RFC: HDF5 Encryption VFD Overview

Kyle Lofthus kyle.lofthus@lifeboat.llc
John Mainzer john.mainzer@lifeboat.llc

This document described Encryption VFD implemented for enabling data encryption in HDF5.

1 Introduction

The purpose of the Encryption VFD is to keep HDF5 files encrypted at all times, ensuring sensitive data remains secure. The VFD decrypts only the necessary data for paged I/O requests. It reads the encrypted pages for a read request into an encryption buffer and decrypts only those pages. Write I/O requests, the data is encrypted, and then written to the file.

2 Encryption

Initially, the Encryption VFD supports two encryption ciphers: AES256[1] and Twofish. Both ciphers have block sizes of 128 bits and key sizes of 256 bits, providing high security without compromising performance. Future updates may add more ciphers and key size options.

This implementation uses CIpher Block Chaining (CBC) mode. Both AES256 and Twofish encrypt 128 bit blocks of data. To prevent repetition patterns in the encrypted data, CBC incorporates randomization using an Initialization Vector (IV) the same size as the cipher blocks, which the VFD generates when encrypting pages.

Since the Encryption VFD works with paged data, encrypting a single page requires multiple cipher blocks. CBC XORs[2] the encryption key with the IV to encrypt the first block of the page, and each subsequent block is encrypted using the previous block. To further avoid repetition a new IV is generated for each page also ensuring each page is independent of other pages.

Two different page sizes are used, a plaintext and a ciphertext page size. The ciphertext page size is the plaintext page size plus any additional size needed for the encryption[3]. Ciphertext pages are used for

[1] AES keys can also be of size 128 and 196, but currently just support 256 because it's more secure.

[2] XOR (exclusive or) is a boolean operator that compares two input bits and outputs one bit.

[3] If the encryption cipher increases the size of data or requires additional information, such as an IV.

the file itself to store the IV for each page in the first block of that ciphertext page unencrypted. IVs are left unencrypted because they are random values and cannot be used alone to decrypt a page.

3 Encrypted Files and Encryption VFD Operations

To prevent incorrect cipher configurations, when a file is created the first page stores the file's cipher configuration[4], and the second page contains an encrypted test phrase to verify the configuration with the provided key. These pages are protected from accidental tampering by incorporating an encryption offset for paged I/O requests. Thus, any I/O operations to the first or second pages are redirected by treating the third page in the file as the first page, ensuring the integrity of the cipher configuration and test phrase.

3.1 Design

The first page is left as plaintext and is not encrypted, though future iterations will explore encrypting it. For strong ciphers like AES and Twofish, leaving the cipher configuration as plaintext does not weaken the security of the file, especially when the mode of operation is cipher block chaining (CBC). The first page is not encrypted because the VFD needs to know the cipher configuration used to encrypt the file to be able to decrypt the file. Encrypting the first page with a user-selected cipher poses challenges, as there is no reliable method to determine the cipher used.

However, if a user-selected cipher with known vulnerabilities is used, knowing the cipher greatly decreases the file's security, necessitating encryption of the first page. One solution we plan to explore is to use a standard cipher to encrypt the first page of all files, allowing users to select a different cipher for the rest of the file. This ensures an encrypted first page while giving users flexibility in choosing their cipher. However, this solution may require two keys if the ciphers have different key sizes.

The next iteration of the Encryption VFD will store the key(s) in a secure memory pool to protect them from being exposed to potential vulnerabilities like swapping[5] or unauthorized access. For this iteration the keys are stored in regular memory for testing purposes. Key distribution is something we have explored a few options for, as an example, allowing users to create a password for the encrypted file and using a unique salt to randomize the password, and then a Password-Based Key Derivation Function (PBKDF) to convert the password into a key of appropriate size for the cipher used. However, we will review with cybersecurity experts and modify any options and their priorities as we continue to investigate the issue and gather requirements.

[4] Example cipher configuration shown below.

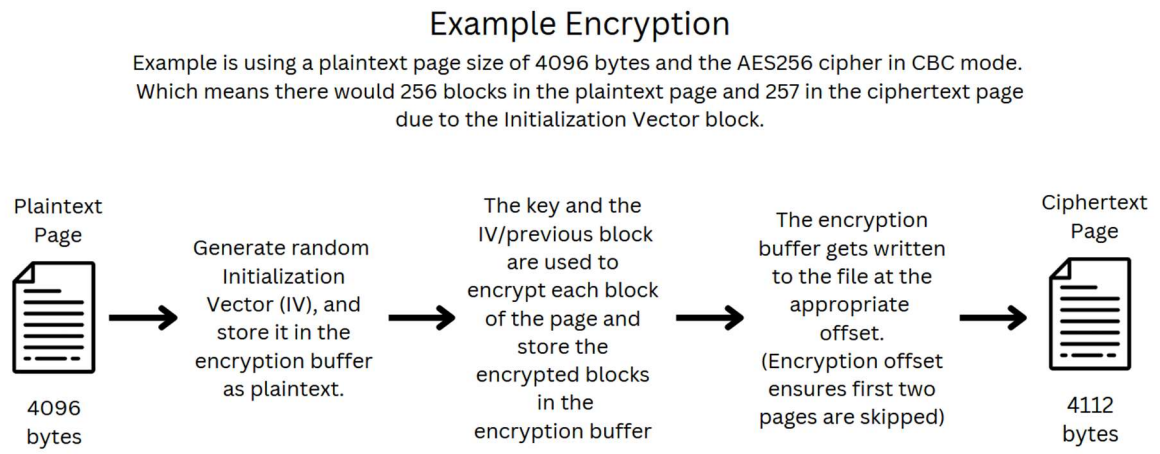
[5] Swapping: is a technique used by operating systems to manage physical memory, and leads to data stored in memory to be moved to a special area of disk space.

3.2 Example: Cipher Configuration

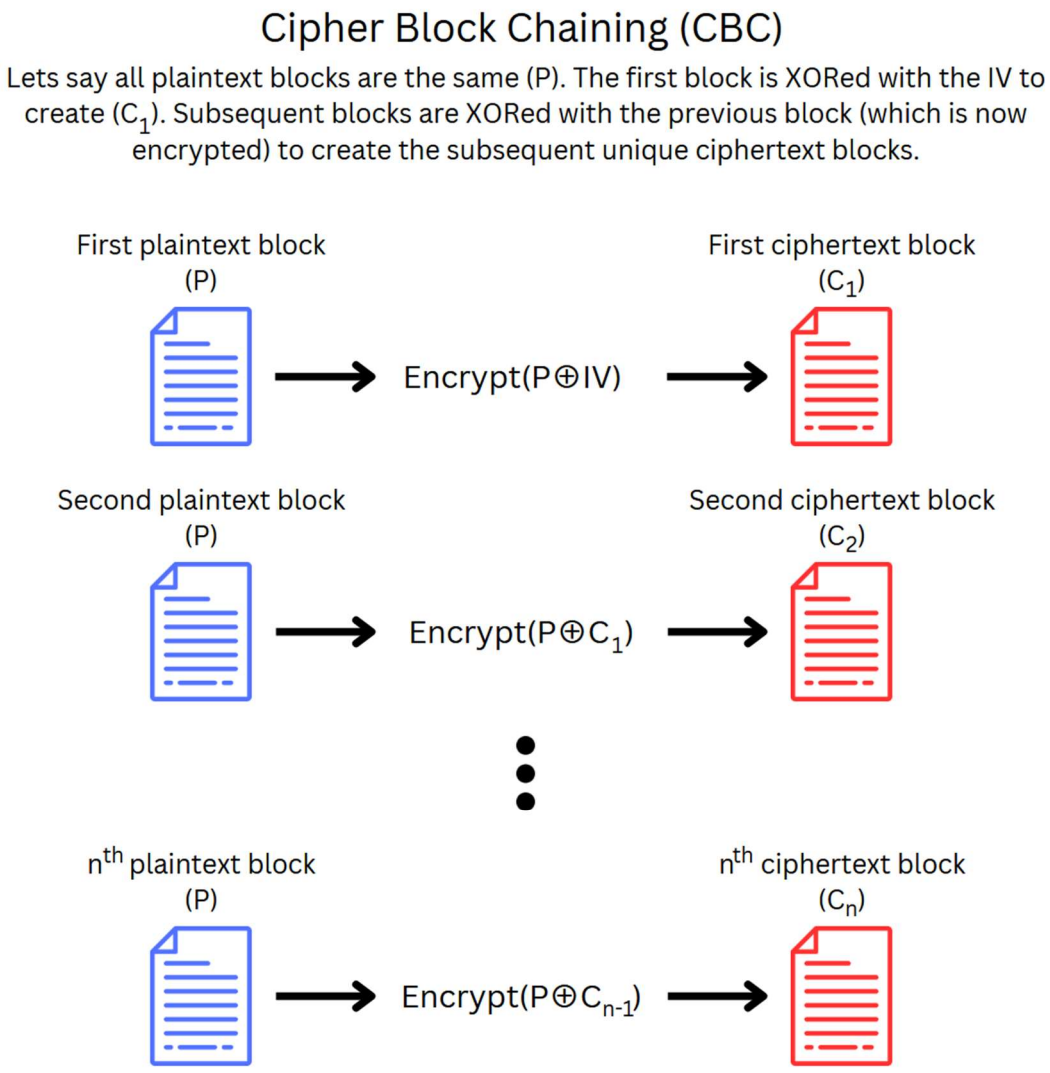
Example Cipher Configuration
(Size of data is in bytes)

Cipher Configuration		Description
cipher:	0	Integer value which represents the cipher that the VFD needs to use to encrypt/decrypt the file (0 = AES cipher).
key size:	32	Size of the key used to encrypt/decrypt the file.
cipher block size:	16	The block size the cipher can encrypt/decrypt at a time.
mode:	0	Integer value which represents the mode of operation for handling successive cipher blocks in page (0 = CBC).
iv size:	16	Size of the initialization vector (IV). If the mode of operation uses an IV, generally it is the same size as the cipher block size.
plaintext page size:	4096	Size of the plaintext pages.
ciphertext page size:	4112	Size of the ciphertext pages (larger to store the IV)
encryption buffer size:	65792	Size of the encryption buffer, which is limited to ensure there is enough space set in memory, and must be a multiple of the ciphertext page size.

3.3 Example: Encryption



3.4 Cipher Block Chaining



Acknowledgement

This work is supported by the U.S. Department of Energy, Office of Science under award number DE-SC0024823 for Phase I SBIR project “Protecting the confidentiality and integrity of data stored in HDF5"

Revision History

August 2, 2024:	Version 1 circulated for comment within Lifeboat, LLC.
August 27, 2024:	Version 2 incorporates feedback and checked into GitHub repo.