NO `.env` NEEDED FOR RAW PRIVATE KEYS AGAIN: How to use encrypted Private Key in your Foundry codebase

This will focus on how to use a cast-encrypted private key in your foundry codebase without providing the raw private key in your `.env` file. Hopefully this will help prevent exploits by stolen private keys.

If you don't know how to encrypt your private keys, **watch these videos** by Cyfrin Audits and Raza Zaidi .

Done? Good.

→ How do you find the PATH to your encrypted key?

To find the path to your encrypted key, run the commands below in your home directory.

```
cd ~
cd .foundry/keystores/
ls
```

Concatenate the paths and add a .json like this 👇🏾

`~/.foundry/keystores/<name-you-gave-your-encrypted-key>.json`

→ How to store the path to your encrypted key in your `.env`

```
ETH_KEYSTORE_ACCOUNT=~/.foundry/keystores/<name-you-gave-your-encrypted-key>.json
```

When reading this value from `.env` in your code, you use `vm.envString(ETH_KEYSTORE_ACCOUNT)`. Note that this reads from the keystore path (a string) to find the public key associated with the encrypted private key.

→ How can you use this in your codebase?

To use this in a codebase, I have snippets from a recent project. You can find the full project here.

> `.env`

Store the path to your encrypted key in your `.env` as described above.

> `HelperConfig.s.sol`

```
contract HelperConfig is Script {
    NetworkConfig public activeNetworkConfig;
    NetworkConfigSepolia public activeNetworkConfigSepolia; // 1

    /SNIPPED/

    struct NetworkConfig {/Specifically for Anvil/
        address wethUsdPriceFeed;
        address wbtcUsdPriceFeed;
        address weth;
        address wbtc;
        uint256 deployerKey;
    }

    struct NetworkConfigSepolia { /Specifically for Sepolia/
        address wethUsdPriceFeed;
        address wbtcUsdPriceFeed;
        address weth;
        address wbtc;
@>      string deployerKey;
    }

   /SNIPPED/

    constructor() {
        if (block.chainid == 11155111) {
@>          activeNetworkConfigSepolia = getSepoliaEthConfig();
        } else {
            activeNetworkConfig = getOrCreateAnvilEthConfig();
        }
    }

          function getSepoliaEthConfig() public view returns (NetworkConfigSepolia memory
sepoliaNetworkConfig) {
@>      sepoliaNetworkConfig = NetworkConfigSepolia ({
                    wethUsdPriceFeed: 0x694AA1769357215DE4FAC081bf1f309aDC325306, // ETH / USD
https://docs.chain.link/data-feeds/price-feeds/addresses?network=ethereum&page=1
                    wbtcUsdPriceFeed: 0x1b44F3514812d835EB1BDB0acB33d3fA3351Ee43, // BTC / USD
https://docs.chain.link/data-feeds/price-feeds/addresses?network=ethereum&page=1
            weth: 0xdd13E55209Fd76AfE204dBda4007C227904f0a81,
            wbtc: 0x8f3Cf7ad23Cd3CaDbD9735AFf958023239c6A063,
```

```
@>       deployerKey: vm.envString("ETH_KEYSTORE_ACCOUNT") // Reads encrypted SEPOLIA_PRIVATE_KEY
path from .env file. PRIVATE STUFF!

        });
    }
```

> `DeployDSC.s.sol`

```
function run() external returns (DecentralisedStableCoin, DSCEngine, HelperConfig) {
        HelperConfig helperConfig = new HelperConfig(); // This comes with our mocks!

@>      (address wethUsdPriceFeedSepolia, address wbtcUsdPriceFeedSepolia, address wethSepolia,
address wbtcSepolia, /*uint256*/string memory deployerKeySepolia) =
            helperConfig.activeNetworkConfigSepolia(); // For Sepolia
         (address wethUsdPriceFeed, address wbtcUsdPriceFeed, address weth, address wbtc, uint256
deployerKey) =
            helperConfig.activeNetworkConfig(); // For Anvil
        tokenAddresses = [weth, wbtc]; //setting token[] for dscEngine constructor
          priceFeedAddresses = [wethUsdPriceFeed, wbtcUsdPriceFeed]; //setting priceFeedAddress[]
for dscEngine constructor sequentially for each corresponding token in tokenAddresses[]

@>     vm.startBroadcast();
        // Powering up each of the contracts
        DecentralisedStableCoin dsc = new DecentralisedStableCoin();
        DSCEngine dscEngine = new DSCEngine(tokenAddresses, priceFeedAddresses, address(dsc));
        dsc.transferOwnership(address(dscEngine));
        vm.stopBroadcast();
        return (dsc, dscEngine, helperConfig);
    }
```

Notice that you do not pass any argument for `vm.startBroadcast()`.

Run your deploy with

```
forge script script/DeployDSC.s.sol:DeployDSC --fork-url $<rpc-url endpoint in .env> --account
defiProtocol2 --sender <public-key-of-encrypted-private-key>  --broadcast
```