RFC: HDF5 Encryption VFD

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

The purpose of the Encryption VFD is to keep HDF5 files encrypted at all times, ensuring sensitive data remains secure. The VFD decrypts data from the HDF5 file only as required to satisfy read requests, and data in write requests is encrypted before it is written to file.

© 2024 Lifeboat, LLC

© 2024 Lifeboat, LLC

1 Introduction

The Encryption VFD requires paged I/O requests because ciphers have a set block size, which is the amount of data they can encrypt/decrypt at a time. To optimize encryption and decryption, the page size must be a multiple of the cipher's block size. This ensures efficient handling of data without requiring padding or unused space within a page, while allowing simple calculations to ensure that only the data necessary to satisfy the request is decrypted.

Two types of pages are used in the Encryption VFD, plaintext pages and ciphertext pages. Plaintext pages are pages of unencrypted data either coming into the Encryption VFD as a write request or leaving the VFD as a read request. The ciphertext pages contain the encrypted data to write to the file or to decrypt for a read request. The HDF5 file itself is made up of ciphertext pages, and the ciphertext page size is equal to the plaintext page size plus any additional space needed for the cipher or mode of operation.

2 Conceptual Overview

Read and write requests have three main parts: an address, the size of the request, and a buffer for the incoming or outgoing pages. The address of each I/O request must align with a plaintext page boundary. The Encryption VFD then converts this address to the corresponding ciphertext page boundary. Similarly, the size of each I/O request must be a multiple of the plaintext page size, which the VFD converts to the appropriate multiple of the ciphertext page size

For read requests the Encryption VFD will read the target ciphertext pages from the file via the underlying VFD stack, into the encryption buffer. The pages in the encryption buffer are then decrypted and copied into the request's buffer as plaintext pages to satisfy the request.

Write requests do the same but in the reverse order. The plaintext pages in the request's buffer are encrypted and written into the encryption buffer and are then written into the file at the appropriate location.

The encryption buffer must be a multiple of the ciphertext page size. Read or write requests larger than the encryption buffer will require the above process to be executed repeatedly to satisfy the request.

2.1 Ciphers

This initial version of the Encryption VFD supports two ciphers: AES256 and Twofish. Both ciphers have block sizes of 128 bits and key sizes of 256 bits. These are considered two of the more common and more secure symmetric ciphers.

2.2 Modes of Operation

A mode of operation is the method the encryption uses to handle multiple blocks in succession. The current version supports the mode Cipher Block Chaining (CBC). CBC mode adds randomization to the plaintext prior to encryption. It does this by using an Initialization Vector (IV), which is a random value the same size as the cipher block size. When a page is encrypted the Encryption VFD generates a random IV for the page and the first plaintext block of the page is XOR'd with the IV, and the result is

© 2024 Lifeboat, LLC

encrypted. The second plaintext block is XOR'd with the newly encrypted ciphertext block and then that result is encrypted. This creates a level of randomization to the data prior to encryption which prevents attacks that use repeating characters.

Initialization Vectors can still potentially show repetition when used on 2^{64} blocks or more. We avoid this by generating a new IV for every page and storing that IV as cleartext, which is different than plaintext, in the first block of the ciphertext page. Cleartext is non-encrypted data that is expected to not be encrypted and to have never been encrypted. This is currently the only addition in size we have for the ciphertext page, making it equal the plaintext page size plus the size of the IV. This makes each page independent from all other pages.

IVs are left unencrypted because they are random values and cannot be used alone to decrypt a page. Knowing the IV does not allow a ciphertext page to be decrypted.

2.3 Configuration Pages

To detect incorrect cipher configurations, when a file is created the first page stores the file's cipher configuration, and the second page contains an encrypted test phrase to verify the configuration with the provided key. These pages are protected from accidental modification by incorporating an encryption offset for paged I/O requests. Thus, all I/O operations are redirected to the physical page two pages further in the file than that specified, ensuring the integrity of the cipher configuration and test phrase.

In this iteration the first page is cleartext, because when using strong ciphers like AES256 and Twofish knowing the cipher configuration does not weaken the security of the file, especially when the mode of operation is CBC.

The second page containing the encrypted test phrase is encrypted like every other page in the file. It contains an IV in the first block and ciphertext after that.

2.4 Keys

Key management is still under investigation, with several options being explored. We are collaborating with cybersecurity experts to refine this process and ensure secure handling of encryption keys.

3 Encryption VFD Design

While the above discussion of the Encryption VFD is a good conceptual overview, this section goes into details of new data structures and the details of the Encryption VFD's open function.

3.1 H5FD_crypt_t

The primary structure used in the Encryption VFD is H5FD_crypt_t which contains the encryption management fields, and the fields used to manage the End of Allocation (EOA) and the End of File (EOF).

Since the Encryption VFD accepts only paged I/O, in principle, the EOA that it receives via the set EOA VFD callback should always be on a plaintext boundary. At present, the Encryption VFD is only used in combination with the Page Buffer VFD, and under these circumstances, this invariant should hold.

© 2024 Lifeboat, LLC

Since the Encryption VFD reads and writes ciphertext pages to the VFD below it in the VFD stack, this EOA must be converted to the equivalent page boundary of the ciphertext page plus two pages (to account for the configuration and test phrase pages mentioned above) before the set EOA call is relayed to the underlying VFD, and the value returned by a call to the get EOA callback to the underlying VFD must similarly be converted before it is relayed up the VFD stack.

While this conversion can be done on the fly, the EOA as seen from above (eoa_up) and the EOA as seen from below (eoa_down) are maintained for debugging purposes. The EOF requires similar conversions and thus the eof_up and eof_down fields are also maintained.

The definition of this structure is given below:

```
/****************************************************************************
 *
 * Structure:   H5FD_crypt_t
 *
 * Structure used to store all information required to manage the encryption
 * VFD.
 *
 * An instance of this structure is created when the file is "opened" and
 * discarded when the file is closed.
 *
 * The fields of this structure are discussed individually below.
 *
 * pub: An instance of H5FD_t which contains fields common to all VFDs.
 *      It must be the first item in this structure, since at higher levels,
 *      this structure will be treated as an instance of H5FD_t.
 *
 * fa:  An instance of H5FD_crypt_vfd_config_t containing all configuration
 *      data needed to setup and run the encryption.  This data is contained in
 *      an instance of H5FD_encryption_vfd_config_t for convenience in the get
 *      and set FAPL calls.
 *
 * file: Pointer to the instance of H5FD_t used to manage the underlying
 *  VFD.  Note that this VFD may or may not be terminal (i.e. perform
 *  actual I/O on a file).
 *
 * ciphertext_buf:
 *      Pointer to the dynamically allocated buffer used to storing encrypted
 *      data either loaded from file and then decrypted, on a read, or
 *      encrypted and then written to file on a write.
 *
 *      This buffer is allocated at file open time, and is of size
 *      fa->encryption_buffer_size.  Note that this size must be some positive
 *      multiple of fa->ciphertext_page_size.
 *
 *      The field should be NULL if the buffer is not allocated.
 *
 * num_ct_buf_pages:
 *      convenience field containing the size of the ciphertext_buf in
 *      ciphertext pages.  This field should be zero if the ciphertext_buf is
 *      undefined, and is computed at file open time.
 *
```

© 2024 Lifeboat, LLC

```
 * ciphertext_offset:
 *      The encrypted file has two header pages, the first of which contains
 *      configuration data. The second header page contains known encrypted
 *      phrase and is used to verify that the supplied key is correct.
 *
 *      As a result, the encrypted HDF5 file proper starts two ciphertext
 *      pages after the beginning of the file.  Since the ciphertext_page_size
 *      is variable, the ciphertext_offset is set to 2 *
 *      fa.ciphertext_page_size at file open time as a convenience in
 *      computing the base address of I/O requests to the encrypted HDf5 file.
 *
 *      ciphertext_offset is computed at file open time.
 *
 *
 * EOA / EOF management:
 *
 * The encryption VFD introduces several problems with respect to EOA / EOF
 * management.
 *
 * 1) The most obvious of these is the difference between plain text and
 *    cipher text page size.  Since the VFD stack above the encrypting
 *    VFD is un-aware of the fact that the HDF5 file is encrypted, it is
 *    necessary to interpret between the two views of the EOA and EOF above
 *    and below the encrypting VFD..
 *
 * 2) At least at present, the first two ciphertext pages of the encrypted
 *    file are used to store configuration data on the encrypted file so
 *    as to verify that this matches that passed in through the FAPL, and
 *    to store a known phrase to verify that the provided key is correct.
 *
 * 3) The encryption VFD accepts only paged I/O -- plain taxt pages above,
 *    and cipher text pages below.  From a plain text page perspective,
 *    this should be handled at higher levels in the VFD stack -- if not,
 *    the encryption VFD should flag an error as appropriate.
 *
 * All these adjustments can be done on the fly, with no need for additional
 * fields in H5FD_crypt_t.  However, the following fields are added and
 * maintained for debugging purposes.  We may choose to remove them in
 * the future.
 *
 * eoa_up: The current EOA as seen by the VFD directly above the encryption
 *      VFD in the VFD stack.  This value is set to zero at file open time,
 *      and retains that value until the first set eoa call.
 *
 * eoa_down: The current EOA as seen by the VFD directly below the encryption
 *      VFD in the VFD stack. This field is set to 2 * fa.ciphertext_page_size
 *      (which is also the value of the ciphertext_offset field) at file
 *      open time to allow the encryption VFD to read the configuration pages.
 *
 * eof_up: The current EOF as seen by the VFD directly above the encryption
 *      VFD in the VFD stack.  Note that this value is undefined until the
 *      first get_eof call is received -- in this case the field is set to
 *      HADDR_UNDEF.
 *
 * eof_down: The current EOF as seen by the VFD directly below the encryption
 *      VFD in the VFD stack.  Note that this value is undefined until the
 *      first get_eof call is received -- in this case the field is set to
 *      HADDR_UNDEF.
```

© 2024 Lifeboat, LLC

```
 *
 ****************************************************************************/

typedef struct H5FD_crypt_t {
    H5FD_t                    pub;
    H5FD_crypt_vfd_config_t   fa;
    H5FD_t                  * file;

    /* encryption management fields */
    unsigned char           * ciphertext_buf;
    uint64_t                  num_ct_buf_pages;
    haddr_t                   ciphertext_offset;

    haddr_t                   eoa_up;
    haddr_t                   eoa_down;
    haddr_t                   eof_up;
    haddr_t                   eof_down;

    /* encryption statistics fields */

    /* Must define stats and associated functions if we get
     * the Phase II.
     *                          JRM -- 9/23/24
     */
} H5FD_crypt_t;
```

3.2 File Open

On file creation, the two configuration pages are constructed from the information contained in the FAPL and written to the new file. The first page has the configuration details, and the second page encrypts the test phrase using the cipher configuration provided.

After these two pages have been written, or if the file already existed and is being opened, the Encryption VFD verifies the cipher configuration by comparing it with the contents of the first page, and decrypting the test phrase in the second page with the provided cipher configuration.

4 Testing

Due to time constraints, only a subset of the required unit tests have been implemented. More tests will be implemented in the next version and are discussed in section 4.2 Needed Unit Tests.

4.1 Current Unit Tests

Unit tests are intended to verify correct behavior of the Encryption VFD. This section will be expanded in the future as more unit tests are written.

These tests are using cipher AES256 and CBC mode:

- Test read-only access, including when the file does not exist.
- Test file creation, utilizing different child FAPLs (default vs. specified), logfile, and Write Channel error ignoring behavior.

© 2024 Lifeboat, LLC

- Test creating a file, writing to it, closing it, uses POSIX calls to manually re-open it (otherwise it would be decrypted), verifying its contents.
- Test writing and reading multiple pages, ensuring proper handling when requests exceed the encryption buffer size.

The same tests were done with the Twofish cipher, but with fewer iterations testing different numbers of pages.

4.2 Needed Unit Tests

These unit tests are tests that will be added in the future but were left out due to time constraints.

- Verify that I/O requests will be rejected if they are not paged (i.e. address is not on a page boundary, size is not a multiple of the page size, etc.)
- Test varying page sizes and encryption buffer sizes (the current tests were implemented with a plaintext page size of 4096 bytes, ciphertext page size 4112 bytes, and a buffer size of 16 * ciphertext page size).
- Verify that invalid configuration data will be rejected.

4.3 Needed Integration Tests

- Modify the “check vfd” make option to run the complete HDF5 regression test suit with the Encryption VFD.

© 2024 Lifeboat, LLC

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

September 30, 2024:	Version 1 circulated for comment within Lifeboat, LLC.
October 17, 2024:	Version 2 added copyright symbol to footnotes.

© 2024 Lifeboat, LLC