Your mission, should you choose to accept it, is to decrypt the message

Encrypted message (in hexadecimal) = 0x7f5f44465b5c5c16505b5a4755534c5c5718

Mission tasks:

To be able to decrypt the message you will need to implement a HOTP function. The function is specified in the attached document RFC4226. It describes a way to create a one-time password with a specific length of decimals. When you append the first four HOTP values you will get a new secret key. This key can decrypt the above message with the provided decrypt function. Good luck!

Details:

You will get some code in Swift. But you may use whatever language to solve the mission.

- A protocol for the CryptoLibrary
- Finished code for the decrypt function (including a XOR function)
- Data extension for Data -> hexString and hexString -> Data

Your HOTP object will take four parameters:

- cryptoLibrary: Implementation of a protocol with a HMAC function using sha1
- outputSize: Specifying the number of digits in a HOTP value
- counter: Start value of a counter incrementing with +1 for each output
- secretKey: The secret key used by the HMAC function

Write a unit test:

- Use the test vectors in the RFC4226 document (Appendix D, page 32) to make sure your HOTP function is correct.

Good to know:

- If you implement the HMAC function using Apples CryptoKit, remember that input data needs to be big endian.
- Put extra effort in understanding RFC4226 §5.4, explaining the formatted output
- The protocol uses data types to help you implementing the function with the correct number of bytes

Implementation details:

These are the values (and data types) you should initialize your HOTP object with:

```
let secretKey: Data                = Data("HID Global secretKey".asciiValues)
let counterStartValue:UInt64       = 0
let outputSize:UInt8               = 6

protocol CryptoLibrary {
   func hmac(key: Data, data: UInt64) -> Data
   func decrypt(secretKey: String, message: String) -> String
}
```

To run the decryption:

```
func decryptMessageWithHotpDataAsKey() {

      let cryptoLib = CryptoLib()

      let secretKey = Data("HID Global secretKey".asciiValues)
      let counterStartValue:UInt64 = 0
      let outputSize:UInt8 = 6

      var hotp = Hotp(cryptoLib: cryptoLib, outputSize: outputSize, counter: counterStartValue, secretKey: secretKey)
      var HOTPs: Array<Int> = []

      for _ in 0..<4 {
         HOTPs.append(hotp.generateOTP())
      }

      let generatedSecretKey = String(HOTPs[0]) + String(HOTPs[1]) + String(HOTPs[2]) + String(HOTPs[3])
      let encryptedMessage = "7f5f44465b5c5c16505b5a4755534c5c5718
"
      let decryptedMessage = cryptoLib.decrypt(secretKey: generatedSecretKey, message: encryptedMessage)
      print(generatedSecretKey)
      print(encryptedMessage)
      print(decryptedMessage)
   }
```

```
struct CryptoLib:CryptoLibrary {

   func hmac(key: Data, data: UInt64) -> Data {
      // To implement
   }

   func decrypt(secretKey: String, message: String) -> String {

      let msgData = Data(hexString: message)!
      let msg = msgData.bytes
      let key = secretKey.asciiValues
      let decoded = xor(msg,key)

      var stringMsg = ""
      for ascii in decoded {
         stringMsg.append(Character(UnicodeScalar(ascii)))
      }
      return stringMsg
   }

   func xor<T, V>(_ left: T, _ right: V) -> Array<UInt8> where T: RandomAccessCollection, V: RandomAccessCollection, T.Element == UInt8,
T.Index == Int, V.Element == UInt8, V.Index == Int {
      let length = Swift.min(left.count, right.count)

      let buf = UnsafeMutablePointer<UInt8>.allocate(capacity: length)
      buf.initialize(repeating: 0, count: length)
      defer {
         buf.deinitialize(count: length)
         buf.deallocate()
      }

      // xor
      for i in 0..<length {
         buf[i] = left[left.startIndex.advanced(by: i)] ^ right[right.startIndex.advanced(by: i)]
      }

      return Array(UnsafeBufferPointer(start: buf, count: length))
   }

}
```

```swift
extension Data {

    init?(hexString: String) {
        let len = hexString.count / 2
        var data = Data(capacity: len)
        var i = hexString.startIndex
        for _ in 0..<len {
          let j = hexString.index(i, offsetBy: 2)
          let bytes = hexString[i..<j]
          if var num = UInt8(bytes, radix: 16) {
            data.append(&num, count: 1)
          } else {
            return nil
          }
          i = j
        }
        self = data
    }

    func hex() -> String {
       return map { String(format: "%02x", $0) }
          .joined(separator: "")
    }

    var bytes: [UInt8] {
         return [UInt8](self)
      }
}
```