OpenTofu State and Plan Encryption with AWS KMS

Date:	2024-12-22
modified:	2024-12-22
tags:	OpenTofu, Amazon, AWS, KMS, S3, DynamoDB
description:	Use AWS KMS to encrypt OpenTofu state and plan
category:	Cloud
slug:	opentofu-state-and-plan-encryption-with-aws-kms
Author:	Dominik Wombacher
lang:	en
transid:	opentofu-state-and-plan-encryption-with-aws-kms
Status:	published

Its been a while since OpenTofu 1.7.0 (Archive: [1], [2]) introduced an exciting new feature, the State and Plan Encryption (Archive: [1], [2]) for files at rest, for local storage and remote backends. I always found it challenging to keep `.tfstate` files secure. Now I can use Amazon Web Servers Key Management Service (AWS KMS) with a customer managed KMS key to encrypt the state before it's uploaded to Amazon Simple Storage Service (Amazon S3).

I decided to use CloudFormation to create a KMS key in `eu-central-1`, Europe (Frankfurt) and a replica in `eu-west-1`, Europe (Ireland) for backup purposes. Other AWS resources I've created: IAM User, IAM Roles, IAM Policies, S3 Bucket, DynamoDB Table and AWS Lambda function.. The Lambda function is based on CloudFormation Custom Resource AWS SSM Parameter Store SecureString and is necessary because of AWS CloudFormation and CDK doesn't support AWS SSM Parameter Store SecureString?!. Access and Secret Key for the IAM User are then auto-generated by CloudFormation and stored in AWS SSM Parameter Store.

When the AWS resource are ready, an example config to use state encryption and a S3 based remote backed looks like this:

```
# SPDX-FileCopyrightText: 2024 Dominik Wombacher <dominik@wombacher.cc>
#
# SPDX-License-Identifier: MIT

terraform {
  required_version = ">= 1.8"
  encryption {
    key_provider "aws_kms" "wombelix-sideprojects" {
      kms_key_id = "arn:${var.aws_partition}:kms:${var.aws_region}:${var.aws_account_id}:key/${var.aws_kms_name}"
      region     = var.aws_region
      key_spec   = "AES_256"
      assume_role = {
        role_arn = "arn:${var.aws_partition}:iam::${var.aws_account_id}:role/OpenTofuStateEncryptionRole"
      }
    }
    method "aes_gcm" "wombelix-sideprojects" {
      keys = key_provider.aws_kms.wombelix-sideprojects
    }
    state {
      method = method.aes_gcm.wombelix-sideprojects
    }
  }
  backend "s3" {
    bucket                  = var.aws_s3_bucket
    key                     = "opentofu-states/${var.project}/terraform.tfstate"
    region                  = var.aws_region
    skip_metadata_api_check = true
    encrypt                 = true
    kms_key_id              = "arn:${var.aws_partition}:kms:${var.aws_region}:${var.aws_account_id}:key/${var.aws_kms_name}"
    dynamodb_table          = "arn:${var.aws_partition}:dynamodb:${var.aws_region}:${var.aws_account_id}:table/iac-opentofu-remote-backend"
    assume_role = {
      role_arn = "arn:${var.aws_partition}:iam::${var.aws_account_id}:role/OpenTofuRemoteBackendRole"
    }
  }
}
```

For the above example config to interact with AWS, the following Environment variables have to be set:

```
AWS_REGION
AWS_ACCESS_KEY_ID
AWS_SECRET_ACCESS_KEY
TF_VAR_aws_region
TF_VAR_aws_account_id
TF_VAR_aws_kms_name
TF_VAR_aws_s3_bucket
TF_VAR_project
```

After bootstrapping with CloudFormation, all subsequent IaC can be implemented with OpenTofu. Each project gets its own unique S3 key in the form of `opentofu-states/<PROJECT>/terraform.tfstate`. That's all the customization it needs, which makes the solution basically maintenance free.

I like that the content of the state file is always encrypted. I also encrypt the S3 Bucket with the the customer managed key. When I use OpenTofu on my local system or in sr.ht builds, I leverage long-term Access and Secret key credentials. But the IAM User has no permissions directly assigned and can only assume a specific Role that allows access to KMS, S3 and DynamoDB. `s3:DeleteObject` is explicitly set to `Deny` and versioning enabled on the S3 Bucket. The potential attack surface is therefore very limited. In future I plan to avoid any usage of long-term credentials.

But for now the setup is already pretty decent and secure.