Key-Value store Developer Tutorial

Learn how to use SecretPath on EVM to encrypt payloads.

Overview

SecretPath seamlessly handles encrypted payloads on the EVM, which means EVM developers can use SecretPath to encrypt and decrypt messages cross-chain with little-to-no Rust experience required.

This tutorial explains how to upload your own Key-value store contract on Secret Network, which you can use to encrypt values on the EVM, as well as how to encrypt payloads and transmit them cross-chain. After this tutorial, you will have the tools you need to use SecretPath to encrypt messages on anyEVM-compatible chain .

Getting Started

To get started, clone the SecretPath tutorials repository:

```
Copy gitclonehttps://github.com/SecretFoundation/Secretpath-tutorials
```

Configuring Environment Variables

cd into encrypted-payloads/evm-contract

```
Copy cdencrypted-payloads/evm-contract
```

Install the dependencies:

```
Copy npminstall
```

Create anenv file and add your:

- EVM wallet private key
- Infura API endpoint (Sepolia testnet)
-

Seehere for a properly configured example env file Get sepolia tokens from faucet:

- Sepolia faucet
-

This tutorial is for Sepolia testnet, but there are 10+ chains currently configured that are also compatible by simply swapping out theSecretPath gateway address . Now that your developer environment is properly configured, you're ready to encrypt your first payload with SecretPath!

Upload the Key value store contract on Secret Network

cd into encrypted-payloads/secret-contract

```
Copy cdencrypted-payloads/secret-contract
```

Compile the Secret Network key value store contract:

```
Copy RUSTFLAGS='-C link-arg=-s'cargobuild--release--targetwasm32-unknown-unknown
```

cd into secret-contract/node

```
Copy cdsecret-contract/node
```

Install the dependencies:

```
Copy npmi
```

Open the upload.js file and review the instantiate message atline 70 :

```
Copy letinit={ gateway_address:gatewayAddress, gateway_hash:gatewayHash, gateway_key:gatewayPublicKeyBytes, };
```

- gatewayAddress is the SecretPath gateway contract address for testnet
- gatewayHash is the SecretPath gateway contract hash for testnet
- gatewayKey is public key used for SecretPath encryption on Secret testnet
-

These three parameters remain constant and must be passed for every Secret Network contract that implements SecretPath. They can be foundhere for testnet. To upload and instantiate the contract, runnode upload :

```
Copy nodeupload
```

Upon successful upload, acontractHash andaddress will be returned:

```
Copy codeId:5701 Contract_hash:"6311a3f85261fc720d9a61e4ee46fae1c8a23440122b2ed1bbcebf49e3e46ad2" contract_address:"secret1j0gpu6tlwnc9fw55wcfsfuml00kqpcnqz7dck7"
```

Encrypt a payload

Now that you have your key value store smart contract uploaded on Secret Network, let's use it to store encrypted messages. Most of the ECDH cryptography has been abstracted away so there are only a few values you need to change.

cd into encrypted-payloads/evm-contract:

```
Copy cdencrypted-payloads/evm-contract
```

Openencrypt.js in evm-contract/scripts and navigate to lines 43-49 .

Update therouting_contract androuting_code_hash to the contract address andcodehash of the Secret Network smart contract that you instantiated:

```
Copy //EVM gateway contract address constpublicClientAddress="0x3879E146140b627a5C858a08e507B171D9E43139";

//the contract you want to call in secret constrouting_contract="secret1z9wdcmxdad2c07m6m8l5cwvrhmwrkexp64fck0"; constrouting_code_hash= "6311a3f85261fc720d9a61e4ee46fae1c8a23440122b2ed1bbcebf49e3e46ad2";
```

publicClientAddress is the gateway contract address for Sepolia, which is found in Secret's gateway contract docs here . Next, update lines 73-77 with the EVM wallet address associated with the private key in your env file (myAddress) , a key (anystring of your choosing), a value, (anystring of your choosing), and a viewing_key (anystring of your choosing).

value is the the data that you want to encrypt,key andviewing_key are parameters you pass to encrypt thevalue.

```
Copy constmyAddress=""; constkey=""; constvalue=""; constviewing_key="";
```

Next, you are going to set thehandle variable to call thestore_value function inside of the Secret contract that you instantiated earlier. You do this with line 80 , which corresponds to the store_value function in the Secret contract :

```
Copy lethandle=msg.handle.as_str(); matchhandle { "store_value"=>store_value(deps, env, msg.input_values, msg.task, msg.input_hash), "retrieve_value"=>retrieve_value(deps, env, msg.input_values, msg.task, msg.input_hash), _=>Err(StdError::generic_err("invalid handle".to_string())),
```

Once you have decided upon these parameters, simply runencrypt.js:

```
Copy npxhardhatrunscripts/encrypt.js--networksepolia
```

Upon successful encryption, your payload hash will be returned:

```
Copy PayloadHash:0xcd51559a345f37217a41757cdbe16e7b816f150d8b29ed1bf59fe6b7b5dbfbff
PayloadSignature:0x985085588613f7cf7cd8ca117b747ed8d9f604ee3fca9a66c3cee1ebeb1bc38f17fbe8e63438715f7f479e4e178b4022afbcf0b3967156b8ea1d84ebce7d3f321c
Recoveredpublickey:0x0423d8d8b518902cd6b0da592af0424719c355a724687cb74d96bd1171eb148edb87f3e9ca67f9ccecde109333461162af4dc09b33604c7e82242852a7142878ec
Verifythismatchestheuseraddress:0x49e01eb08bBF0696Ed0df8cD894906f7Da635929 _userAddress:0x49e01eb08bBF0696Ed0df8cD894906f7Da635929
_routingInfo:secret1pfg825wflcl40dqpd3yj96zhevnlxkh35hedks _payloadHash:0xcd51559a345f37217a41757cdbe16e7b816f150d8b29ed1bf59fe6b7b5dbfbff _info:
{"user_key":"0x02b7411401eb089a091ba7680e7f13588d8d297a4ad215aafd24eda0a397d50a2e","user_pubkey":"0x0423d8d8b518902cd6b0da592af0424719c355a724687cb74d96bd1171eb148edb87f
3","handle":"store_value","nonce":"0x3a813b8a8061a79758a2b69d","payload":"0x4814d8e4e160ca3cc3172d0ef3511b586225b490e8ea8bf08f00d82a4f097711d3edec93fb6005ed98b493a98628d9b4dd
_callbackAddress:0x3879e146140b627a5c858a08e507b171d9e43139, _callbackSelector:0x373d450c, _callbackGasLimit:300000
Transactionsent!Hash:0x7cba6149de15e42d0198a1c33548dbcaf6e1142c778f665f62b25a21e9475b57 Transactionconfirmed!BlockNumber:5412596
```

Congrats, you have encrypted your first cross-chain payload with SecretPath!

Decrypt your payload

To decrypt your encrypted payload,cd intosecret-contract/node

```
Copy cdsecret-contract/node
```

Open decrypt.js and update lines 8-9 with yourkey andviewing-key:

```
Copy constkey="secret sauce"; constviewing_key="my viewing key";
```

Then runnode decrypt:

```
Copy nodedecrypt
```

Your decrypted payload will be returned:

```
Copy { key:'secret sauce', value:'secret to the moon', message:'Retrieved value successfully' }
```

Congrats! You have now used SecretPath to encrypt and decrypt cross-chain payloads

Summary

SecretPath is a powerful addition to Secret Network's cross-chain messaging capabilities. Along with IBC and Axelar GMP , and eventually to be joined by additional bridging technologies like Wormhole and Union , it enables groundbreaking new use-cases for Web3 applications by providing access to confidential computation. This facilitates novel applications such as private voting for DAOs , secure random number generation , confidential data access control via NFTs, encrypted DeFi order books, sealed-bid auctions, and storing encrypted data .

We also encourage developers to check out our grants program to get funding for building with SecretPath, and to join our Discord and Telegram to get involved with our community. You can also contact our team directly if you have any questions about building on Secret.

Last updated3 days ago On this page *Overview * Getting Started * Configuring Environment Variables * Upload the Key value store contract on Secret Network * Encrypt a payload * Decrypt your payload * Summary

Was this helpful? Edit on GitHub Export as PDF