Sample Code

Encrypting and Decrypting a String

Encrypt the contents of a string and save the result to the file system, then decrypt and recreate the string from the archive file using Apple Encrypted Archive.

Download

macOS 12.0+ | Xcode 13.0+

Overview

This sample code project implements the Apple Encrypted Archive library to compress and encrypt the contents of a `String` structure using a `SymmetricKey`. The sample saves the encrypted string to the user's temporary directory and then calls a second function that decrypts the contents of the file and recreates the string.

Generate a Symmetric Key

The sample imports the Apple CryptoKit framework to create the symmetric cryptographic key.

```
let key = SymmetricKey(size: SymmetricKeySize.bits256)
```

The sample uses the same key for encryption and decryption.

Create a Context for Encryption

An `ArchiveEncryptionContext` object contains the parameters, keys, and other data that the Apple Encrypted Archive library requires to open an encrypted archive for encryption and

decryption streams. The sample initializes the context with a profile and compression algorithm, and its symmetric key set for encryption.

```
let context = ArchiveEncryptionContext(profile: .hkdf_sha256_aesctr_hmac__symmetric_
                                       compressionAlgorithm: .lzfse)
try context.setSymmetricKey(key)
```

Open the Destination File Stream

The destination file stream writes the encrypted file to the file system. In this case, the file stream's mode is `writeOnly`. The options specify that the stream creates the file if it doesn't exist, and if the file does exist, it should be truncated to zero bytes before the stream performs any operations.

```
guard let destinationFileStream = ArchiveByteStream.fileStream(
        path: destinationFilePath,
        mode: .writeOnly,
        options: [ .create, .truncate ],
        permissions: FilePermissions(rawValue: 0o644)) else {
    return
}
```

Create the Encryption Stream

The encryption stream uses the encryption context and the destination file stream to write the encrypted string to the file system.

```
guard let encryptionStream = ArchiveByteStream.encryptionStream(
        writingTo: destinationFileStream,
        encryptionContext: context) else {
    throw Error.unableToCreateEncryptionStream
}
```

Open the Encode Stream

The encoding stream encodes its data as a byte stream and sends the encoded data to the encryption stream.

```
guard let encodeStream = ArchiveStream.encodeStream(writingTo: encryptionStream) els
    return
}
```

Define the Archive Header

The archive headers contains three fields that specify the unarchived file name, the compressed file type, and the compressed file payload.

```
let header = ArchiveHeader()

// The PAT field contains the file path. Specify the unarchived file name
// for the PAT field.
header.append(.string(key: ArchiveHeader.FieldKey("PAT"),
                      value: unarchivedFileName))

// The TYP field contains the compressed file type. Specify `regularFile`
// for the TYP field.
header.append(.uint(key: ArchiveHeader.FieldKey("TYP"),
                    value: UInt64(ArchiveHeader.EntryType.regularFile.rawValue)))

// The DAT field contains the compressed file payload. Specify the size
// of the uncompressed data, in bytes, for the DAT field.
header.append(.blob(key: ArchiveHeader.FieldKey("DAT"),
                    size: UInt64(string.utf8.count)))

//  Write the header to the encode stream.
try encodeStream.writeHeader(header)
```

Write the String to the Encode Stream

The sample calls `ArchiveStream/writeBlob(key:from:)` to write the contents of the string as a data buffer to the encode stream. In turn, the encode stream writes to the compression stream, and then the encryption stream writes to the file stream. Finally, the file stream writes the archive file to the file system:

```
var mutableString = string
try mutableString.withUTF8 { textPtr in
    let rawBufferPointer = UnsafeRawBufferPointer(textPtr)
```

```
    try encodeStream.writeBlob(key: ArchiveHeader.FieldKey("DAT"),
                               from: rawBufferPointer)
}
```

On return, the file at `encryptedFilePath` exists as an AppleArchive file in the user's temporary directory and contains a single encrypted text file, specified by `unarchivedFileName`. The content of this text file is the specified string.

Open the Source File Stream

The sample creates a source file stream to open the encrypted file.

```
guard let sourceFileStream = ArchiveByteStream.fileStream(
        path: sourceFilePath,
        mode: .readOnly,
        options: [ ],
        permissions: FilePermissions(rawValue: 0o644)) else {
    throw Error.unableToCreateFileStream
}
```

Create a Context for Decryption

The `ArchiveEncryptionContext` object for decryption derives its parameters, keys, and other data from the encrypted source file, and the sample sets the decryption context with the same symmetric key that the sample used for encryption.

```
guard let decryptionContext = ArchiveEncryptionContext(from: sourceFileStream) else
    throw Error.unableToCreateDecryptionContext
}

// Set the key on the context.
try decryptionContext.setSymmetricKey(key)
```

Create the Decryption Stream

The decryption stream uses the encryption context and the source file stream to read the encrypted string from the file system.

```
guard let decryptionStream = ArchiveByteStream.decryptionStream(
    readingFrom: sourceFileStream,
    encryptionContext: decryptionContext) else {
    throw Error.unableToCreateFileStream
}
```

Open the Decode Stream

The decoding stream provides archive elements from the raw, decompressed data.

```
guard let decodeStream = ArchiveStream.decodeStream(
        readingFrom: decryptionStream) else {
    throw Error.unableToCreateFileStream
}
```

Allocate a Buffer for the Decoded Data

The sample derives the size of the decompressed and decrypted string from the DAT field of the decode stream's header.

```
let byteCount: Int = try {
    let header = try decodeStream.readHeader()
    guard
        let datField = header?.field(forKey: ArchiveHeader.FieldKey("DAT")) else {
            throw Error.unableToGetHeaderField
    }

    switch datField {
        case .blob(_, let size, _):
            return Int(size)
        default:
            return 0
    }
}()

guard byteCount != 0 else {
    throw Error.zeroDataSize
}
```

The sample uses this size to alllocate the memory that receives the decoded string.

```
let rawBufferPtr = UnsafeMutableRawBufferPointer.allocate(
        byteCount: byteCount,
        alignment: MemoryLayout<UTF8>.alignment)
defer {
    rawBufferPtr.deallocate()
}
```

Populate the Buffer with Decoded Data

The `ArchiveStream/readBlob(key:into:)` function reads the decompressed data from the DAT field and writes it to the raw buffer pointer. The decode stream parses its input from the decryption stream that, in turn, decrypts its input from the AppleArchive file supplied by the file stream.

```
try decodeStream.readBlob(key: ArchiveHeader.FieldKey("DAT"),
                          into: rawBufferPtr)
```

Initialize a String from the Raw Buffer Pointer

The sample creates a string from the raw buffer pointer by creating a typed pointer that's bound to `CChar`, and calls `init(cString:)` to initialize the new string.

```
let typedPtr = rawBufferPtr.bindMemory(to: CChar.self)
let decryptedString = String(cString: typedPtr.baseAddress!)
```

See Also

Apple Encrypted Archive essentials

{} Encrypting and Decrypting a Single File

Encrypt a single file and save the result to the file system, then decrypt and recreate the original file from the archive file using Apple Encrypted Archive.

{} Encrypting and Decrypting Directories

Compress and encrypt the contents of an entire directory or decompress and decrypt an archived directory using Apple Encrypted Archive.

```
class ArchiveEncryptionContext
```

An object that encapsulates all parameters, keys, and data necessary to open an encrypted archive for both encryption and decryption streams.