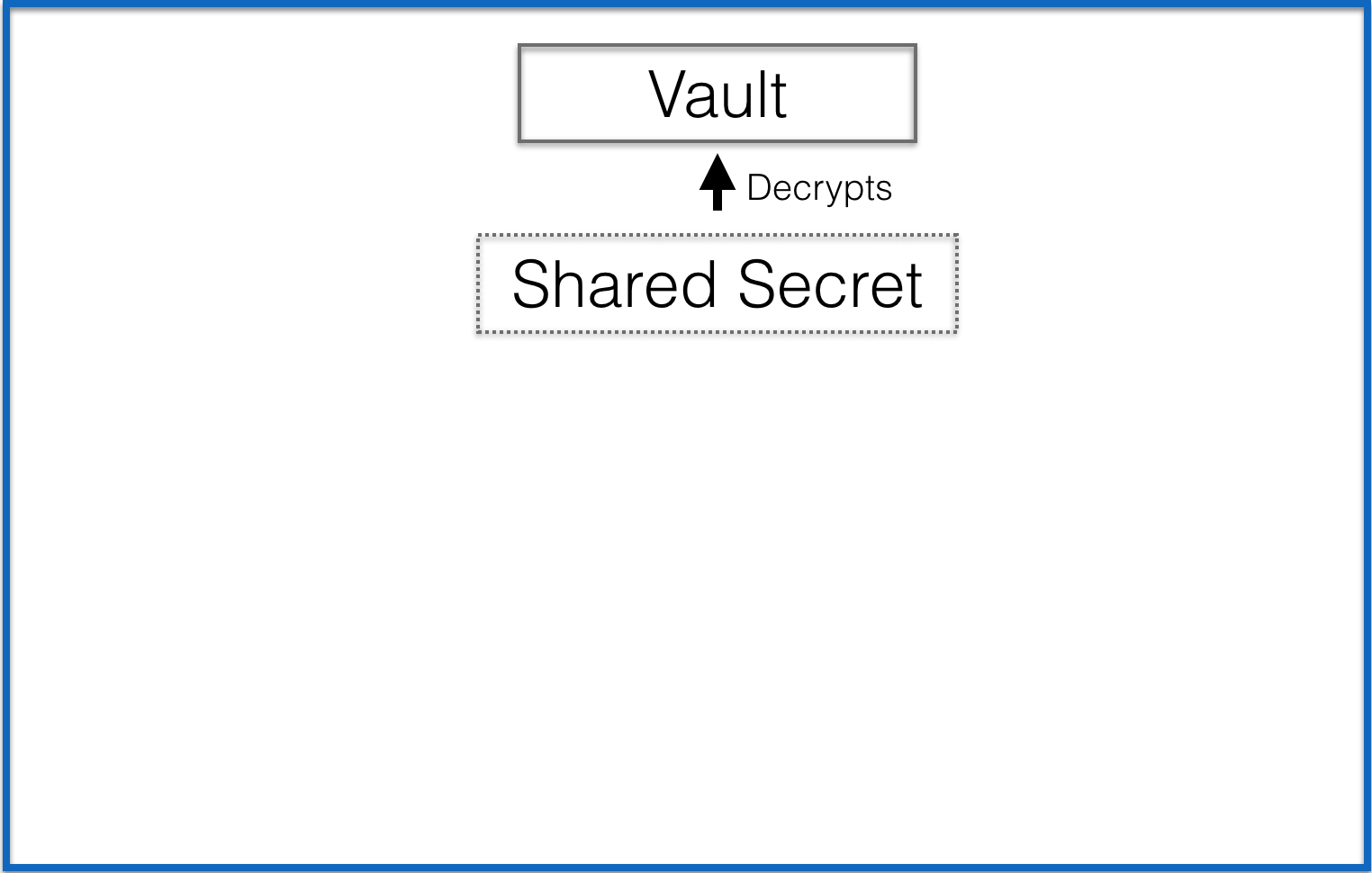


Additionally, every user account on the Chef Server also has an RSA key pair. This is how you authenticate/authorize your workstation to make modifications to the Chef Server.

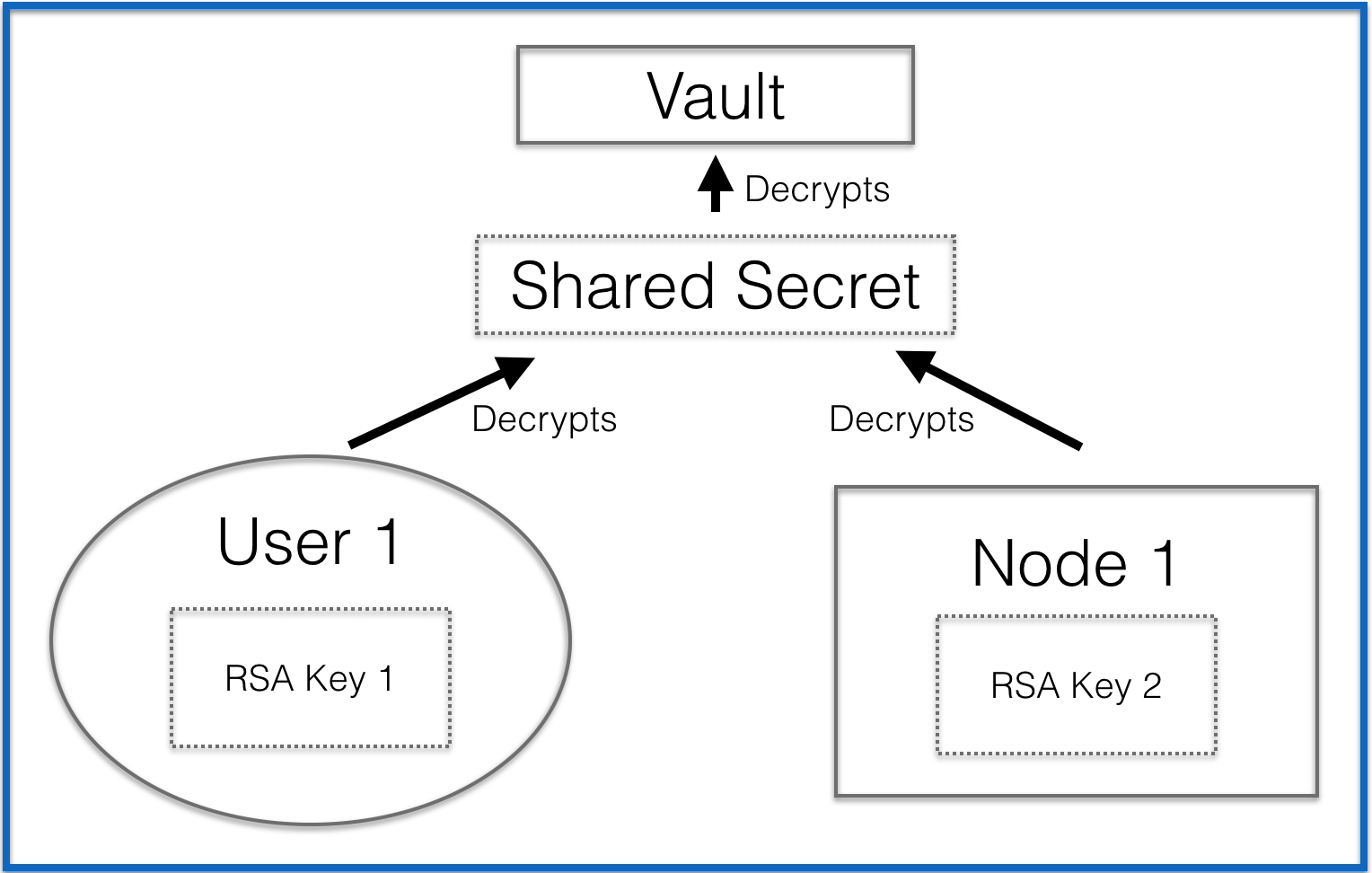
You workstation will retain the private half of the key (often this is in a .chef directory) and the Chef Server will hold the public half of the keypair.



When you create an encrypted data bag with Chef Vault (hereafter referred to as a vault), it will encrypt the data bag item (hereafter referred to as the vault item) with a randomly generated secret string known as the “shared secret.”



When you create a vault, you also specify the users and nodes who will need access to the vault. Chef Vault will then encrypt a copy of the “shared secret” for each user and node using that user or node’s public key on the Chef Server. So if you have two nodes and two admin users, Chef vault with encrypt four copies of the “shared secret” – one for each node or user using the respective node/user’s public key.



When you access an encrypted item in a vault you are using 2 layers of encryption – your RSA key decrypts the shared secret, then the shared secret decrypts the encrypted vault item item.

This makes it easy to add an admin user or node without needing to find and distribute the shared key. It also means when a user or node’s access needs to be removed, you only have to remove that user or node’s key, you don’t need to create a new key and distribute it to everyone.

# Using Chef Vault

## Installing Chef Vault

If you are using the Chef DK…congratulations, you already have Chef Vault installed! If you are not using the Chef DK, you can install Chef Vault with

$ gem install chef-vault

## Creating a Vault

Let’s create a vault. Let’s say you want to store a database password. If you prefer to create data bag items from json files, you can create a json file like this.

database.json

{“db\_password”:”some\_password”}

Finally, let’s define who/what you want to be able to decrypt this vault item. Let’s say you have two users who need to be able to decrypt this item from their workstations – starley and jsnow. You also have two nodes that need to be able to decrypt this item named whitewalker\_node\_1 and whitewalker\_node\_2.

You would then run this command:

$ knife vault create credentials database -A starley, jsnow -M client -S ‘name:whitewalker\_node\_\*’ -J ./database.json

Let’s break this down.

$ knife vault create credentials database -A starley, jsnow -M client -S ‘name:whitewalker\_node\_\*’ -J ./database.json

# credentials is the name of the vault

# database is the name of item within the vault which will be encrypted

# starley and jsnow are the users we want to be able to decrypt the encrypted item from their workstations (hereafter referred to as admin users)

# We also specify a mode for Chef Vault. If you are using it with a Chef server, then you will need to specify it as -M client. If you are using it with Chef Solo, you will need to specify it as -M solo

# You define nodes that need to decrypt this item by doing a search of nodes associated with your organization on the chef server, in this case for ‘name:whitewalker\_node\_\*’ This will find all nodes with names that begin with ‘whitewalker\_node\_’

# Finally, since you are defining the content for this vault item in a JSON file, you include the filename and path to that file

NOTE: After posting this, I received feedback that searching for nodes using the -S tag can be insecure. This is because any node which has been compromised can potentially fake its own data and be picked up by the search. If this happens, the next time someone runs knife vault refresh, the compromised node could be illicitly given decryption access.

To be fully secure, you must use

-A "whitewalker\_node1,whitewalker\_node2"

Or, when combined with adding users as well, it would be

-A "starley,jsnow,whitewalker\_node1,whitewalker\_node2

When this command is run, it will create a vault named ‘credentials’ and TWO items within that data bag – ‘database’ and ‘database\_keys’.

The ‘database’ item contains the actual encrypted content. If we were to look at this item from a user’s workstation who was NOT an admin user for the vault, we would see only the encrypted output.

$ knife vault show credentials database

db\_password:

cipher: aes-256-cbc

encrypted\_data: dsiBtNHX8Sbis42yKuYBvbdNXPpu8bQfJrS20op7zoysfR8roFlzpVHyoaG2

4yb3

iv: +0siNLzFHHqStM07k5JhYw==

version: 1

id: database

If we are on a user’s workstation who IS an admin user (pretend we are on jsnow’s workstation) we would see the decrypted content.

$ knife vault show credentials database

db\_password: some\_password

id: database

Now let’s take a look at that ‘database\_keys’ item

$ knife data bag show credentials database\_keys

admins:

starley

jsnow

clients:

whitewalker\_node\_1

whitewalker\_node\_2

id: database\_test\_keys

starley: SOME KEY

jsnow: SOME KEY

whitewalker\_node\_1: SOME KEY

whitewalker\_node\_2: SOME KEY

This contains 4 keys (one for each user and node). Remember when we created the vault, it generated a shared secret, then it encrypted that shared secret using the public key for each node and user (which it gets from the Chef Server).

Now, when one of these users or nodes attempts to decrypt the encrypted item, the user/node’s private key will decrypt the shared secret, which will then decrypt the item itself. So they will be able to see the contents of the ‘database’ item within the ‘credentials’ vault.

## Using a Vault in a Chef Recipe

Now let’s explore using the vault within a Chef Cookbook. In order to use Chef Vault within our cookbook, we will use the Chef Vault Cookbook. Although you CAN use Chef Vault without this cookbook, the cookbook adds many helpful features which make interacting with vaults much easier.

To do this, make sure to add this line to your metadata.rb

metadata.rb

depends ‘chef-vault’

Then, in your recipe where you will use chef-vault, make sure to include the chef-vault recipe.

recipe.rb

include\_recipe ‘chef-vault’

Now let’s call the encrypted data bag item within the vault

recipe.rb

include\_recipe ‘chef-vault’

vault = chef\_vault\_item(:credentials, ‘database’)

And now we can use an attribute from within that encrypted data bag item.

recipe.rb

include\_recipe ‘chef-vault’

vault = chef\_vault\_item(:credentials, ‘database’)

node.set[‘database’][‘password’] = vault[‘password’]

## Editing a Vault

What if you needed to edit a vault, perhaps to change the database password? This can be done (provided you are using the workstation of an admin user of the vault) through this command.

$ knife vault edit credentials database

Remember, credentials is the name of the vault itself, and database is the encrypted item within that vault.

## Destroying a Vault

**Destroying the Encrypted Item**

To destroy an encrypted item within a vault, use the knife vault delete command.

$ knife vault delete credentials database

**Destroying the Data Bag (Vault)**

There is not currently a way to delete the vault itself through the knife vault commands. However, you CAN delete a vault using a knife data bag command.

$ knife data bag delete credentials

# Managing Chef Vault

## Users

**Adding a new Admin User**

If you need to add a new admin user, run knife vault update with the vault name, the encrypted item within the vault, and -A flag, and the user’s username on the Chef Server.

$ knife vault update credentials database -A “new-username”

This will encrypt a new copy of the shared secret with the user’s key on the Chef Server. After this is run, the user will be able to decrypt the vault.

**Updating an Admin User’s Key**

If an admin user changes their key associated with the Chef Server, you will need to refresh the encrypted copies of the shared secret (the string that is used to decrypt the vault item). You can do this by running:

$ knife vault refresh credentials database

**Removing an Admin User**

If you need to remove an admin user, use the knife vault remove command with the user’s Chef Server username.

$ knife vault remove credentials database -A “role:base”

## Nodes

**Adding a New Node**

To give a new node the ability to decrypt the vault item, run knife vault update. Make sure to use the -S flag and include a query that will include the new node (i.e. ‘role:base’ if the node (AND the nodes that already have access to the vault item) has the base role).

$ knife vault update credentials database -S “search-query-for-nodes”

If you are adding a new node (say a node that has been added to the base role) but are using the SAME search query as before (i.e. ‘role:base’), you can run knife vault refresh. This will re-run the search query and add or remove nodes as needed.

**Removing a Node**

$ knife vault update credentials database -S “search-query-for-nodes that excludes node to remove”

# Limits of Chef Vault

Chef Vault is limited when it comes to auto scaling or self-healing systems.

“Once a server is granted access to a secret, it can continue to access it in an online manner. However granting or revoking new servers requires human interaction. This means chef-vault is incompatible with auto-scaling or self-healing systems. It also inherits the same issues with audit logging as all other data-bag driven approaches.”

Consult his blog post for information on alternatives tools available to manage secrets when using Chef.