we need to externalize our some of the sensitive config properties such as DB username and password in a properties file.As long as the properties file are at classpath, we are good to go. But while dealing with microservices and cloud config we have our properties stored at some external store such as GIT store. In this case we can't just keep these sensitive properties in a plain-text.Hence, spring cloud config provides ways to add encryption to these properties and while serving these properties it automatically decrypts it and returns the plain-text. Hence, it is highly recommended to use HTTPS/TSL for this.

We have our GIT store at https://github.com/only2dhir/config-repo and we have 2 spring boot applications. One is cloud config server and the other is cloud config client. Now we will define our config properties in encrypted form and push to GIT.

test.local.property=test local property changed twice

random.property=random property changed twice

encrypted.property={cipher}c789b2ee5bdfd2ab95b7b13d115596485408a82bfb3ba599199b356673a0b949

**While using encrypted property it is required to store the properties in the format {cipher} to GIT**. This indicates spring cloud config that the value is in encrypted format and it should be decrypted first before serving it to the client.

Property Config Encryption and Decryption

Now let us set up the spring boot app for encryption and decryption of config properties. Since we are using symmetric encryption, we only require to tell the spring about the secret key we are using for encryption and decryption in application.properties.

**application.properties**

server.port=8888

spring.cloud.config.server.git.uri=<**Put your git repo here>**

encrypt.key=secretkey

Now after adding this property, [symmetric encryption](http://www.devglan.com/corejava/java-aes-encypt-decrypt) is automatically enabled and the config property values with format {cipher} will be automatically decrypted and served by spring cloud config.

For asymmetric encryption the value of encrypt.key should be a PEM-encoded string value or we can configure our custom keystore to use. Following is a sample to generate a keystore.

keytool -genkeypair -alias mytestkey -keyalg RSA \

-dname "CN=Web Server,OU=Unit,O=Organization,L=City,S=State,C=US" \

-keypass changeme -keystore server.jks -storepass letmein

Now, once the server.jks is generated,we can configure it as below in application.properties.

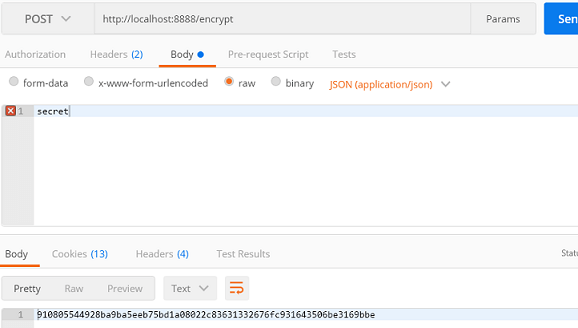
encrypt.key-store.location=classpath:/server.jks

encrypt.key-store.password=letmein

encrypt.key-store.alias=mytestkey

encrypt.key-store.secret=changeme

Now, the question comes how can we generate the encrypted text to put in the GIT store. To do so, cloud config provides /encrypt endpoint. Following is an example to generate the encrypted key and the same key in the response is placed in the GIT store.

[](https://imgur.com/Y7cyJyC)

You can also use /encrypt/{name}/{profiles} to encrypt profile specific properties.

The encryption algorithm is not strong enough

With spring cloud config server 2, we get error response for endpoint /encrypt as **The encryption algorithm is not strong enough**. This error occurs if you don't have server.key property defined. Even if this is defined, to avoid this error, the property **encrypt.key** should be placed in bootstrap.properties rather then **application.properties**

Consuming Encrypted Property Config

TO consume this encrypted property values, there is no change required at the client side as spring will serve these properties after decryption by itself. We have following configuration in application.properties and bootstrap.yml

**application.properties**

server.port=8080

security.basic.enabled=false

management.security.enabled=false

**bootstrap.properties**

spring.application.name=spring-cloud-config-client

spring.profiles.active=local

spring.cloud.config.uri=http://localhost:8888

Now, we have following controller defined for demonstration.

**DemoController.java**

package com.devglan.springcloudconfigclient.controller;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.cloud.context.config.annotation.RefreshScope;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

@RefreshScope

public class DemoController {

@Value("${encrypted.property}")

private String testProperty;

@Value("${test.local.property}")

private String localTestProperty;

@RequestMapping("/")

public String test() {

StringBuilder builder = new StringBuilder();

builder.append("global property - ").append(testProperty).append(" || ")

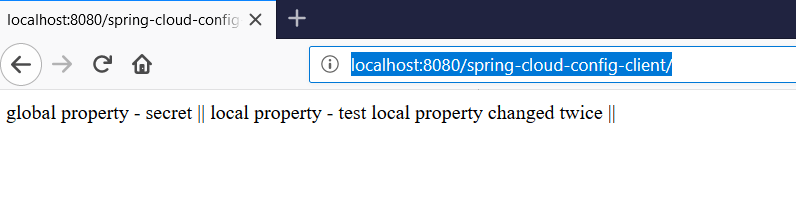
.append("local property - ").append(localTestProperty).append(" || ");

return builder.toString();

}

}

encrypted.property is the config property that is encrypted in the GIT store. Now, we can hit http://localhost:8080/spring-cloud-config-client/ to check the result.

[](https://imgur.com/6jC5Tua)