Decipher the encoding

Moscow 2017

Marcin Krzyżanowski

pspdfkit.com PDFViewer.io

@krzyzanowskim

krzyzanowskim.com

github.com/krzyzanowskim

CryptoSwift
ObjectivePGP
Natalie

“This is a noob question, but I wanna know why there are different encoding types and what are their differences

(ie. ASCII, utf-8 and 16, base64, etc.)”

–Coola asked on StackOverflow

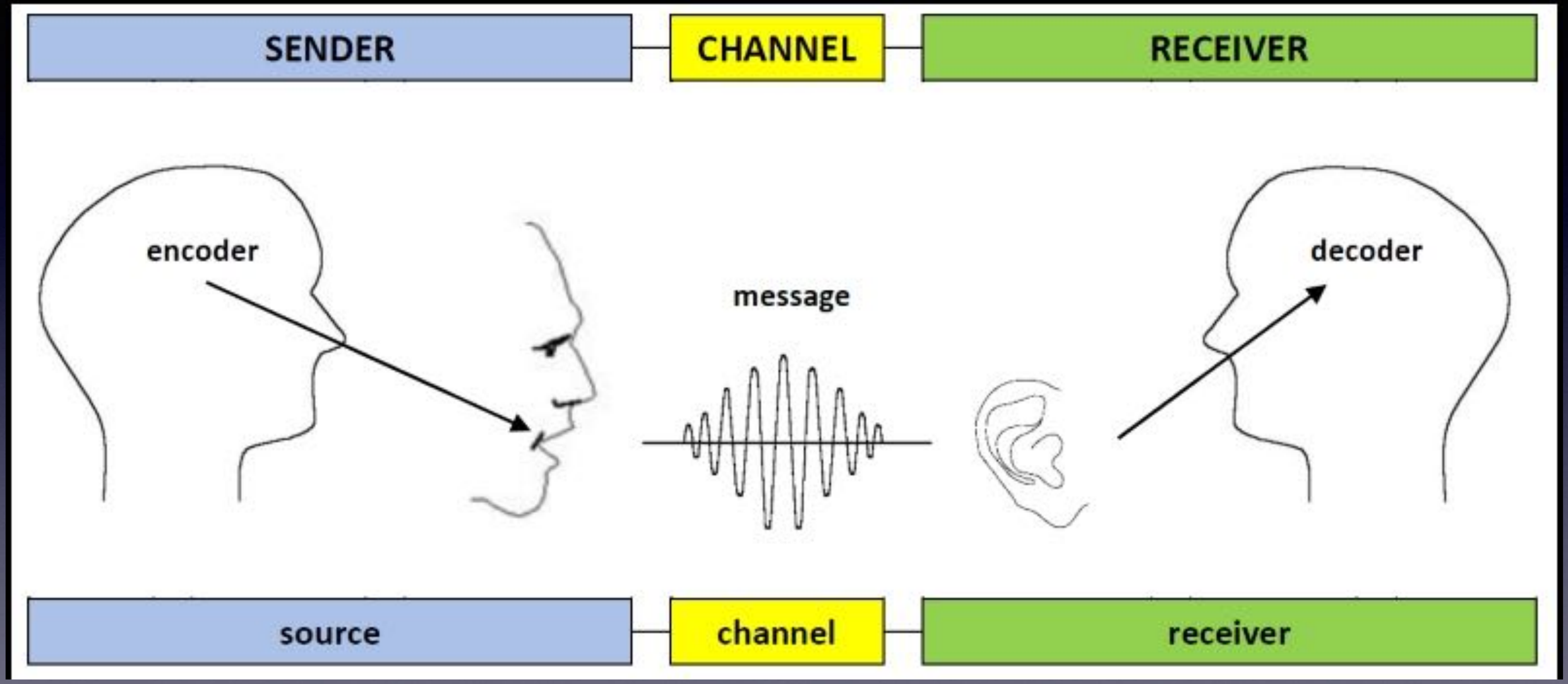
“One common mistake that people make when using managed encryption classes is that they attempt to store the result of an encryption operation in a string by using one of the Encoding classes. ”

–Shawn Farkas, .NET Security Blog

“What exactly is the difference
between encoding and encryption?

I know that, for example, **Crypt::Blowfish** is encryption,
where as **MIME::Base64** is encoding, but I don't exactly see
the difference!”

–r.joseph, perlmonks.org


SENDER
CHANNEL
RECEIVER
encoder
message
decoder
source
channel
receiver

Example

UTF-8

Unicode

U+1F1F5 U+1F1F1 U+1F985 U+1F95F U+1F34E U+1F954 U+1F372 U+1F942

U+1F95F

(RFC 3629) UTF-8, a transformation format of ISO 10646

U+1F95F

Char. number range (hexadecimal)	UTF-8 octet sequence (binary)
0000 0000-0000 007F	0xxxxxxx
0000 0080-0000 07FF	110xxxxx 10xxxxxx
0000 0800-0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx
0001 0000-0010 FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Determine the number of octets required

11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

U+1F95F

11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

U+1F95F

11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

U+1F95F

11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Fill in the bits marked “**x**”
from the bits of the **character number** expressed in binary

U+1F95F

11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Fill in the bits marked “**x**”
from the bits of the **character number** expressed in binary

1F95F

1 11111001 01011111

U+1F95F

11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

1 11111001 01011111

U+1F95F

11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

1 11111001 01011111

U+1F95F

11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

1 11111001 01011111

11110xxx 10xxxxxx 10xxxxxx 10011111

U+1F95F

```
11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
         1 11111001 01011111
11110xxx 10xxxxxx 10xxxxxx 10011111
         1 11111001 01011111
11110xxx 10xxxxxx 10100101 10011111
```

U+1F95F

11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

1 11111001 01011111

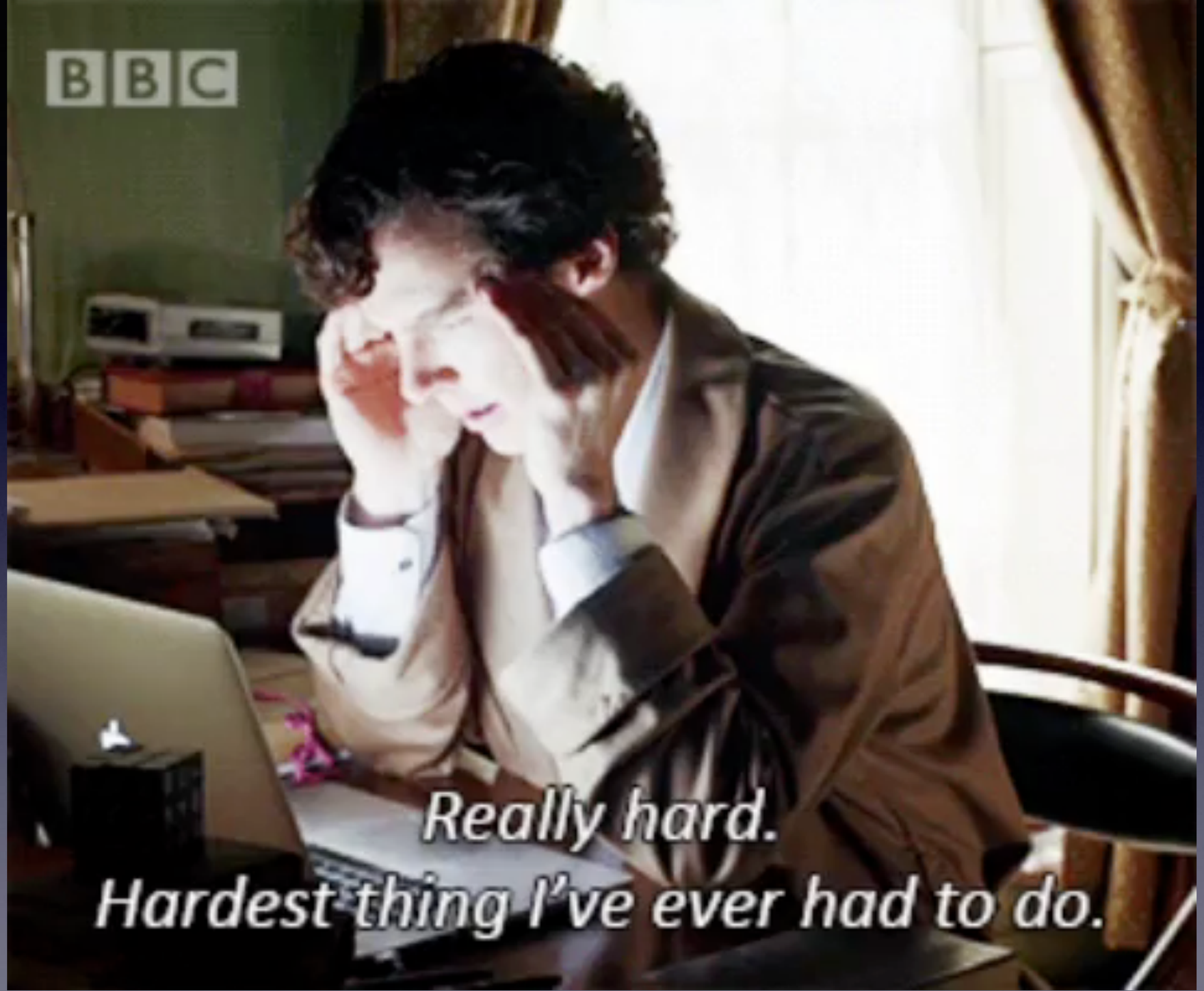
11110xxx 10xxxxxx 10xxxxxx 10011111

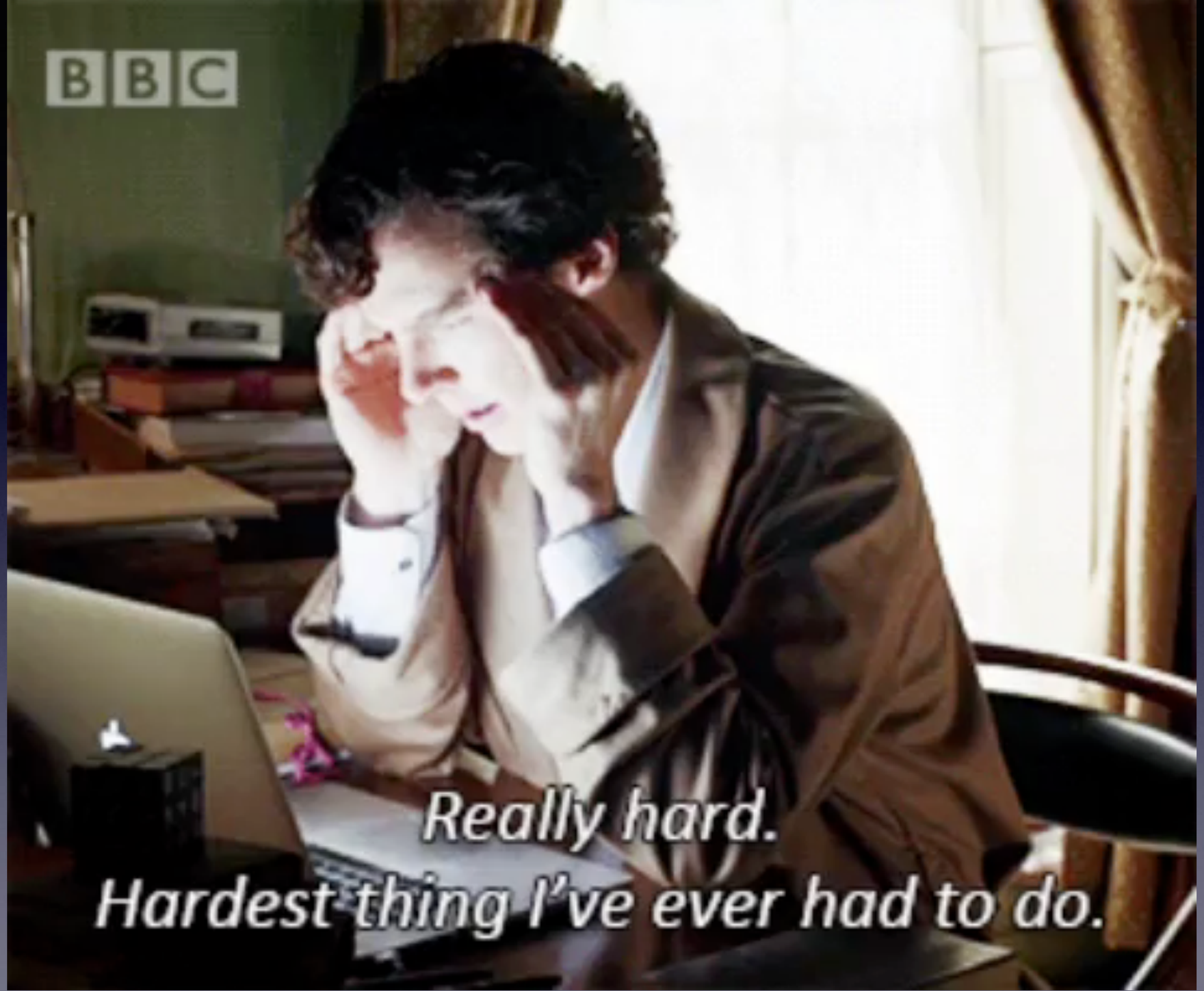
1 11111001 01011111

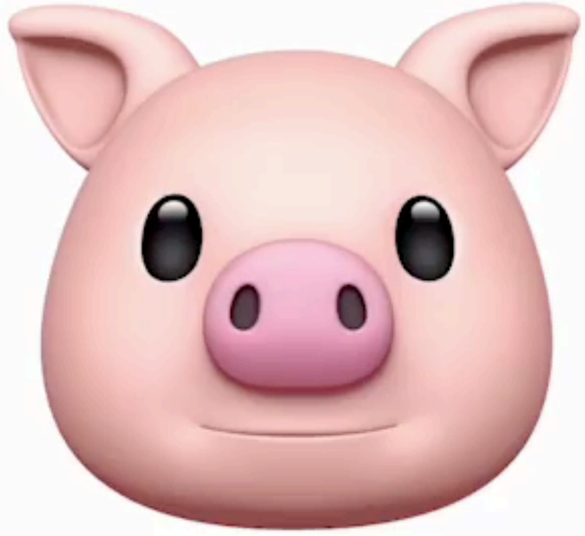
11110xxx 10xxxxxx 10100101 10011111

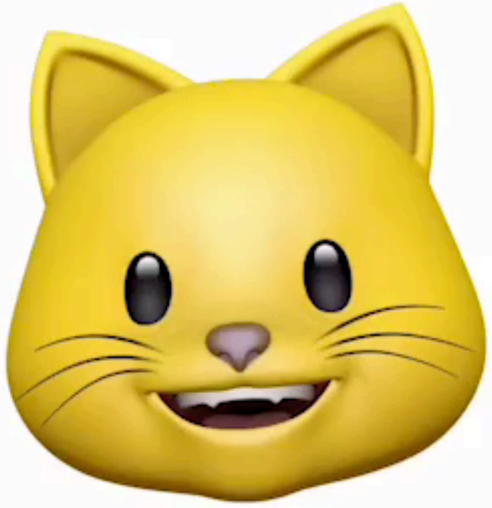
1 11111001 01011111

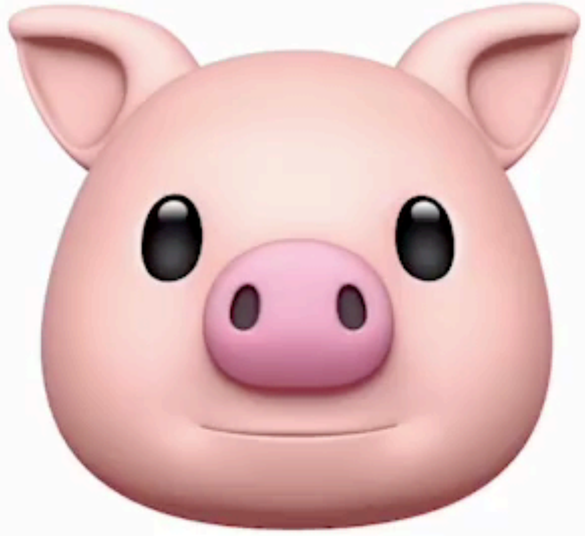
11110000 10011111 10100101 10011111

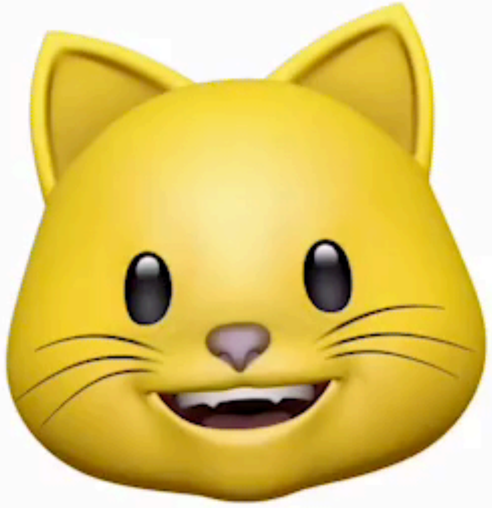

BBC
Really hard.
Hardest thing I've ever had to do.


BBC
Really hard.
Hardest thing I've ever had to do.









U+1F95F

11110000 10011111 10100101 10011111

U+1F95F

11110000 10011111 10100101 10011111

0xF0 0x9F 0xA5 0x9F **HEX**adecimal

U+1F95F

11110000	10011111	10100101	10011111	
↓	↓	↓	↓	
0xF0	0x9F	0xA5	0x9F	**HEX**adecimal
↓	↓	↓	↓	
$0\times16^0+15\times16^1$	$15\times16^0+9\times16^1$	$5\times16^0+10\times16^1$	$15\times16^0+9\times16^1$	
↓	↓	↓	↓	
240	159	165	159	**DEC**imal

U+1F95F

0xF0	0x9F	0xA5	0x9F	**HEX**adecimal
240	159	165	159	**DEC**imal

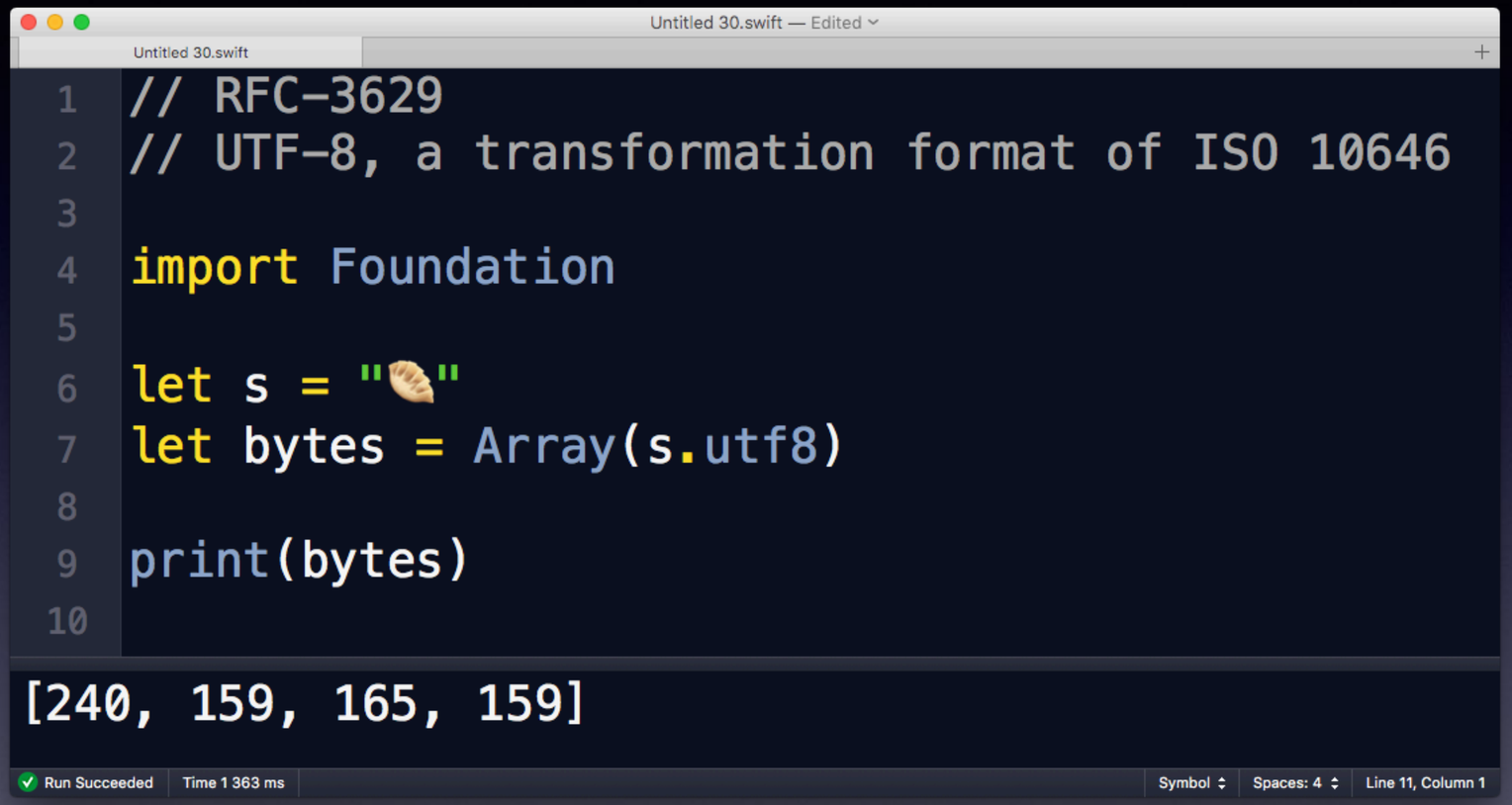
U+1F95F

F09FA59F

HEXadecimal

2678431728

DECimal



240 159 165 159

Endianness

Untitled 31.swift — Edited

Untitled 30.swift | Untitled 31.swift

```swift
// https://gist.github.com/krzyzanowskim/f311589d8dc6dc9aafc1ff9f31aec360
import Foundation

extension UInt32 {
    var bitsString: String {
        var str = String(self, radix: 2)
        for (i,j) in Swift.stride(from: UInt8.bitWidth, through: str.count - 1,
            by: UInt8.bitWidth).enumerated() {
            str.insert(" ", at: str.index(str.startIndex, offsetBy: i + j))
        }
        return str
    }
}

let num = 0xF09FA59F as UInt32
print(num.littleEndian.bitsString)
print(num.bigEndian.bitsString)
```

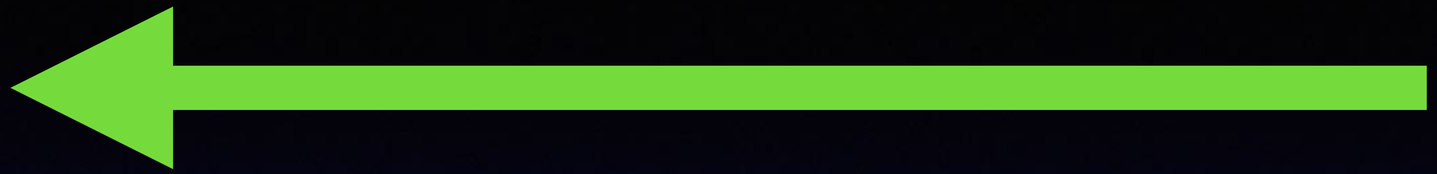
```
11110000 10011111 10100101 10011111
10011111 10100101 10011111 11110000
```

Run Succeeded | Time 1.403 ms | Symbol | Spaces: 4 | Line 5, Column 29

240 159 165 159 little-endian

240 159 165 159 little-endian

159 165 159 240 big-endian

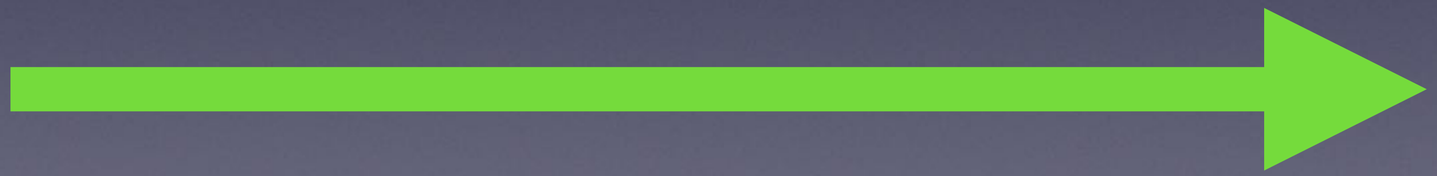


240	159	165	159

little-endian

159	165	159	240

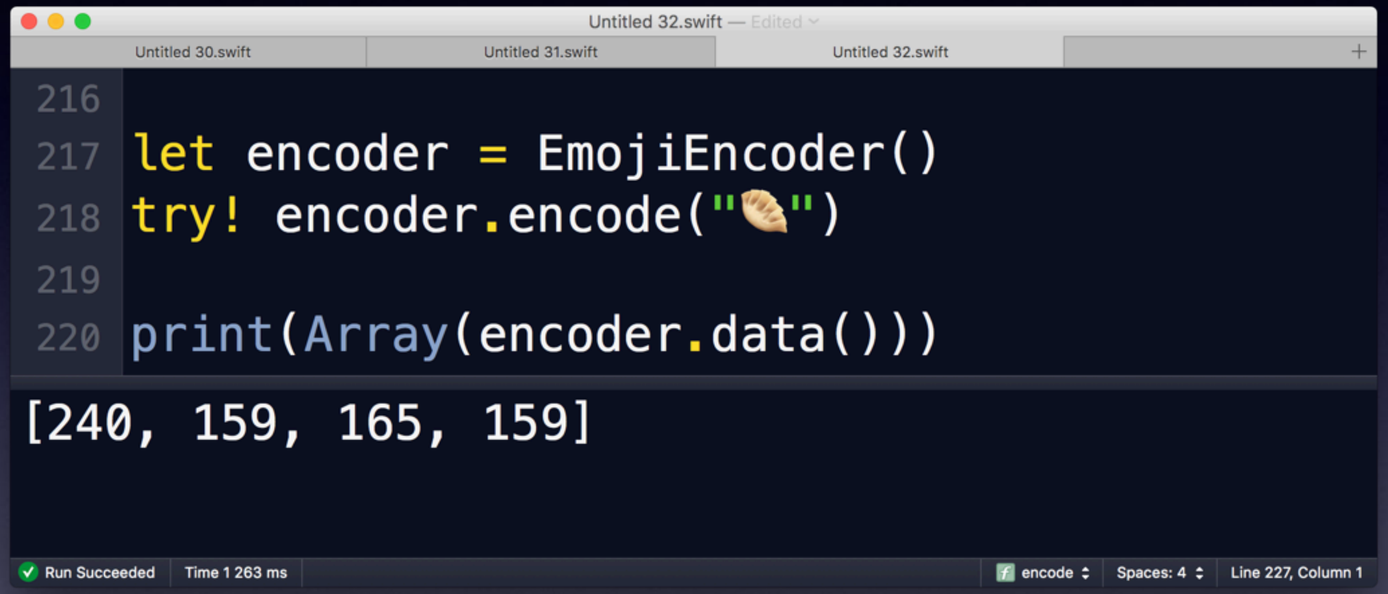
big-endian



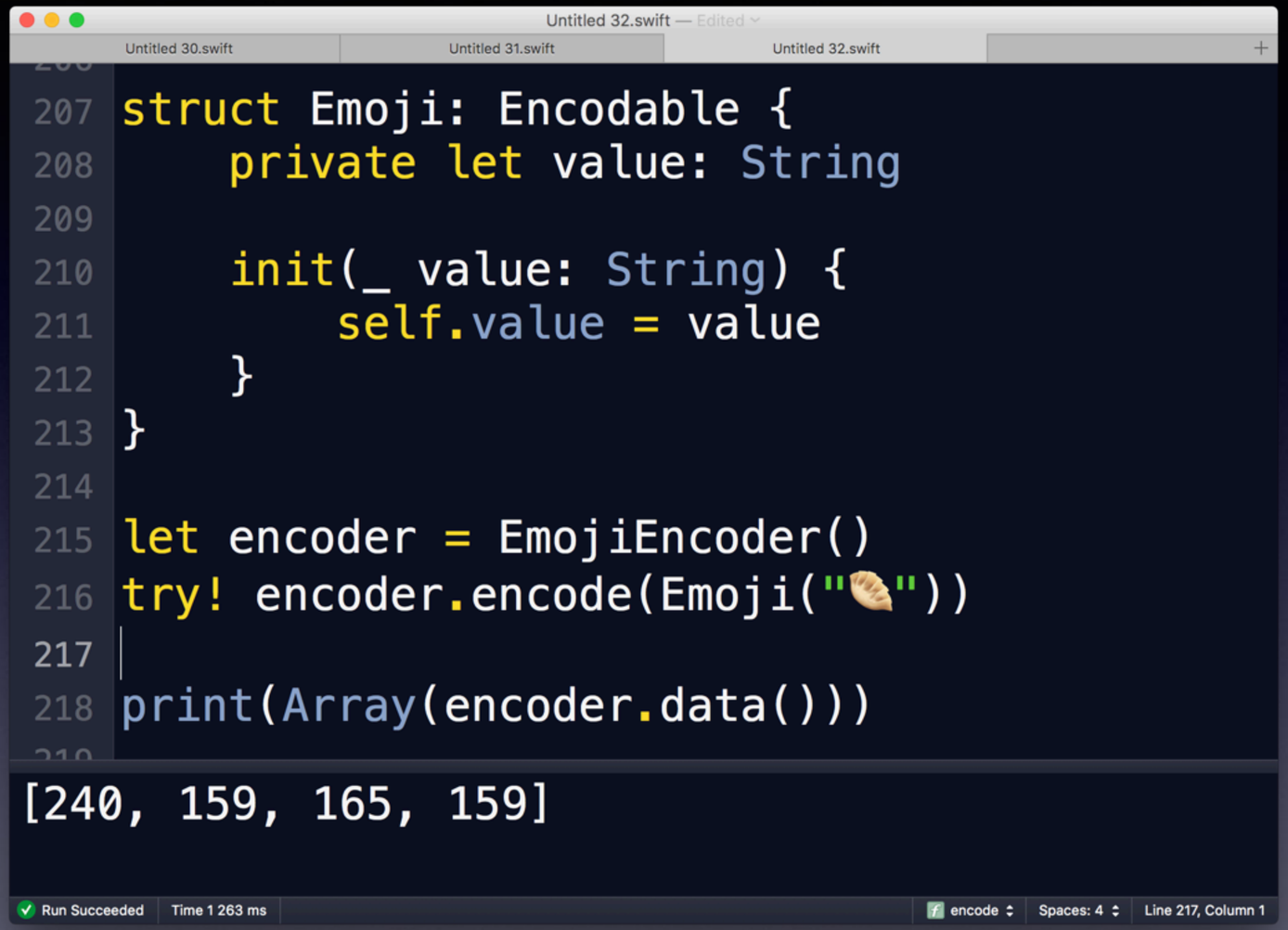
Encodable

- Encodable protocol
- Encoder (JSON, Plist, Custom)
- EmojiEncoder
  - 🥟 → [240, 159, 165, 159]
  - http://bit.ly/2jeriGl

```swift
public class EmojiEncoder: Encoder {
    public var codingPath: [CodingKey]
    public var userInfo: [CodingUserInfoKey : Any]

    fileprivate(set) var storage: [UInt8]

    public init() {
        codingPath = []
        userInfo = [:]
        storage = [UInt8]()
    }

    public func container<Key>(keyedBy: Key.Type) -> KeyedEncodingContainer<Key> {
        let container = EmojiEncodingContainer<Key>(referencing: self)
        return KeyedEncodingContainer(container)
    }

    public func singleValueContainer() -> SingleValueEncodingContainer {
        return self
    }

    public func unkeyedContainer() -> UnkeyedEncodingContainer {
        return EmojiUnkeyedEncodingContainer(referencing: self)
    }

    //

    public func data() -> Data {
        return Data(self.storage)
    }
}
```



http://bit.ly/2jeriGI



http://bit.ly/2jeriGI

Overview

Encoding

Base64

0JHQtdC70LXQtdGCINC/0LDRgNGD0YEg0L7QtNC40L3QvtC60LjQuQ0KDQrQkiDRgtGD0LzQsNC90LUg0LzQ
vtGA0Y8g0LPQvtC70YPQsdC+0LwhLi4NCg0K0KfRgtC+INC40YnQtdGCINC+0L0g0LIg0YHRgtGA0LDQvdC1
INC00LDQu9C10LrQvtC5Pw0KDQrQp9GC0L4g0LrQuNC90YPQuyDQvtC9INCyINC60YDQsNGOINGA0L7QtNC9
0L7QvD8uLg0KDQogDQoNCtCY0LPRgNCw0Y7RgiDQstC+0LvQvdGLIC0g0LLQtdGC0LXRgCDRgdCy0LjRidC1
0YIsDQoNCtCYINC80LDRh9GC0LAg0LPQvdC10YLRgdGPINC4INGB0LrRgNGL0L/QuNGCLi4uDQoNCtCj0LLR
iywgLSDQvtC9INGB0YfQsNGB0YLQuNGPINC90LUg0LjRidC10YINCg0K0Jgg0L3QtSDQvtGCINGB0YfQsNGB
0YLQuNGPINCx0LXQttC40YIhDQoNCiANCg0K0J/QvtC0INC90LjQvCDRgdGC0YDRg9GPINGB0LLQtdGC0LvQ
tdC5INC70LDQt9GD0YDQuCwNCg0K0J3QsNC0INC90LjQvCDQu9GD0Ycg0YHQvtC70L3RhtCwINC30L7Qu9C+
0YLQvtC5Li4uDQoNCtCQINC+0L0sINC80Y/RgtC10LbQvdGL0LksINC/0YDQvtGB0LjRgiDQsdGD0YDQuCwN
Cg0K0JrQsNC6INCx0YPQtNGC0L4g0LIg0LHRg9GA0Y/RhSDQtdGB0YLRjCDQv9C+0LrQvtC5IQ==

Encoding

Percent-encoding, also known as URL encoding

```
%D0%91%D0%B5%D0%BB%D0%B5%D0%B5%D1%82%20%D0%BF%D0%B0%D1%80%D1%83%D1%81%20%D0%BE%D0%B4%D0%B8%D0%BD%D0%BE%D
0%BA%D0%B8%D0%B9%0D%0A%0D%0A%D0%92%20%D1%82%D1%83%D0%BC%D0%B0%D0%BD%D0%B5%20%D0%BC%D0%BE%D1%80%D1%8F%20%
D0%B3%D0%BE%D0%BB%D1%83%D0%B1%D0%BE%D0%BC%21..%0D%0A%0D%0A%D0%A7%D1%82%D0%BE%20%D0%B8%D1%89%D0%B5%D1%82%
20%D0%BE%D0%BD%20%D0%B2%20%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B5%20%D0%B4%D0%B0%D0%BB%D0%B5%D0%BA%D0%BE%D0
%B9%3F%0D%0A%0D%0A%D0%A7%D1%82%D0%BE%20%D0%BA%D0%B8%D0%BD%D1%83%D0%BB%20%D0%BE%D0%BD%20%D0%B2%20%D0%BA%D
1%80%D0%B0%D1%8E%20%D1%80%D0%BE%D0%B4%D0%BD%D0%BE%D0%BC%3F..%0D%0A%0D%0A%20%0D%0A%0D%0A%D0%98%D0%B3%D1%8
0%D0%B0%D1%8E%D1%82%20%D0%B2%D0%BE%D0%BB%D0%BD%D1%8B%20-%20%D0%B2%D0%B5%D1%82%D0%B5%D1%80%20%D1%81%D0%B2
%D0%B8%D1%89%D0%B5%D1%82%2C%0D%0A%0D%0A%D0%98%20%D0%BC%D0%B0%D1%87%D1%82%D0%B0%20%D0%B3%D0%BD%D0%B5%D1%8
2%D1%81%D1%8F%20%D0%B8%20%D1%81%D0%BA%D1%80%D1%8B%D0%BF%D0%B8%D1%82...%0D%0A%0D%0A%D0%A3%D0%B2%D1%8B%2C%
20%20%D0%BE%D0%BD%20%D1%81%D1%87%D0%B0%D1%81%D1%82%D0%B8%D1%8F%20%D0%BD%D0%B5%20%D0%B8%D1%89%D0%B5%D1%82
%0D%0A%0D%0A%D0%98%20%D0%BD%D0%B5%20%D0%BE%D1%82%20%D1%81%D1%87%D0%B0%D1%81%D1%82%D0%B8%D1%8F%20%D0%B1%D
0%B5%D0%B6%D0%B8%D1%82%21%0D%0A%0D%0A%20%0D%0A%0D%0A%D0%9F%D0%BE%D0%B4%20%D0%BD%D0%B8%D0%BC%20%D1%81%D1%
82%D1%80%D1%83%D1%8F%20%D1%81%D0%B2%D0%B5%D1%82%D0%BB%D0%B5%D0%B9%20%D0%BB%D0%B0%D0%B7%D1%83%D1%80%D0%B8
%2C%0D%0A%0D%0A%D0%9D%D0%B0%D0%B4%20%D0%BD%D0%B8%D0%BC%20%D0%BB%D1%83%D1%87%20%D1%81%D0%BE%D0%BB%D0%BD%D
1%86%D0%B0%20%D0%B7%D0%BE%D0%BB%D0%BE%D1%82%D0%BE%D0%B9...%0D%0A%0D%0A%D0%90%20%D0%BE%D0%BD%2C%20%D0%BC%
D1%8F%D1%82%D0%B5%D0%B6%D0%BD%D1%8B%D0%B9%2C%20%D0%BF%D1%80%D0%BE%D1%81%D0%B8%D1%82%20%D0%B1%D1%83%D1%80
%D0%B8%2C%0D%0A%0D%0A%D0%9A%D0%B0%D0%BA%20%D0%B1%D1%83%D0%B4%D1%82%D0%BE%20%D0%B2%20%D0%B1%D1%83%D1%80%D
1%8F%D1%85%20%D0%B5%D1%81%D1%82%D1%8C%20%D0%BF%D0%BE%D0%BA%D0%BE%D0%B9%21
```

RFC 3986

Encoding

ASN.1

- Closely associated with a set of encoding rules that specify how to represent a data structure as a series of bytes.
- The standard ASN.1 encoding rules include
  - Distinguished Encoding Rules (DER)
  - Basic Encoding Rules (BER)
  - Canonical Encoding Rules (CER)
  - XML Encoding Rules (XER)
  - Canonical XML Encoding Rules (CXER)
  - …

Encoding

Abstract Syntax Notation (ASN)

```
FooProtocol DEFINITIONS ::= BEGIN

    FooQuestion ::= SEQUENCE {
        trackingNumber INTEGER,
        question       IA5String
    }

    FooAnswer ::= SEQUENCE {
        questionNumber INTEGER,
        answer         BOOLEAN
    }

END
```

Encoding

ASN.1 DER

```
30 — type tag indicating SEQUENCE
13 — length in octets of value that follows
  02 — type tag indicating INTEGER
  01 — length in octets of value that follows
    05 — value (5)
  16 — type tag indicating IA5String
     (IA5 means the full 7-bit ISO 646 set, including variants,
      but is generally US-ASCII)
  0e — length in octets of value that follows
    41 6e 79 62 6f 64 79 20 74 68 65 72 65 3f — value ("Anybody there?")
```

Encoding

Abstract Syntax Notation (ASN.1)

```
import Security.SecAsn1Coder

import Security.SecAsn1Templates
```

Encode and decode Distinguished Encoding Rules (DER) and Basic Encoding Rules (BER) data streams

PEM

Privacy-Enhanced Mail

Encoding

ASN.1 PEM(DER)

```
-----BEGIN CERTIFICATE-----
MIICLDCCAdKgAwIBAgIBADAKBggqhkjOPQQDAjB9MQswCQYDVQQGEwJCRTEPMA0G
A1UEChMGR251VExTMSUwIwYDVQQLExxHbnVUTFMgY2VydGlmaWNhdGUgYXV0aG9y
aXR5MQ8wDQYDVQQIEwZMZXV2ZW4xJTAjBgNVBAMTHEdudVRMUyBjZXJ0aWZpY2F0
ZSBhdXRob3JpdHkwHhcNMTEwNTIzMjAzODIxWhcNMTIxMjIyMDc0MTUxWjB9MQsw
CQYDVQQGEwJCRTEPMA0GA1UEChMGR251VExTMSUwIwYDVQQLExxHbnVUTFMgY2Vy
dGlmaWNhdGUgYXV0aG9yaXR5MQ8wDQYDVQQIEwZMZXV2ZW4xJTAjBgNVBAMTHEdu
dVRMUyBjZXJ0aWZpY2F0ZSBhdXRob3JpdHkwWTATBgcqhkjOPQIBBggqhkjOPQMB
BwNCAARS2I0jiuNn14Y2sSALCX3IybqiIJUvxUpj+oNfzngvj/Niyv2394BWnW4X
uQ4RTEiywK87WRcWMGgJB5kX/t2no0MwQTAPBgNVHRMBAf8EBTADAQH/MA8GA1Ud
DwEB/wQFAwMHBgAwHQYDVR0OBBYEFPC0gf6YEr+1KLlkQAPLzB9mTigDMAoGCCqG
SM49BAMCA0gAMEUCIDGuwD1KPyG+hRf88MeyMQcqOFZD0TbVleF+UsAGQ4enAiEA
l4wOuDwKQa+upc8GftXE2C//4mKANBC6It01gUaTIpo=
-----END CERTIFICATE-----
```

Encoding

PGP (Pretty Good Privacy)

```
-----BEGIN PGP MESSAGE-----

hQEMA6k+nfDIl5a8AQf+Jsh5KNNdMnFgJLMw85j+F4xq244NR2J/tPXC2Kf6Woyk
9FvT/wf1pAIzILtZlGhfU+uVLwl8TLI0Vz8thAijzr7bWz78oOMM54DD2CMDzgiL
BlNSboQGK1/WNgALIoXQvxIcr81NNPcC+xDs4up9qeXAnaErX9viMfxWewLWQGZ7
tsPSCojI2E4QmSp0uB2WXba4Yf3LchCQjOGI+Q+HV93peVhPWWZI8BMkq/a/GqZJ
h9g0TtDSNTs/XpVeTPM05hbh+uY8s5ht+oHosvwU22uRrdoPGybVpSahzOaiflzM
0MwFWRvssFjupWBmJuZSZR3ldhfiYmXsnSoZLf3h8dLpATsRRQVOmDTZZsec1MAk
fyfzxqMJObpPU3E/iWTzDvuT0s1XbxnS2VImiCe6bDacb9eaF98nwBUkNQCsh2hP
/Z/iNA1fVVqznQLv5UNpeDA9b1WWgmW1KbqQUUo49PKn//xz+T2R4iKIaltdyPkT
h7hex47NZEoE18YhY3vY9mEzmrl3GurqX0SbtX+bfajIjGryf6U3Pto3lkBCYQiw
BvWvZ/8Ko2vRFBEiO7Q+xP4L4EhHRK4IY0gOxb22R1SJivMH3QWaSgH7oMXhP7Tz
BdIZOpqwkGDdIk+tC+uTBqcmML3XDELyI3ZsqWoc8w7KWZ7npPtvWJ5lpSr7suSV
tyvZemcAtLLBdUI/NXC8BYNSP6FMQVPlMH2+wN8Qw2Q2yV4eyCVWnYNojBzXEAzl
DW19HvD8kQU7OBB6+BA839WvssJCgDeiUnFwzvr2rP46oxcnRy7drY7Rxd3JXKBx
36LgqJuO29U7bkX9Eil7hJanq0xgJry/gaDlWenqrxluOQ7XzVKLzKe7TQ1yBSj6
u1D/KmOu0FknJItOpWulXWgMYrcTUSv92RBEsJhBY4a/dm071SMJchLg2GaM5n3C
7K/7taSYhTHzuwec0DrT97gA+p3/F+RlrJf4/Lp35EwbcZm606SXvE6eZq6wClO2
eDMZtagwJ6RwhRSsfjiTW97AL8b1AUoyOaGVkVg2qEdmuIl2hbw/O/9p91duuB8S
+1Ptzk4WHCSrjnl1GGvEkVNwRCF5ZC7n0/YGdo1jDZXacRwrQC0wgwjhhbBufGa+
whPUHIGhW4EWbbnlpjngADVJ0U5nM6iVitR0DNOh7CROmvNkisHoW/TR+/mM4Xlt
mopES6lfS0jzqPd8FaVH1fn1S/odO9Qko/4/hpkbXa4HsuOeP2nLmj35MkMNGKk4
Z1pZEX8H4mhzSK9rqL3vP+drVFfLBcvOgAV4BJ1HRJjOFvxc2DNPECCSQT/mm5Du
JH5ulx3W5C/MHHbk+hTGFh/8
=n5KT
-----END PGP MESSAGE-----
```

Encoding

- Protocol Buffers (protobuf)
- JSON
- XML
- 1_000_000 more

Encryption


Black Magic

Cryptography


Devil itself

What exactly is the difference between encoding and encryption?

I know that, for example, **Crypt::Blowfish** is encryption, where as **MIME::Base64** is encoding, but I don't exactly see the difference!

–r.joseph, perlmonks.org

RSA

RSA

Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES)

RSA + AES

RSA + AES

RSA + AES

023968f86413l2c71965a8b83c1c5acfed9e07919436b1db15d2006be8f
d232250f0b347065753dbc1c2ba04dea296368ae0ae8429cc2825c42b0
7eb238e9f716dbc8b553ce1a3e7c3e97e1e16b62e0208fde779729df437
d5bac27327730eeff61fb5c600c7c8c69b27ae379f0fe64b

Symmetric Cipher (AES)

Key

IV (Initialization Vector)

~~ECB~~, CBC, CFB, CTR, ….

023968f8641312c71965a8b83c1c5acfed9e07919436b1db15d2006be8f
d232250f0b347065753dbc1c2ba04dea296368ae0ae8429cc2825c42b0
7eb238e9f716dbc8b553ce1a3e7c3e97e1e16b62e0208fde779729df437
d5bac27327730eeff61fb5c600c7c8c69b27ae379f0fe64b

“**One common mistake** that people make when using managed encryption classes is that they **attempt to store the result of an encryption operation in a string** by using one of the Encoding classes. ”

–Shawn Farkas, .NET Security Blog

Convert encrypted data to string? #20

Closed [redacted] opened this issue on 29 Jan 2015 · 50 comments

[redacted] commented on 29 Jan 2015

I need to send encrypted data as string to server.
I tried :
let encryptedString = NSString(data: encryptedData, encoding: NSUTF8StringEncoding)
println(encryptedString)
but the result is **nil**
So can you add a function to convert encrypted data to string?

krzyzanowskim commented on 29 Jan 2015 Owner

It is not much related to CryptoSwift, more stackoverflow question.

Anyway, depends what format is expected by your server you may expect different result. It may be Base64, or hex representation of bytes.

- base64:

```
let base64String = data.base64EncodedStringWithOptions(NSDataBase64EncodingOptions.al
```

- hex representation by reading hexString property on NSData defined in NSDataExtension.

```
let hexString = data.hexString
```

krzyzanowskim closed this on 29 Jan 2015

AES Encryption to strange characters #436

Closed

opened this issue on 23 May · 1 comment

commented on 23 May

This is my code to encrypt a string and the result is readable string: /QOEtrf3o8buv2wA9FeAyg==.
How can I get the strange character (non readable) like this: 'l  ^h  y^ꟻ.

```
var input = "CryptoSwift"
var key = "passwordpassword"
var iv = "drowssapdrowssap"

func aesEncrypt(input: String, key: String, iv: String) throws -> String {
let data = input.utf8
let encrypted = try! AES(key: key, iv: iv, blockMode: .CBC, padding: PKCS7()).encrypt(UInt8)
let encryptedData = Data(encrypted)
return encryptedData.base64EncodedString()
}

let encrypted = try! aesEncrypt(input: input, key: key, iv: iv)
print("encrypted: (encrypted)")
```

krzyzanowskim commented on 7 Jun — Owner

`/QOEtrf3o8buv2wA9FeAyg==` looks like Base64 encoded data. Don't use `base64EncodedString()` so you won't get base64 encoded data. Does that answer your question?

krzyzanowskim closed this on 7 Jun

Can't able to decrypt the data #400

Closed opened this issue on 1 Feb · 4 comments

commented on 1 Feb

Hi Everyone i am unable to decrypt the data coming the server
here are my key and iv

let key = "9876a5p4a3a2z1q0"
let iv = "0q1z2a3a4p5a6789"

The string i need to decrypt is this
"5ca9b162fe2936c0fd7f5a834d3d874a8ae39a5d482509a82b50e8ea5248cc2ce8acce5ae4c6ac16
77918ffea9e3454b894fd06d1a0baae59384aa456b3eb90b"
i am unable to do it any help it's a json string and need to decode and parse it

krzyzanowskim commented on 1 Feb Owner

it's a hex (string) encoded bytes, you can use this:

```
let bytes = Array<UInt8>(hex: "0x010203")
```

to convert it to the array of bytes.

krzyzanowskim commented on 20 Dec 2015 Owner

I think it fail on `decodedString! as String`, isn't it? that's because `decodedData` bytes can't form valid string (as expected, it's just bytes). Decoded base64 string are bytes, not string.

```swift
import CryptoSwift

let data = Data(bytes: [0x01, 0x02, 0x03])
let bytes = data.bytes // Array<UInt8>
let bytesHex = Array<UInt8>(hex: "0x010203")
let hexString = bytesHex.toHexString() // String
```

```
3 bytes
[1, 2, 3]
[1, 2, 3]
"010203"
```

super quick recap


Animoji is the best

encoding is a transcoding

Encryption is
encrypt (+ transcode) data

Don’t store encrypted data as raw String

Thank you!

@krzyzanowskim

krzyzanowskim.com