Secrets management for CMSWEB

Aroosha Pervaiz[1], Muhammad Imran[2], Valentin Y Kuznetsov[3], Panos Paparrigopoulos[1], Spyridon Trigazis[1] and Andreas Pfeiffer[1]

[1] CERN, Geneva, Switzerland
[2] NCP, Islamabad, Pakistan
[3] Cornell University, New York, U.S.A.

Abstract. Secrets are key-value pairs that are used for authentication or authorization, such as certificates, database credentials, tokens, and API keys. A robust secret management strategy is imperative to ensure agile development and to protect business-critical functions. In the previous CMSWEB (the portfolio of CMS internal IT services) infrastructure, the operators managed and maintained all the secrets, which implied that we could get locked out from deploying them if the relevant person was unavailable. To have a proper workflow for managing access to the secrets, we explored various strategies such as HashiCorp Vault, GitHub Credential Manager(GCM), and SOPS/age. In this paper, we investigate these strategies and perform a feasibility analysis. We also discuss why CMS adopted SOPS with age as a solution and how the other experiments can potentially adopt our solution. Along with this, we provide bash scripts for keys generation, encryption/decryption, and secrets deployment in CMSWEB k8s infrastructure.

1. Introduction

A growing amount of sensitive data, also called secrets, which includes API keys, database credentials, certificates, and passwords, ends up being exposed on the Internet. According to a report published by GitGuardian in 2022 [1], nearly 3 out of 1000 Git commits contained at least one secret. Overall, they accumulated over 6 million secrets from the public GitHub repositories, implying a two-fold increase from 2020. When it comes to Kubernetes [2], it does not encrypt secrets by default. An adversary with an Application Programming Interface (API) or *etcd* (a consistent, highly available key-value store) access can read or modify them. If someone has a write-access to a namespace, they can also use this privilege to read any secret in that particular namespace. Kubernetes provides an optional base64 encoding which does not protect against intrusions. This necessitates the implementation of a robust secret management system. Secrets management, in particular, refers to the tools and practices used to manage digital authentication mechanisms.

CMSWEB is a Kubernetes-based infrastructure that hosts multiple services for the operational needs of the Compact Muon Solenoid (CMS) experiment at CERN [3]. These mission-critical services include CouchDB [4], Crab [5], DBS [6], DAS [7], WMCore [8], and DQMGUI [9]. CMSWEB consists of two clusters: the frontend and the backend ones. The frontend cluster, based on Apache, incorporates the x509 authentication to authenticate users and forwards their requests to the backend cluster. The Ingress controller in the backend cluster only accepts the requests from the frontend cluster and redirects them to the appropriate service.

The critical nature of CMSWEB demands an impervious secret management strategy that can effectively secure the secrets while allowing us to distribute them centrally across the organization. In the past, the CMSWEB Operator was in charge of maintaining and deploying these secrets. However, if the operator is unreachable, we could be locked out from redeploying our services. To address these issues, we carried out an extensive study to explore various secret management strategies such as Hashicorp Vault [10], GitHub credential manager [11], and SOPS/age [12]. In this paper, we investigate these strategies, present a feasibility analysis, and discuss why we selected the SOPs/Age strategy for securing secrets and configuration.

Following the Introduction, Section 2 provides the feasibility analysis of the potential solutions, while Section 3 elaborates on our currently incorporated methodology. Finally, Section 4 concludes the paper and presents some future directions.

2. Potential Solutions

In this section, we explore the three main secret management strategies typically employed by organizations to effectively secure and centralize Kubernetes secrets, namely: HashiCorp Vault, Git Credentials Manager, and SOPS with *age* [12]. We also present a thorough feasibility analysis of these strategies and discuss the solution adopted by CMS.

2.1. HashiCorp Vault

Hashicorp Vault is a powerful identity-based credential management service, providing several features like password rotation, auditing, and encryption as a service capability. Vault primarily works with tokens that are associated with the client's policy. Vault uses these policies to manage and control the behavior of the clients and provide Role Based Access Control (RBAC) as depicted in Figure. 1.

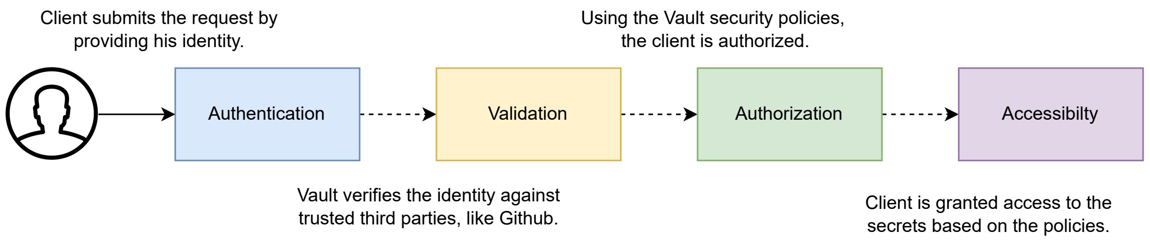

Client submits the request by providing his identity.
Using the Vault security policies, the client is authorized.
Authentication
Validation
Authorization
Accessibilty
Vault verifies the identity against trusted third parties, like Github.
Client is granted access to the secrets based on the policies.

Figure 1. Vault provides accessibility to the client using a four part mechanism.

All these extended features require complex configuration, which entails that new dependencies for our services will be introduced, requiring constant management and control.

2.2. Git Credentials Manager(GCM)

GitHub has developed a solution to authenticate to remote git repositories securely. Git credentials manager is a tool that allows users to securely store their credentials (such as usernames and passwords) on their local machine, so the users don't have to enter them every time they interact with a remote Git repository. Several versions of Git credential managers are available for different platforms [13]. Here is an example of how we could use Git Credentials Manager Core:

- Download and install the GCM Core package.
- Clone a git repository using `git clone repository-link.git`

- Use the `git config --global credential.helper` and specify the following parameter: `"/usr/local/share/gcm-core/git-credential-manager-core"` to configure Git to use GCM.
- The first time you interact with the repository, you will be asked to enter your credentials. The previous command will cache these credentials.
- From this point, GitHub will use the stored credentials to authenticate you to the repository.

Where Git credentials manager is a secure way to encrypt secrets and there is cross-platform support, its scope is limited to one machine, there is no centralized mechanism to store the credentials, and it does not provide any auditing mechanism. Nonetheless, it is very simplistic and can integrate with many tools like Azure DevOps and Visual Studio.

2.3. SOPS with age

SOPS (Secrets OPerationS) is a *Go*-based application, developed by Mozilla, for managing secrets. It is specifically used for the encryption and decryption of YAML, JSON, ENV, INI, and BINARY files. SOPS leverages the use of public-key cryptography to encrypt secrets: the public key is used to encrypt the data, and the private key is used to decrypt it. Usually, the private key is stored securely on the server, and the encrypted data is stored over version-controlled systems like Git.

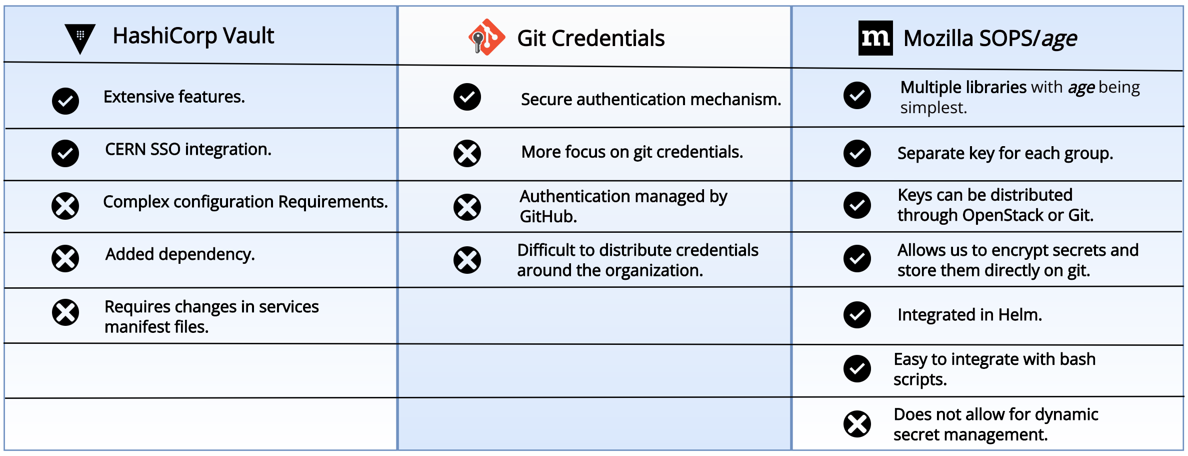
However, SOPS in itself cannot do the encryption: it relies on the use of multiple encryption libraries like AWS KMS, GCP KMS, Azure Key Vault, *age*, and PGP. PGP is deprecated, so the most straightforward of these tools is the *age* tool [14]. *Age* is a simple, modern tool that provides automated encryption and decryption of secrets using GPG encryption. *Age* does not care about the file formatting, therefore, to encrypt only the values or specific parts of the file, we use SOPS.

Using SOPS with *age* enables us to encrypt the secrets, and push them directly to GitHub. On top of this, each group can use its decryption key, which can be stored and distributed using the OpenStack secrets manager (Barbican) [15]. SOPS allows us to use the deployment scripts, which can access the key and decrypt the config files automatically before deploying the applications. Helm also comes with out-of-the-box SOPS integration: the *helm install* command can decrypt the secrets, given the correct key. The run manager can be given access to the various OpenStack projects, thereby ensuring observation and management of all the secrets.

2.4. Feasibility Analysis

In this section, we compare the advantages and disadvantages for each of the discussed solutions.

HashiCorp Vault offers several advantages. For instance, it is open-source, self-hosted, and easily integratable with CERN SSO. Moreover, it provides dynamic secret management with high availability. Vault can be configured with several interfaces, such as API, CLI, and UI. Despite these advantages, Vault requires complex configurations, introducing dependencies in our services. Hence, maintaining the secrets becomes an inconvenient task. On the other hand, Git Credentials Manager provides a simple mechanism to authenticate the users, but it is more focused on Git credentials rather than secrets in general. Although Git Credentials Manager can store secrets, it is not a secure or recommended method. Using GCM to store secrets can lead to a single point of failure. Because if someone gains access to the local machine, they may be able to access the Git credentials if they are stored in the Git credential manager. Since the credentials management is entirely managed by GitHub, it becomes laborious to centralize the management of secrets across the whole organization. These limitations can limit the effectiveness of GCM, and make it less suitable for some use cases where more secure methods of managing sensitive information are required.



HashiCorp Vault	Git Credentials	Mozilla SOPS/*age*
✓ Extensive features.	✓ Secure authentication mechanism.	✓ Multiple libraries with *age* being simplest.
✓ CERN SSO integration.	✗ More focus on git credentials.	✓ Separate key for each group.
✗ Complex configuration Requirements.	✗ Authentication managed by GitHub.	✓ Keys can be distributed through OpenStack or Git.
✗ Added dependency.	✗ Difficult to distribute credentials around the organization.	✓ Allows us to encrypt secrets and store them directly on git.
✗ Requires changes in services manifest files.		✓ Integrated in Helm.
		✓ Easy to integrate with bash scripts.
		✗ Does not allow for dynamic secret management.

Figure 2. A summary of the advantages and disadvantages of adopting various solutions.

SOPS with *age* is a simple, lightweight, and secure secret management strategy, which also allows for version control. It enables us to have a separate key for each group, whereby these keys can be easily distributed through OpenStack. Because of its compatibility with GitHub, the encrypted secrets can be pushed and pulled from the Git repository at any time. It also provides integration with Helm and can be easily integrated with the bash scripts. Nonetheless, it does not allow for dynamic secret management.

An outline of this feasibility analysis is provided in Figure 2.

2.5. Our Solution

With all the available solutions, it becomes quite difficult to pinpoint and choose one of them. Therefore, it is integral to identify the needs of one's system to know what is the most suitable option. There are some crucial requirements that we need to address, which are:

- To securely store secrets in a central location and to easily manage access to this location.
- To ensure that the run manager has access to all the CMSWEB secrets to enable seamless distribution to the experts in case of any emergency.
- To have an easy way of handing over the secrets when people join/leave groups.
- To ensure that only the most minimal dependencies are introduced in the system.
- To have a solution that works across all CMS platforms and to ensure that it is accessible from anywhere (e.g. remote nodes, and local machines).

It is also worth mentioning that the security robustness and the system's complexity go hand-in-hand: the more robust the system is, the more complex it becomes to configure and maintain. In CMSWEB, we want to achieve a trade-off between security complexity and robustness. SOPS with *age* highly complies with our requirements, which is why we propose using it for CMSWEB's secret management.

3. Proposed Technique

In this section, we provide a brief description of the proposed technique. We developed a set of bash scripts that can generate keys and mount them in the k8s clusters' namespaces. The list of bash scripts with their main functions is given below.

As illustrated in Figure 3, we have *public* keys deployed as secrets in our Kubernetes clusters. Using those keys, *User 1* encrypts the secrets using SOPS. These encrypted secrets are pushed to GitHub, or any other trusted third party. Thereafter, when other users need to use those encrypted secrets, they can pull them from the GitHub repository, garner the *private* key from the Kubernetes cluster and successfully decrypt them. The whole procedure does not require a cumbersome setup and is fairly easy to follow. Note that the *dev* teams in CMSWEB have an assigned namespace, and they cannot see, read or access the other namespaces. This implies that only authorized users can use the key secrets in each namespace.

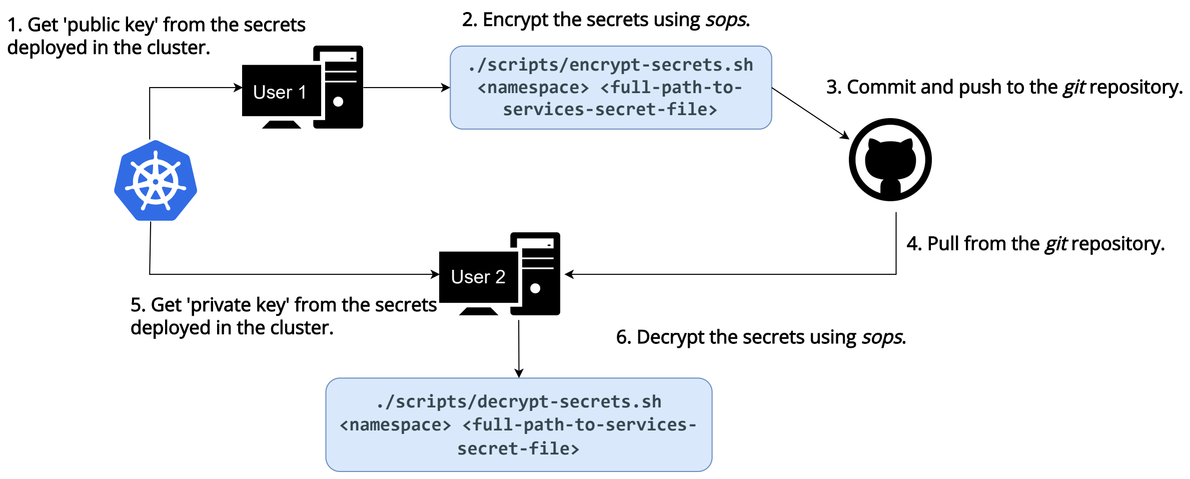

1. Get 'public key' from the secrets deployed in the cluster.
User 1
2. Encrypt the secrets using *sops*.
./scripts/encrypt-secrets.sh <namespace> <full-path-to-services-secret-file>
3. Commit and push to the *git* repository.
4. Pull from the *git* repository.
User 2
5. Get 'private key' from the secrets deployed in the cluster.
6. Decrypt the secrets using *sops*.
./scripts/decrypt-secrets.sh <namespace> <full-path-to-services-secret-file>

Figure 3. The *age* keys that are installed in the Kubernetes cluster enable the users to easily encrypt and decrypt the secrets. The encrypted secrets can be easily pushed and fetched from the private GitHub repository that has a Role Based Access Control.

Since SOPS is fairly easy to integrate, we have prepared two bash scripts to encrypt and decrypt the secrets automatically. We have one encryption key per namespace. The CMSWEB operator is in charge of deploying these keys in the particular namespaces. Once these keys are mounted in each namespace, access to these namespaces is granted based on role bindings (A role binding defines a role that grants access permission to a user or a group of users). This ensures that only specific people can access the secrets, for instance, only the CRAB developers can access/edit secrets in the CRAB namespace. In case of any infrastructure failure, these keys can also be obtained from OpenStack Barbican [16] or git. Currently, we proposed that only a selected subset of the managers can have access to these keys. Once the keys exist in the required namespace, the users with read-write permissions to the namespace can use our *deploy-secrets.sh* script to install encrypted secrets. This procedure is illustrated in Figure 4.

The helper scripts to carry out the above-mentioned operations are as follows:

- **encrypt-secrets.sh** [17]: to encrypt services secrets file using sops-age keys mounted as secrets in Kubernetes clusters.
- **decrypt-secrets.sh** [18]: to decrypt files using sops-age keys mounted as secrets in Kubernetes clusters.
- **deploy-keys-secrets.sh** [19]: to deploy sops-age keys secrets in the given namespace.
- **deploy-secrets.sh** [20]: to deploy a given service with a given tag to the k8s infrastructure.

There are many available use cases of SOPS already employed at CERN. One of the most notable ones is that of CERN IT – they have a SOPS plugin integrated with Barbican. We could adopt this solution, however, this plugin has not been merged in the official SOPS code and requires

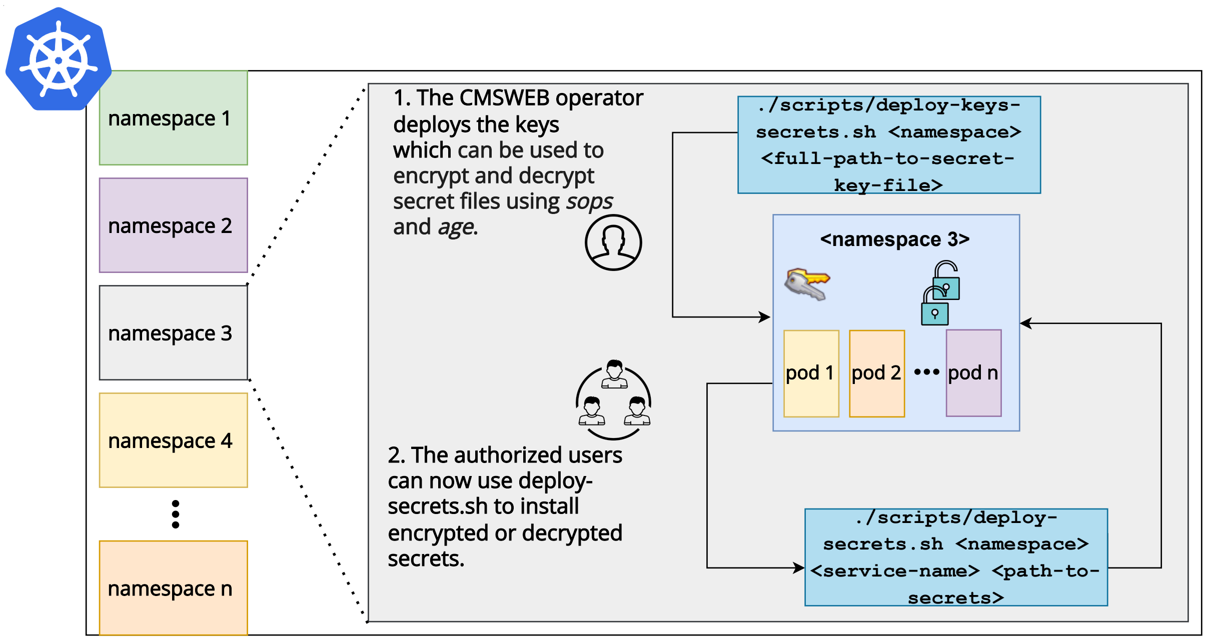

namespace 1
namespace 2
namespace 3
namespace 4
namespace n
1. The CMSWEB operator deploys the keys which can be used to encrypt and decrypt secret files using *sops* and *age*.
./scripts/deploy-keys-secrets.sh <namespace> <full-path-to-secret-key-file>
<namespace 3>
pod 1
pod 2
pod n
2. The authorized users can now use deploy-secrets.sh to install encrypted or decrypted secrets.
./scripts/deploy-secrets.sh <namespace> <service-name> <path-to-secrets>

Figure 4. After the CMSWEB Operator deploys the key secrets in a particular namespace in the k8s cluster, the authorized users/team can seamlessly configure the secrets in their respective namespace according to their needs.

setting up a custom fork. Since this is a significant overhead, we wanted an alternative solution. The RUCIO team in the ATLAS experiment is also using SOPS to encrypt their secrets. In their infrastructure, the decryption key is stored in *teigi* (a puppet infrastructure secret manager) [16]. The secrets are encrypted and pushed to GitHub. Along with this, they are also using Helm integration to decrypt the secrets. To garner more information about our secret management practices, the users can also visit our official documentation referenced here [21].

4. Conclusions and Future Work

This paper highlights the need for a robust secret management strategy for the Kubernetes infrastructure while providing an in-depth feasibility analysis of the state-of-the-art techniques that are commonly used. We described how and why CMSWEB selected SOPS with *age* as a secret management strategy. We have provided detailed documentation for the setup and incorporation of our solution and employed the use of a deployment script with SOPS to decrypt and deploy the encrypted secrets automatically. This proposed method is flexible to be tuned per the different use cases at CERN. We will devote our future efforts to integrating SOPS with Helm and to further automating our strategy.

References

[1] 2022 The state of secrets sprawl 2022 GitGuardian. URL `https://www.gitguardian.com/state-of-secrets-sprawl-report-2022(Accessed:January27,2023).`

[2] Burns B *et al.* 2016 *Borg, Omega, and Kubernetes.* URL `https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/44843.pdf(Accessed:January27,2023).`

[3] Petrucciani G 2013 *The Search for the Higgs Boson at CMS* **58** URL `https://doi.org/10.1007/978-88-7642-482-3_2`

[4] Anderson J C 2010 *The definitive guide: TIME TO RELAX, Google Books.* (O'Reilly Media, Inc.) URL `https://books.google.com/books/about/CouchDB_The_Definitive_Guide.html?id=G4N-DPk9R5sC(Accessed:January27,2023)`

[5] Mascheroni M *et al.* 2015 *Journal of Physics* URL `https://www.osti.gov/pages/biblio/1250780(Accessed:January27,2023)`

[6] Afaq A, Dolgert A, Gao Y, Jones C, Kosyakov S, Kuznetsov V, Lueking L, Riley D and Sekhri V 2007 The CMS Dataset Bookkeeping Service *Distributed Data Analysis and Information Management* (United States: IOP Publishing Ltd) URL `https://www.osti.gov/biblio/922732(Accessed:January27,2023)`

[7] Cinquilli M *et al.* 1975 *Crab3: Establishing a new generation of services for Distributed Analysis at CMS* vol 032026 (United States: Moscow State University Press) p 032026

[8] DMWM/WMCore: Core Workflow Management Components for CMS., GitHub. URL `https://github.com/dmwm/WMCore(Accessed:January27,2023)`

[9] Basic DQM GUI development, Sandbox, TWiki. URL `https://twiki.cern.ch/twiki/bin/view/Sandbox/BasicDQMGUIdevelopment(Accessed:January27,2023)`

[10] Vault by HashiCorp. URL `https://www.vaultproject.io/(Accessed:January30,2023)`

[11] Gitcredentials documentation. URL `https://git-scm.com/docs/gitcredentials(Accessed:January27,2023)`

[12] Mozilla/SOPS: Simple and flexible tool for managing secrets. URL `https://github.com/mozilla/sops(Accessed:January27,2023)`

[13] Git Credential Manager URL `https://github.com/GitCredentialManager`

[14] A simple, modern and secure encryption tool (and go library) with small explicit keys, no config options, and unix-style composability., GitHub. URL `https://github.com/FiloSottile/age(Accessed:January27,2023)`

[15] Barbican OpenStack. URL `https://wiki.openstack.org/wiki/Barbican(Accessed:January27,2023)`

[16] Secrets for puppet, TeigiVault. URL `https://twiki.cern.ch/twiki/bin/view/Main/TeigiVault(Accessed:January27,2023)`

[17] 2022 CMSKUBERNETES/encrypt-secrets.sh at master · DMWM/CMSKUBERNETES, GitHub. URL `https://github.com/dmwm/CMSKubernetes/blob/master/kubernetes/cmsweb/scripts/encrypt-secrets.sh(Accessed:January30,2023)`

[18] 2022 CMSKUBERNETES/decrypt-secrets.sh at master · DMWM/CMSKUBERNETES, github. URL `https://github.com/dmwm/CMSKubernetes/blob/master/kubernetes/cmsweb/scripts/decrypt-secrets.sh(Accessed:January30,2023)`

[19] 2022 CMSKUBERNETES/deploy-keys-secrets.sh at master · DMWM/CMSKUBERNETES, GitHub. URL `https://github.com/dmwm/CMSKubernetes/blob/master/kubernetes/cmsweb/scripts/deploy-keys-secrets.sh(Accessed:January30,2023)`

[20] 2022 CMSKUBERNETES/deploy-secrets.sh at master · DMWM/CMSKUBERNETES, GitHub. URL `https://github.com/dmwm/CMSKubernetes/blob/master/kubernetes/cmsweb/scripts/deploy-secrets.sh(Accessed:January30,2023)`

[21] 2022 Secret Management/SOPS and Age Tutorial URL `https://cms-http-group.docs.cern.ch/k8s_cluster/sops_age/`