

Механизм секретов TOTP

Механизм секретов TOTP генерирует учетные данные, основанные на времени, в соответствии со стандартом TOTP. Механизм секретов может использоваться для генерации нового ключа и проверки паролей, сгенерированных с помощью этого ключа.

Механизм секретов TOTP может выступать как в роли генератора (как Google Authenticator), так и в роли провайдера (как служба входа в систему Google.com).

1. TOTP в роли провайдера

Механизм секретов TOTP может также работать в качестве провайдера TOTP. В этом режиме он может использоваться для генерации новых ключей и проверки паролей, сгенерированных с помощью этих ключей.

1.1. Настройка

Большинство механизмов секретов должны быть предварительно сконфигурированы, прежде чем они смогут выполнять свои функции. Эти шаги обычно выполняются оператором или инструментом управления конфигурацией.

1. Включите механизм секретов TOTP:

```
$ starvault secrets enable totp
Success! Enabled the totp secrets engine at: totp/
```

BASH | 

По умолчанию механизм секретов будет монтироваться по имени движка. Чтобы подключить механизм секретов по другому пути, используйте аргумент `-path`.

2. Создайте именованный ключ, используя опцию `generate`. Это указывает StarVault на роль провайдера:

```
$ starvault write totp/keys/my-user \
  generate=true \
  issuer=Vault \
  account_name=user@test.com
```

BASH | 

Вывод:

Key

Value

barcode

url

iVBORw0KGgoAAAANSUgAAAMgAAADIEAAAAADYoy0BA...

otpauth://totp/Vault:user@test.com?
algorithm=SHA1&digits=6&issuer=Vault&period=30&secret=V7MB-
SK324I7KF6KVW34NDFH2GYHIF6JY

Ответ включает в себя штрих-код в кодировке base64 и URL OTP. Оба варианта эквивалентны. Передайте их пользователю, который проходит аутентификацию с помощью TOTP. ---

2. TOTP в роли генератора

Механизм секретов TOTP может работать как генератор TOTP-кодов. В этом режиме он может заменить традиционные генераторы TOTP, такие как Google Authenticator. Это обеспечивает дополнительный уровень безопасности, поскольку возможность генерировать коды защищена политиками, а весь процесс подвергается аудиту.

2.1. Настройка

Большинство механизмов секретов должны быть предварительно сконфигурированы, прежде чем они смогут выполнять свои функции. Эти шаги обычно выполняются оператором или инструментом управления конфигурацией.

1. Включите механизмы секретов TOTP

```
$ starvault secrets enable totp
Success! Enabled the totp secrets engine at: totp/
```

BASH | 

По умолчанию механизм секретов будет монтироваться по имени движка. Чтобы подключить механизм секретов по другому пути, используйте аргумент `-path`.

2. Настройте именованный ключ. Имя этого ключа будет являться идентификатором пользователя.

```
$ starvault write totp/keys/my-key \
  url="otpauth://totp/Vault:test@test.com?
  secret=Y64VEVMBTSXCYIWRSHRNDZW62MPGVU2G&issuer=Vault"
Success! Data written to: totp/keys/my-key
```

BASH | 

`url` соответствует секретному ключу или значению штрих-кода, предоставленного сторонним сервисом.

2.2. Использование

После того как механизм секретов настроен и у пользователя/машины есть токен StarVault с соответствующими правами, он может генерировать учетные данные.

Сгенерируйте новый OTP на основе времени, считав из конечной точки `/code` имя ключа:

```
$ starvault read totp/code/my-key
Key      Value
---      -
code     260610
```

BASH | 

С помощью ACL можно ограничить использование механизма секретов TOTP таким образом, чтобы доверенные операторы могли управлять определениями ключей, а пользователи и приложения были ограничены в полномочиях, которые им разрешено читать.

2.3. Использование

В качестве пользователя проверьте код TOTP, сгенерированный сторонним приложением:

```
$ starvault write totp/code/my-user code=886531
Key      Value
---      -
valid    true
```

BASH | 

3. API

У механизма секретов TOTP есть полноценный HTTP API. Более подробную информацию можно найти в разделе API движка секретов TOTP.

Механизм управления секретами Transit

Механизм управления секретами Transit выполняет криптографические функции для данных в процессе передачи. StarVault не хранит данные, отправленные в механизм секретов. Это также можно рассматривать как "криптографию как услугу" или "шифрование как услугу". Механизм управления секретами Transit также может подписывать и проверять данные, генерировать хэши и HMAC данных, а также выступать в качестве источника случайных байтов.

Основная задача `transit` - шифровать данные из приложений, сохраняя зашифрованные данные в каком-то основном хранилище данных. Это снимает обязанность правильного шифрования/дешифрования с разработчиков приложений и перекладывает эту обязанность на операторов StarVault.

Поддерживается деривация ключей, что позволяет использовать один и тот же ключ для разных целей путем создания нового ключа на основе заданного пользователем значения контекста. В этом режиме опционально поддерживается конвергентное шифрование, которое позволяет из одних и тех же входных значений получать один и тот же шифртекст.

Генерация Datakey позволяет процессам запрашивать возврат высокоэнтропийного ключа заданной длины, зашифрованного с помощью именованного ключа. Обычно ключ также возвращается в открытом виде для немедленного использования, но это может быть отключено, чтобы удовлетворить требования аудита.

1. Управление набором менеджмента

Механизм Transit поддерживает версию ключей. Версии ключей, более ранние, чем заданная `min_decryption_version` ключа, архивируются, а остальные версии ключей попадают в рабочий набор. Это соображение производительности для быстрой загрузки ключей, а также соображение безопасности: запрещая расшифровку старых версий ключей, найденный шифртекст, соответствующий устаревшим (но чувствительным) данным, не может быть расшифрован большинством пользователей, но в экстренных случаях `min_decryption_version` может быть сдвинут назад, чтобы обеспечить легитимную расшифровку.

В настоящее время этот архив хранится в одной записи хранилища. В некоторых бэкендах хранилищ, в частности использующих Raft или Paxos для HA-возможностей, частая ротация может привести к тому, что размер записи в хранилище для архива будет больше, чем может выдержать бэкенд хранилища. При необходимости частой ротации использование именованных ключей, соответствующих временным границам (например, пятиминутные

периоды, увеличенные до ближайшего кратного пяти), может стать хорошей альтернативой. Это позволяет одновременно использовать несколько ключей и детерминированно решать, какой ключ использовать в каждый момент времени.

2. Руководство по ротации NIST

Рекомендуется периодическая ротация ключей шифрования, даже в отсутствие компрометации. Для ключей AES-GCM ротация должна происходить до того, как версия ключа выполнит примерно 2^{32} шифрования, в соответствии с рекомендациями публикации NIST 800-38D. Операторам рекомендуется оценить скорость шифрования ключа и использовать ее для определения частоты ротации, которая не позволит достичь пределов, указанных в руководстве. Например, если определить, что расчетная скорость составляет 40 миллионов операций в день, то достаточно ротировать ключ каждые три месяца.

3. Типы ключей

На данный момент механизм секретов поддерживает следующие типы ключей (все типы ключей также генерируют отдельные ключи HMAC):

- `aes128-gcm96` : AES-GCM со 128-битным ключом AES и 96-битным одноразовым кодом; поддерживает шифрование, дешифрование, выведение ключа и конвергентное шифрование
- `aes256-gcm96` : AES-GCM с 256-битным ключом AES и 96-битным одноразовым кодом; поддерживает шифрование, дешифрование, извлечение ключа и конвергентное шифрование (по умолчанию)
- `chacha20-poly1305` : ChaCha20-Poly1305 с 256-битным ключом; поддерживает шифрование, дешифрование, деривацию ключа и конвергентное шифрование
- `ed25519` : Ed25519; поддерживает подписание, проверку подписи и извлечение ключа
- `ecdsa-p256` : ECDSA с использованием кривой P-256; поддерживает подписание и проверку подписи
- `ecdsa-p384` : ECDSA с использованием кривой P-384; поддерживает подписание и проверку подписи
- `ecdsa-p521` : ECDSA с использованием кривой P-521; поддерживает подписание и проверку подписи
- `rsa-2048` : 2048-битный ключ RSA; поддерживает шифрование, дешифрование, подписание и проверку подписи
- `rsa-3072` : 3072-битный ключ RSA; поддерживает шифрование, дешифрование, подписание и проверку подписи

- `rsa-4096` : 4096-битный ключ RSA; поддерживает шифрование, дешифрование, подписание и проверку подписи
- `hmac` : HMAC; поддержка генерации и проверки HMAC * `managed_key` : Управляемый ключ; поддерживает различные операции в зависимости от решения для управления ключами. Дополнительные сведения см. в разделе Управляемые ключи.



В режиме FIPS 140-2 следующие алгоритмы не сертифицированы и поэтому не должны использоваться: `chacha20-poly1305` и `ed25519`.



Все типы ключей поддерживают операции HMAC благодаря использованию второго случайно сгенерированного ключа, созданного в момент создания ключа или его ротации. Тип ключа HMAC поддерживает только HMAC и ведет себя идентично другим алгоритмам в отношении операций HMAC, но поддерживает импорт ключей. По умолчанию тип ключа HMAC использует 256-битный ключ.

В операциях RSA используется один из следующих методов:

- **OAEP** (шифрование, дешифрование), с хэш-функцией SHA-256 и MGF
- **PSS** (подпись, проверка), с настраиваемой хэш-функцией, также используемой для MGF
- **PKCS#1v1.5**: (подпись, проверка), с настраиваемой хэш-функцией

4. Конвергентное шифрование

Конвергентное шифрование — это режим, при котором один и тот же набор открытый текст+контекст всегда приводит к одному и тому же шифртексту. Это достигается путем получения ключа с помощью функции получения ключа, а также путем детерминированного получения одноразового кода. Поскольку эти свойства различны для любой комбинации открытого текста и шифротекста в пространстве ключей размером (2^{256}) , риск повторного использования одноразового кода близок к нулю.

Это имеет множество практических применений. Один из распространенных способов использования - позволить хранить значения в базе данных в зашифрованном виде, но с ограниченной поддержкой поиска/запросов, чтобы при запросе можно было вернуть строки с одинаковыми значениями для определенного поля.

Чтобы учесть все необходимые обновления алгоритма, исторически поддерживались различные версии конвергентного шифрования:

- В версии 1 клиент должен был предоставить свой собственный одноразовый код, что очень гибко, но при неправильном использовании может быть опасно. Ключи, использующие эту версию, не могут быть обновлены.

- В версии 2 для получения параметров использовался алгоритмический подход. Однако используемый алгоритм был восприимчив к атакам с подтверждением открытого текста в автономном режиме, что могло позволить злоумышленникам расшифровать ключ грубой силой, если размер открытого текста был небольшим. Ключи, использующие версию 2, можно обновить, просто выполнив операцию обновления до новой версии ключа; затем существующие значения могут быть перевернуты в соответствии с новой версией ключа и будут использовать алгоритм версии 3.
- В версии 3 используется другой алгоритм, разработанный для защиты от атак с подтверждением открытого текста в автономном режиме. Он похож на AES-SIV тем, что использует PRF для генерации одноразового кода из открытого текста.

5. Настройка

Большинство механизмов секретов должны быть предварительно сконфигурированы, прежде чем они смогут выполнять свои функции. Эти шаги обычно выполняются оператором или инструментом управления конфигурацией.

1. Включите механизм секретов Transit:

```
$ starvault secrets enable transit
Success! Enabled the transit secrets engine at: transit/
```

BASH | 

По умолчанию механизм секретов будет монтироваться по имени механизма. Чтобы подключить механизм секретов по другому пути, используйте аргумент `-path`.

2. Создайте именованный ключ шифрования:

```
$ starvault write -f transit/keys/my-key
Key                               Value
---                               -
allow_plaintext_backup            false
auto_rotate_period                0s
deletion_allowed                  false
derived                           false
exportable                        false
imported_key                      false
keys                              map[1:1747922955]
latest_version                    1
min_available_version              0
min_decryption_version            1
min_encryption_version            0
name                              my-key
supports_decryption               true
supports_derivation               true
supports_encryption               true
```

BASH | 

supports_signing	false
type	aes256-gcm96

Обычно у каждого приложения есть свой ключ шифрования.

6. Использование

После того как механизм секретов настроен, и у пользователя/машины есть токен StarVault с соответствующими правами, он может использовать этот механизм секретов.

1. Зашифруйте некоторые данные в открытом виде с помощью конечной точки /encrypt с помощью именованного ключа:



Все открытые данные должны быть закодированы в base64. Причина этого требования заключается в том, что StarVault не требует, чтобы открытый текст был текстом. Это может быть двоичный файл, например PDF или изображение. Самый простой механизм безопасной транспортировки таких данных в составе полезной нагрузки JSON - это base64-кодирование.

```
$ starvault write transit/encrypt/my-key plaintext=$(echo "my secret data" | base64)
```

BASH |

Вывод:

Key	Value
---	----
ciphertext	vault:v1:8SDd3WHDOjf7mq69CyCqYJBXAiQQAVZRkFM13ok481zoCmHnSeDX9vyf7w==

Возвращаемый шифротекст начинается с `vault:v1:`. Первый префикс (vault) указывает на то, что он был завернут StarVault. v1 указывает на то, что для шифрования открытого текста использовался ключ версии 1; таким образом, при смене ключей StarVault знает, какую версию использовать для расшифровки. Остальное - это конкатенация вектора инициализации (IV) и шифротекста в формате base64.

Обратите внимание, что StarVault не хранит никаких этих данных. За хранение зашифрованного шифротекста отвечает вызывающая сторона. Когда вызывающая сторона захочет получить открытый текст, она должна будет вернуть шифротекст обратно в StarVault, чтобы расшифровать значение.





StarVault HTTP API устанавливает максимальный размер запроса в 32 МБ для предотвращения атак типа "отказ в обслуживании". Этот параметр можно настроить для каждого блока listener в конфигурации сервера StarVault.

2. Расшифруйте часть данных с помощью конечной точки `/decrypt` с помощью именованного ключа:

```
$ starvault write transit/decrypt/my-key  
ciphertext=vault:v1:8SDd3WHD0jf7mq69CyCqYjBXAiQQAVZRkFM13ok481zoCmHnSeDX9vyf  
7w==
```

BASH |

Вывод:

Key	Value
---	----
plaintext	bXkgc2VjcmV0IGRhdGEK

Полученные данные закодированы в base64 (подробнее о причинах смотрите в примечании выше). Расшифруйте его, чтобы получить исходный открытый текст:

```
$ base64 -d <<< "bXkgc2VjcmV0IGRhdGEK"  
my secret data
```

BASH |

Также можно написать сценарий расшифровки, используя некоторые умные сценарии оболочки в одной команде:

```
$ starvault write -field=plaintext transit/decrypt/my-key ciphertext=... |  
base64 -d  
my secret data
```

BASH |

С помощью ACL можно ограничить использование механизма секретов Transit таким образом, чтобы доверенные операторы могли управлять именованными ключами, а приложения могли шифровать и расшифровывать только те именованные ключи, к которым им необходим доступ.

3. Ротируйте базовый ключ шифрования. Это сгенерирует новый ключ шифрования и добавит его в связку ключей для названного ключа:

```
$ starvault write -f transit/keys/my-key/rotate  
Key Value  
---  
allow_plaintext_backup false  
auto_rotate_period 0s
```

BASH |

```
deletion_allowed      false
derived               false
exportable            false
imported_key          false
keys                  map[1:1747922955 2:1747923688]
latest_version        2
min_available_version 0
min_decryption_version 1
min_encryption_version 0
name                  my-key
supports_decryption   true
supports_derivation   true
supports_encryption   true
supports_signing      false
type                  aes256-gcm96
```

В последующих шифрованиях будет использоваться этот новый ключ. Старые данные все еще могут быть расшифрованы благодаря использованию кольца ключей.

4. Обновление уже зашифрованных данных на новый ключ. StarVault расшифрует значение с помощью соответствующего ключа в связке ключей, а затем зашифрует полученный открытый текст с помощью новейшего ключа в связке ключей.

```
$ starvault write transit/unwrap/my-key
ciphertext=vault:v1:8SDd3WHD0jf7mq69CyCqYjBXAiQQAVZRkFM13ok481zoCmHnSeDX9vyf
7w==
```

Вывод:

Key	Value
---	----
ciphertext	vault:v2:0VHTTBb2EyyNYHsa3XiXsvXOQSLKulH+NqS4eRZdtc2TwQCxqj7PUipvqQ==

Этот процесс **не раскрывает** открытый текст данных. Таким образом, политика StarVault может предоставить почти недоверенному процессу возможность "перевернуть" зашифрованные данные, поскольку процесс не сможет получить доступ к открытым данным.

7. Импортируйте свой собственный ключ (BYOK)



Функция импорта ключей поддерживает случаи, когда необходимо принести существующий ключ из HSM или другой внешней системы. Более безопасно, если Transit генерирует и управляет ключом внутри StarVault.

Сначала нужно считать обертываемый transit:

```
$ starvault read transit/wrapping_key
```

BASH | 

Обертываемый ключ будет представлять собой 4096-битный открытый ключ RSA.

Затем обертываемый ключ используется для создания входного шифротекста для конечной точки `import`, как описано ниже. Ниже под целевым ключом подразумевается импортируемый ключ.

7.1. HSM

Если ключ импортируется из HSM, поддерживающего PKCS#11, возможны два сценария:

- Если HSM поддерживает механизм CKM_RSA_AES_KEY_WRAP, его можно использовать для обертывания целевого ключа с помощью обертываемого ключа.
- В противном случае для обертывания целевого ключа можно использовать два механизма. Сначала необходимо сгенерировать 256-битный ключ AES, а затем использовать его для обертывания целевого ключа с помощью механизма CKM_AES_KEY_WRAP_KWP. Затем ключ AES должен быть обернут под обертывающий ключ с помощью механизма CKM_RSA_PKCS_OAEP с использованием MGF1 и либо SHA-1, SHA-224, SHA-256, SHA-384, либо SHA-512.

Шифротекст создается путем добавления обернутого целевого ключа к обернутому ключу AES.

Байты шифротекста должны быть закодированы в base64.

7.2. Импорт вручную

Если целевой ключ не хранится в HSM или KMS, для создания шифротекста на входе конечной точки импорта можно использовать следующие шаги:

- Сгенерируйте эфемерный 256-битный ключ AES
- Оберните целевой ключ эфемерным ключом AES с помощью AES-KWP.



При обертывании симметричного ключа (такого как ключ AES или ChaCha20), обертывайте необработанные байты ключа. Например, при использовании 128-битного ключа AES это будет массив байтов длиной 16 символов, который будет непосредственно преобразован без использования base64 или других кодировок.

При преобразовании асимметричного ключа (такого как ключ RSA или ECDSA) следует преобразовать формат этого ключа, закодированный в PKCS8, в необработанный DER/двоичный формат. Не применяйте кодировку PEM к этому большому двоичному объекту перед шифрованием и не кодируйте его в base64.

- Оберните ключ AES под ключ обертки StarVault, используя RSAES-OAEP с MGF1 и SHA-1, SHA-224, SHA-256, SHA-384 или SHA-512
- Удалите эфемерный ключ AES
- Добавьте обернутый целевой ключ к обернутому ключу AES
- Закодируйте результат в Base64

8. API

У механизма секретов Transit есть полноценный HTTP API. Более подробную информацию можно найти в разделе API механизма секретов Transit.

Общие сведения о платформе StarVault

StarVault - это инструмент для управления конфиденциальной информацией, обеспечивающий централизованное хранение и эффективное управление доступом к таким секретам, как токены, пароли, сертификаты и ключи шифрования. Этот инструмент не только гарантирует безопасное хранение чувствительных данных, но и предоставляет возможность их строгого контроля, а также автоматизации связанных процессов, повышая тем самым уровень защиты данных в целом.

Секреты представляют собой конфиденциальные данные, такие как токены, API-ключи, пароли, ключи шифрования и сертификаты, доступ к которым должен быть строго ограничен. StarVault предлагает унифицированный интерфейс для управления этими секретами, гарантируя при этом тщательный контроль за доступом и ведение детализированного журнала аудита всех операций с ними.

1. Содержание документации

Переходите в нужный раздел по быстрым ссылкам:

- [Общие сведения о платформе StarVault](#)
 - [Устройство StarVault](#)
 - [Отказоустойчивость](#)
 - [Модель безопасности](#)
 - [Лимиты и ограничения](#)
- [Руководство по установке платформы](#)
 - [Варианты установки](#)
 - [Установка в ОС Linux](#)
 - [Установка в режиме высокой доступности \(HA\)](#)
 - [Установка в Kubernetes](#)
 - [Общие сведения об установке с помощью HELM](#)
 - [Установка с помощью HELM](#)
 - [Параметры Helm](#)
 - [Примеры конфигураций](#)
 - [Установка в среде выполнения контейнеров](#)
 - [Установка в тестовом режиме](#)

- Параметры конфигурации
 - Рекомендации по автоматизации
 - Блок конфигурации listener
 - Блок конфигурации seal
 - Опция service registration
 - Блок конфигурации storage
 - Filesystem
 - In-memory
 - PostgreSQL
 - Filesystem
 - Integrated Storage
 - Блок конфигурации telemetry
 - Блок конфигурации ui
 - Блок конфигурации блокировки пользователей
 - Логирование выполненных запросов
- Миграция секретов из Vault в StarVault
- Руководство администратора
 - Хранилище
 - Распечатывание хранилища
 - Ротация ключей
 - Встроенное хранилище
 - Механизмы управления секретами
 - Механизм секретов Cubbyhole
 - Базы данных
 - MySQL/MariaDB
 - Oracle
 - PostgreSQL
 - Механизм секретов Identity
 - Механизм секретов KV
 - Механизм секретов Kubernetes
 - Механизм секретов LDAP
 - Механизм секретов PKI

- Настройка и использование
- Настройка корневого центра сертификации
- Настройка промежуточного центра сертификации
- Рекомендации
- Решение проблем с ACME
- Rotation primitives
- RabbitMQ
- Механизм секретов SSH
- Механизм секретов TOTP
- Механизм управления секретами Transit
- Управление доступом
 - Аутентификация на основе токенов
 - Методы аутентификации
 - AppRole
 - JWT/OIDC
 - Провайдер OIDC
 - Kerberos
 - Kubernetes
 - LDAP
 - LDAP Meta
 - Login MFA
 - RADIUS
 - TLS сертификаты
 - Токены
 - Логин и пароль
 - Логин и пароль. Расширенный
 - Аренда, обновление и отзыв
 - Блокировка пользователей
 - Идентификация
 - OIDC провайдер
 - Политики доступа
- Аудит

- Устройство для файлового аудита
- Устройство аудита Syslog
- Устройство аудита Socket

- Телеметрия

- Включение сбора телеметрии
- Ключевые метрики для общей проверки работоспособности
- Описание метрик
 - Основные системные метрики
 - Метрики логирования
 - Метрики аутентификации
 - Метрики доступности
 - Метрики базы данных
 - Метрики политик
 - Метрики встроенного хранилища
 - Метрики секретов
 - Полный список метрик

- Рекомендации

- Рекомендации по настройке
- Рекомендации по обеспечению безопасности
- Контроль расходования ресурсов
- Настройка производительности

- Инструкции

- Миграция точек монтирования
- Работа с пользователями, группами
- Ограничение доступа при расследовании инцидентов
- Отправка логов аудита в OpenSearch
- Измерение производительности
- Использование шаблонов в политиках доступа
- Настройка политик паролей
- Диагностика проблем с запуском
- Настройка MFA с использованием TOTP

- Утилиты

- Библиотеки
- Программы
 - Агент и прокси
 - Автоматическая аутентификация
 - AppRole
 - Сертификат
 - JWT
 - Kerberos
 - Kubernetes
 - Создание файла токена
 - Приемники автоматической аутентификации (Sinks)
 - Агент
 - Кэширование
 - Постоянный кэш
 - Kubernetes
 - Генерация конфигураций
 - Режим супервизора
 - Шаблоны
 - Сервис Windows
 - Совместимость версий
 - Прокси
 - Прокси для API
 - Кэширование
 - Постоянное кэширование
 - Kubernetes
 - Совместимость версий
- История изменений
 - v1