UNSEALING THE VAULT

A review of the secret sharing implementation in Hashicorp's Vault

JAMES HALL

July 2016

1. What is Vault?

Vault is a tool for managing secrets securely. This extract from the Vault project page summarises it's functionality:

> Vault secures, stores, and tightly controls access to tokens, passwords, certificates, API keys, and other secrets in modern computing. Vault handles leasing, key revocation, key rolling, and auditing. Vault presents a unified API to access multiple backends: HSMs, AWS IAM, SQL databases, raw key/value, and more.
>
> **vaultproject.io**

Vault must be *unsealed* with a *master key* before it can perform these operations. Instead of giving the key to a single trusted party, it is split into pieces and distributed amongst multiple parties. A predefined minimum number of these parties must provide their share of the master key for it to be reconstructed and used to unseal the vault. *Shamir's secret sharing scheme* has been implemented to provide the splitting functionality.

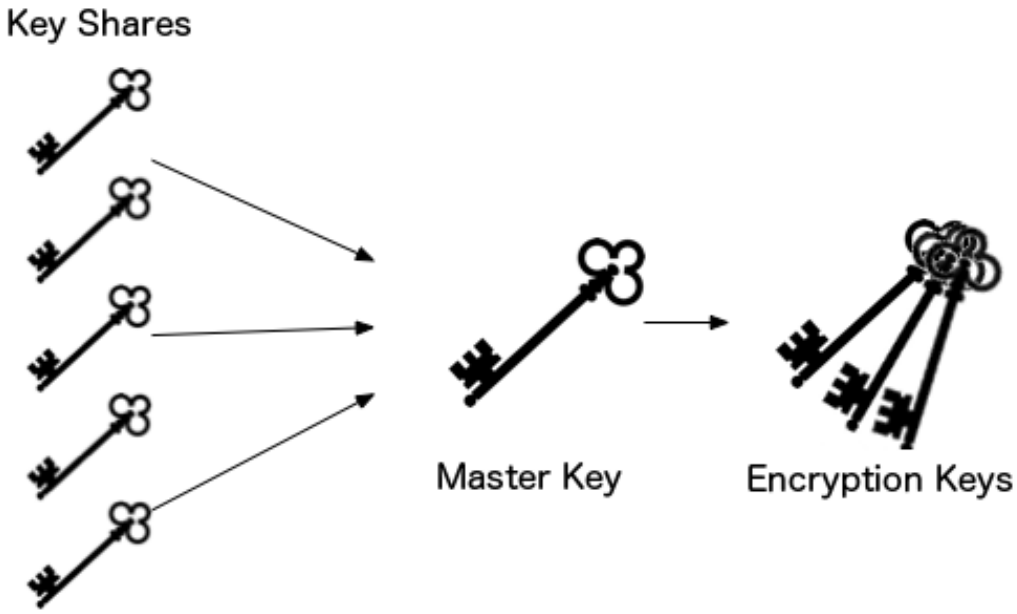
This document has been produced to discuss the mathematics behind the scheme in an accessible way. Also, to discuss whether the measures taken to ensure the security of data passed to Vault are sufficient to consider it for use in an enterprise setting.

2. How does Vault Protect Encryption Keys?

Secrets passed to the vault are encrypted with a key before they are passed to storage back-ends. Without further protection, this encryption key is a single point of failure as if this key is recovered, the data can be decrypted.

Vault protects the encryption key by creating a *master key* when it is initialised. When the vault is *sealed*, the master key is used to encrypt the encryption key. Of course, this is a chicken-and-egg situation as the master key is now a single point of failure.

To avoid this the master key is split into shares so that they can be distributed to different parties. After sealing, a minimum number of shares must be passed to the server before the vault can be *unsealed*.



3. How is the Master Key Split?

Vault makes use of *Shamir's Secret Sharing Scheme* to split the master key amongst multiple parties. In Shamir's Scheme, numbers representing the coordinates of points lying on a curve generated at run-time are distributed instead of the secret itself. The secret can only be recovered if a predefined number of the shares are retrieved first.

3.1 What does the curve look like?

The curve is defined to be a $polynomial$ which takes the form:

$$y = a_1 x^n + a_2 x^{n-1} + \ldots + a_{n-1} x + a_n$$

Here $a_1, \ldots, a_n$ are called *coefficients* and are constant in value. The value of n is called the *degree* of the polynomial and is simply the highest power that the variable x is raised to. As an example, $x^4 + 2x^2 + 3x$ has coefficients 1, 2 and 3 and is of degree 4.

In Shamir's scheme, we define a polynomial of degree $k - 1$ and set the value of a_n to be the secret, which we will call S from here on. Then, n points on the curve (other than at 0) are chosen as the values for the shares. If each of the parties have the same level of trust, then each is given the coordinates of one of these n points as their share of the secret.

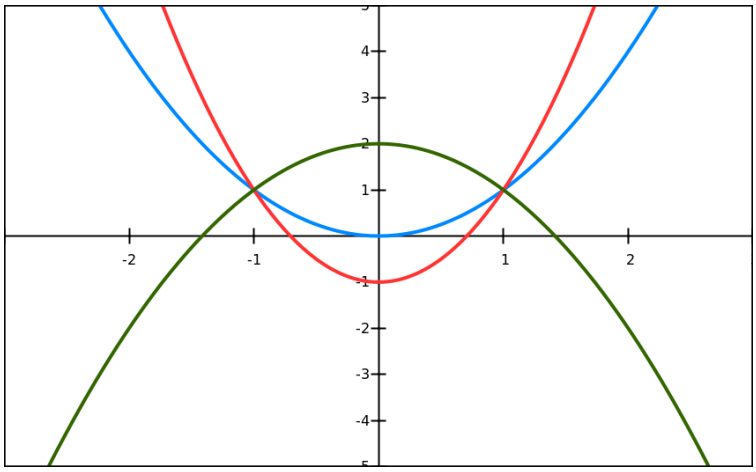
3.2 How is the secret recovered?

Suppose we don't know what form a particular polynomial takes but we do know a number of points that lie on it. If the polynomial is of degree $k - 1$, then we can *interpolate* between k known points to calculate the value at 0, i.e. the secret, S. This cannot be done with fewer than k points as that would not constrain the polynomial enough and there would be an infinite number of polynomials with degree $k - 1$ that could be defined.

For example, if we know that a polynomial p has degree 2 and we have only the points $(-1, 1)$ and $(1, 1)$, then p could be any of the following polynomials, since all of these equations are true when the x and y values from the coordinates of these two points are used:

$$y = x^2, \quad y = 2x^2 - 1, \quad y = -x^2 + 2$$

The graph below shows these three polynomials, all of which pass through the two points $(-1, 1)$ and $(1, 1)$. As can be seen clearly from the graph, no two polynomials of degree 2 which pass through these points have a third point of intersection. This shows only one additional point is needed to be able to establish which polynomial is the true value.



It is important to note here that we can use algebraic methods to gain more information about the possible values of a secret if we have two points. Shamir's scheme *leaks* information when used in our everyday field of numbers, called the *real* number field $\mathbb{R}$, which contains the positive and negative whole numbers, simple fractions and irrational numbers such as $\sqrt{2}$ and π. An example has been included as an appendix. To avoid leaking, the scheme must be implemented using what is called a *finite field*.

Shamir's scheme is *information-theoretic secure* when used with a finite field. That is, an attacker simply does not have enough information to compute the secret. They cannot use algebraic methods to narrow down the possible solutions; they can do no better than to guess. An adversary must have at least k shares to recover the secret without trying all possible values.

3.3 What is a finite field?

Finite fields have a limited number of elements and have different rules when performing arithmetic than on everyday real numbers. The size of the field is important as it must be a prime power (p^n where p is prime and $n \geqslant 1$) for certain arithmetic operations to be well defined.

3.4 How do we perform addition in finite fields?

Suppose an arithmetic operation produces the result x when working with the real numbers. In a finite field of prime size p, x should then be divided by p and the remainder should be taken as the result. This is equivalent to the *mod* operation in computing. For example, addition of 3 and 4 in the finite field with 5 elements would work as follows:

$$3 + 4 = 7 = 2 \pmod 5$$

as 5 divides into 7 once with remainder 2.

In digital cryptography, we often use a finite field with $2^8 = 256$ elements. When the size of the field is a power of two, we call it a *binary* field. As there are 256 elements, they can all be represented within a single byte. As the size of the field isn't prime, the values don't directly correspond to the integers 0 to 255. Instead, we must take a polynomial representation.

3.5 How do we use a byte to represent polynomials?

If a polynomial is of degree 7 or lower and has only binary values for $a_1, \ldots, a_n$, we can use each bit to represent the values of the coefficients. For example:

$$x^7 + x^5 + x^2 + x + 1$$

is equivalent to:

$$1 \times x^7 + 0 \times x^6 + 1 \times x^5 + 0 \times x^4 + 0 \times x^3 + 1 \times x^2 + 1 \times x + 1$$

which can be represented as:

$$10100111$$

3.6 How do we perform addition in a binary field?

Having polynomial elements with binary coefficients simplifies addition and subtraction. As the coefficients of the polynomials are binary, -1 becomes equal to 1 and 2 is equal to 0 as shown below.

$$-1 \pmod 2 = 1$$

$$2 \pmod 2 = 0$$

This means that addition and subtraction now become equivalent to the XOR operation.

Example

Suppose we have a binary field with 256 elements. Two elements 145 and 123 have values 10010001 and 01111011 in binary. If we use the representation above, we can deduce the polynomial representations, $x^7 + x^4 + 1$ and $x^6 + x^5 + x^4 + x^3 + x + 1$. The addition of these two polynomials is shown below.

$$(x^7 + x^4 + 1) + (x^6 + x^5 + x^4 + x^3 + x + 1)$$

$$= x^7 + x^6 + x^5 + (2 \times x^4) + x^3 + x + 2$$

$$= x^7 + x^6 + x^5 + x^3 + x$$

$x^7 + x^6 + x^5 + x^3 + x$ has binary representation 11101010 (or 234 in decimal). The equivalent XOR operation using the binary representations is shown below.

$$10010001$$

$$\oplus\ 01111011$$

$$= 11101010$$

3.7 How do we perform multiplication in a binary field?

Multiplying polynomials may produce polynomials with a degree higher than 7. For example, multiplying two degree 7 polynomials will yield a degree 14 polynomial. Clearly, this can no longer be represented in the field of 256 elements. As with addition in a finite field, we must take the modulus of the result. However, we must use what is called an *irreducible polynomial* instead of a prime number.

Polynomials can be multiplied and factored much like numbers. For example,

$$(x+1)(x+4) = x^2 + 5x + 4$$

In this case we can say that $x+1$ and $x+4$ are *factors* of x^2+5x+4. However x^2+1 does not have factors as it cannot be represented as the product of simpler polynomials. It is an *irreducible* polynomial of degree 2. In Shamir's scheme an irreducible polynomial of degree 8 is used when taking the modulus of large polynomials.

A quick way to implement multiplication for relatively small binary fields is to make use of the following logarithm law:

$$log_b(A) + log_b(B) = log_b(AB)$$

If we take the base, b, to be a *generator* for the field, we can use lookup tables to find the results.

3.8 How can generators be used to aid multiplication?

A generator is an element of a finite field that can be used to produce every other element of the field except zero when repeated exponentiation is applied. There are typically many generators in a given field but not all elements may be generators. As an example, in the field of 5 elements, 4 is not a generator as only the values 1 and 4 are produced. However, 2 is a generator as repeated exponentiation produces all the values in the field other than zero.

x	4^x (mod 5)	2^x (mod 5)
0	1	1
1	4	2
2	1	4
3	4	3

For finite fields that are not prime in size, a polynomial is used as a generator. In the 256 element binary field, $x+1$ is such a generator. Therefore, we can build a lookup table of all the values $(x+1)^i$ takes for $0 \leqslant i \leqslant 255$ so that we can calculate $log_{x+1}(A)$ and $log_{x+1}(B)$. Similarly, an inverse log table can be built up to calculate AB after finding $log_{x+1}(AB)$.

As an example, if we wished to calculate 123 multiplied by 19, we would first look up the values for $log_{x+1}(123)$ and $log_{x+1}(19)$ in the log table. The values are 229 and 14 respectively. We add these values to get 243 and look up this value in the inverse log table to get the result 124. If the result of the addition is greater than 255, we can simply subtract 255 as the table is cyclic.

3.9 Summary of the Scheme

Suppose we have a secret S which we would like to split into n pieces such that:

- At least k pieces must be present for the secret to be reconstructed
- Any k of the n pieces can be used to reconstruct the secret
- Someone with fewer than k pieces gains no further information about the secret than someone with no pieces

We can use Shamir's Secret Sharing Scheme in a finite field to satisfy these conditions with information theoretic security.

In Shamir's scheme, we define a polynomial of degree $k-1$ that passes through the point $(0, S)$. Then, n points on the curve (other than at 0) are chosen as the values for the shares. These are then distributed without leaking information about S as at least k of them are required to reconstruct the polynomial.

When we wish to reconstruct S, we can interpolate between k points to find the polynomial. The value at 0 is then S.

4. How does Vault's Implementation Stack up?

This section discusses the implementation in Vault and how it takes precautions to protect the encryption key.

4.1 Encryption Keys

There is little point in protecting the encryption key if the strength of encryption is sufficiently low as to warrant a direct attack on the data.

Vault encrypts data in memory before it is sent out for storing. AES-GCM-256 is used for encrypting the data which is an algorithm NIST recommends provided certain conditions are met regarding uniqueness of initialisation vectors and the randomness of certain values. Vault makes use of Go's cipher package for encryption which meets these requirements.

4.2 Rotating the Encryption Key

Whether there is value in rotating encryption keys is debated but sometimes organisational policy specifies that it must occur. Vault makes it possible to use key rotation if necessary.

Rotation of the encryption key is simplified in Vault as it is never revealed to users. When a new encryption key is created, it is added to the *keyring*. This means that a key rotation does not force a re-encryption of the entire dataset. When a secret is requested, it is decrypted with the key it was last encrypted with but then re-encrypted with the most recent encryption key. The entire keyring is encrypted with the master key when the vault is sealed.

4.3 Generation of the Master Key

When generating keys it is important to use a *cryptographically strong* number generator. While computers cannot be truly random, they can take input from sensors which produce seemingly random data. A cryptographically strong number generator makes it sufficiently hard for an adversary to determine the next number that it will generate.

At initialisation time a random 256 bit master key is generated using the rand package supplied by Go. The rand.Read method fills a given buffer with bytes generated from the commonly used /dev/urandom file. /dev/urandom is a cryptographically strong random number generator available on UNIX-like systems that gathers environmental noise from device drivers and other sources depending on the system it is running on.

4.4 Generation of the Shares

In Vault, the master key is split byte by byte. So, instead of creating one polynomial with a constant value equal to the value of the master key, a different polynomial is created for each byte in the master key. Each polynomial must have randomly generated coefficients to be cryptographically secure and the elements must lie within a finite field to be information-theoretic secure.

Vault uses a 256 element finite field generated using $x^7 + x^6 + x^5 + x^2 + 1$. The same method for generating random numbers is used as with the generation of the master key so it is also cryptographically secure. The shares have the following format:

$$y_1(x_i)y_2(x_i)y_3(x_i)\ldots y_{32}(x_i)x_i \quad 1 \leqslant i \leqslant k$$

We can see that the same x-coordinate is used for each of the 32 polynomials which means the x-coordinate only needs to be stored once, on the end of the share. The corresponding y-coordinates for each byte are stored in the same order as they appear in the master key to ease reconstruction.

4.5 Distributing Shares

Creating shares securely is only half the battle. They must then be distributed to the respective parties in a manner which keeps them safe. The master key is generated when Vault is initialised. The default number of shares is set to 5 and by default at least 3 of these must be brought together to reconstruct the master key. When the initialisation command is sent, the shares are outputted in plain text. However, it is also possible to pass in a set of PGP keys which can then be used to encrypt the shares before they are outputted.

It is highly recommended that this functionality is used in order to protect the shares from being displayed to users. The private keys related to the keys passed to the server should be exclusively owned by the respective share holders so that only they can decrypt the shares when required.

4.6 Unsealing the Vault

A minimum of k shares must be returned to the server before it can be unsealed. It should be noted that any share that has been passed over is held in memory until either the vault is unsealed or a reset occurs. It should also be noted that Vault requires the shares in raw form to perform unsealing. Hashicorp therefore recommends that TLS v1.2 is used to protect them while in transit. The client should ensure the certificate used by the server during the handshake is the one expected of the Vault instance.

During unsealing, numbers must be multiplied and their multiplicative inverses must be found. To speed up the calculation of these values, arrays containing the results have been precomputed and stored alongside the implementation. This helps to reduce the chance of a successful timing attack as the time to compute any value is reduced to the time of accessing a single element of the array. Note that this doesn't affect the encrypt/decrypt operations, only reconstruction of the master key.

If an authentication back-end has been registered, operations in Vault still require an authenticated user. The level of authentication should be appropriate for the sensitivity of the secrets being stored and should be considered on a case by case basis.

4.7 Rotating the Master Key

The master key can also be rotated independently of the encryption key after sending the threshold number of shares to the server. The key constructed from these shares is then compared with the current master key and if they match, a new master key is generated. The new key is used to encrypt the keyring and then new shares are generated and distributed. The old master key is discarded.

4.8 Adding and removing shares

Shamir's scheme allows the number of parties that hold a share to increase without affecting those already holding a share. As the shares are based on points that lie on a polynomial, a new point simply has to be chosen and sent to the new party. Unfortunately, Vault does not provide this functionality and instead requires a new set of shares to be produced which satisfies the new value of n (and optionally k).

Similarly, there is no way to dynamically reduce the number of shares in use. Suppose a party left the organisation or was no longer trusted, the master key would have to be redefined with a new value of n. The shares would again have to be redistributed to the remaining trusted parties.

5. Final Thoughts

Vault 0.6.0 provides a strong implementation of Shamir's secret sharing scheme to protect the key(s) used to encrypt sensitive data. Appropriate packages are used for the cryptographic operations and precautions are taken to protect against common attacks. This ranges from the correct use of finite field theory to providing support for encrypting the shares ready for secure transit after generation. Key rotation is also supported for both the encryption key and the master key.

While not all properties of Shamir's secret sharing scheme have been utilised, the necessary functionality has been provided to keep secrets safe.

Appendix: How can using an infinite field leak information?

Suppose an adversary finds the 2 points $D_0 = (1, 1494)$ and $D_1 = (2, 1942)$. They still don't have $k = 3$ points so in theory they shouldn't have won anymore info about the secret, S.

However, suppose they combine the info from the 2 points with the public info: $n = 6, k = 3$ and the polynomial $f(x) = a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}$ where $a_0 = S$.

If they fill in the polynomial with the values from each of the two points, they can find:

(i) $1494 = S + a_1 + a_2$

(ii) $1942 = S + 2a_1 + 4a_2$

Subtracting (ii) from (i) gives:

$448 = a_1 + 3a_2$

and rearranging gives:

(iii) $a_1 = 448 - 3a_2$

The attacker knows that the values of a_1 and a_2 are positive, so if they fill in all possible values for a_2 and find:

(iv) $a_2 \in [0, 1, \ldots, 148, 149]$

The formula in (iii) can be substituted into (i) to give:

$S = 1046 + 2a_2$

and so it can be deduced from (iv) that:

$S \in [1046, 1048, \ldots, 1344]$

Knowing fewer than the requisite number of shares has allowed the adversary to narrow down the possible values of the secret S.