

# ECE 518 Project 1

Alan Palayil

Due Date: 10/02/2022

## Section I Overview

This project is used to understand how encryption process takes place. We are provided with the information of a 256-bit key to find a 4-digit password using SHA256 and AESGCM cryptographic hash functions, we use the Go programming language to figure out the 4-digit password using brute-force computation.

## Section II Cryptographic Hash Functions and Ciphers

1. From the output of `validateSHA256()`, decide the length of the hash generated by SHA256. Is it as expected?
  - The length in the output of `validateSHA256()`: **c0535e4be2b79ffd93291305436bf889314e4a3faec05ecffcb7df31ad9e51a** which is 32bytes and SHA256() should have 32bytes or 256bits.
2. What is the length of the nonce used for AES-GCM?
  - The length of the nonce used for AES-GCM: **3fac2e807972e1c33abfe433** is 12bytes or 96bits.
3. What is the length of the plaintext in `validateAESGCM()`?
  - The length of the plaintext in `validateAESGCM()` is **12bytes** as each character in the string is 8bits long.
4. What is the length of the byte slice returned by the `Seal` function? Why is it named `cipharmac`?
  - The length of the byte slice returned by the `Seal` function is **28 bytes**. The reason why we use `cipharmac` as the name is because we use MAC in the cryptographic checksum on the data along with the session key to detect accidental and intentional modification in the data.

## Section III Find the Password

In the `findPassword()` function with the for loop provided by the professor, we can find the correct password by generating all possible 4-digit password and compare the hashes of each of the generated password.

This implementation uses brute-force computation. We create a new SHA hash with each iteration of the 4-digit password during the for loop. We open the hash and use the functions

provided in `validateAESGCM()` and compare it with the original `ciphermac`. Using the provided `nonce`, we can use the `aeawsgcm.Open()` function which is the decryption for the seal function. The loop breaks once we reach the correct 4-digit password.

The following is the code used within the `findPassword()` function with the comments to explain the lines:

```
func findPassword() {
    nonce := make([]byte, 12)
    data := "jwang34@iit.edu"

    ciphermac, _ := hex.DecodeString(
        "7d4eb640daf43844bd605bb1c2a66046fd33cba7d2ba35828d25056c953834d93b9a04c54fa86147f62a")

    fmt.Printf("finding password for nonce=%x, data=%s, ciphermac=%x...\n",
        nonce, data, ciphermac)

    for i := 0; i < 10000; i++ {
        password := fmt.Sprintf("%04d", i)
        /*
            Modify code below to check that if sha256(password) is a correct
            key to decrypt the ciphertext with MAC in ciphermac, which is
            obtained via AES-GCM with a 0 nonce and the data being
jwang34@iit.edu
            Once found, show the correct password as well as the plaintext.
        */
        sha := sha256.New() //Creating a new SHA256
        sha.Write([]byte(password)) //SHA of the new password
        hash := sha.Sum(nil) //SHA hash
        fmt.Printf("SHA256(%s)=%x\n",
            password, hash) //Printing each 4-digit password and hash
        block, _ := aes.NewCipher(hash) //From AES function in validate.go
        aesgcm, _ := cipher.NewGCM(block) //From AES function in validate.go
        plaintext, error := aesgcm.Open(nil, nonce, []byte(ciphermac),
[]byte(data)) //From AES function in validate.go
        if error == nil {
            fmt.Printf("Correct password=%s\n", password)
            fmt.Printf("%s\n", string(plaintext))
            break
        }
    }
}
```

Output:

```
PROBLEMS 1 OUTPUT TERMINAL JUPYTER: VARIABLES DEBUG CONSOLE Filter (e.g. text, lexclude)
Starting: C:\Users\alanp\go\bin\dlv.exe dap --check-go-version=false --listen=127.0.0.1:62831 from c:\Users\alanp\Downloa
ds\prj01-go
DAP server listening at: 127.0.0.1:62831
Type 'dlv help' for list of commands.
Validated true: SHA256(Hello world!)=c0535e4be2b79ffd93291305436bf889314e4a3faec05ecffcb7df31ad9e51a
Validated true: AES-GCM(Hello world!, data=ece443, nonce=0775e11fb4736fd7de1ed6e3, key=fe983bfd58db8d651e6eb7e0c6d5f56eb5
2ec993f30847793b96bf5b150ff87d)=d9aede95ff4b4158538ef9124e75c8244e61534d69793f8df5b2807a
finding password for nonce=000000000000000000000000, data=jwang34@iit.edu, ciphermac=7d4eb640daf43844bd605bb1c2a66046df33
cba7d2ba35828d25056c953834d93b9a04c54fa86147f62a...
SHA256(0000)=9af15b336e6a9619928537df30b2e6a2376569fcf9d7e773eccede65606529a0
SHA256(0001)=888b19a43b151683c87895f6211d9f8640f97bdc8ef32f03dbe057c8f5e56d32
SHA256(0002)=4fac6dbe26e823ed6edf999c63fab3507119cf3cbfb56036511aa62e258c35b4
SHA256(0003)=446e21f212ab200933c4c9a0802e1ff0c410bbd75fca10168746fc49883096db
SHA256(0004)=0591b59c1bdd9acd2847a202ddd02c3f14f9b5a049a5707c3279c1e967745ed4
SHA256(0005)=1ed8eb363bcd64c52f5f8703ecd464008979ac2ef462e6c5df342fe56c561bd5
...
SHA256(2013)=7931aa2a1bed855457d1ddf6bc06ab4406a9fba0579045a4d6ff78f9c07c440f
SHA256(2014)=96da37e95d5cc34fe3bef6c89428df859b8a217630d0c664da1daf1539caacf5
SHA256(2015)=a85e9db4851f7cd3efb8db7bf69a07cfb97bc528b72785a9cfff7bdfef7e2279d
SHA256(2016)=da6e2f539726fabd1f8cd7c9469a22b36769137975b28abc65fe2dc29e659b77
SHA256(2017)=46e67c525617663b392a53c0e94ba79e62db62a851fb175ae87756d4e73c9718
SHA256(2018)=152e69cf3c8e76c8d8b0aed924ddd1708e4c68624611af33d52c2c2814dd5df9
SHA256(2019)=023e33504ab909cf87a6f4e4e545090e40bdc0a2153e5b68b19f7fad2b737904
SHA256(2020)=73a2af8864fc500fa49048bf3003776c19938f360e56bd03663866fb3087884a
SHA256(2021)=1bea20e1df19b12013976de2b5e0e3d1fb4ba088b59fe53642c324298b21ffd9
SHA256(2022)=b1ab1e892617f210425f658cf1d361b5489028c8771b56d845fe1c62c1fbc8b0
Correct password=2022
ECE 443 Project 1 Due 10/2
Process 25916 has exited with status 0
Detaching
dlv dap (29300) exited with code: 0
```

## Section IV Bonus: Performance Evaluation

- Time the process to compute the SHA256 hash of a 16-byte message.
  - The following is the code used for computing the time to process the SHA256 hash of a 16-byte message.

We create a SHA256time() function, in which a 16-byte message and a variable t for the duration of time. In the for loop, we start the time and create the SHA256 hash of message. Variable t is updated once the SHA is created, and we print it the time it takes to process using the Printf function and format the print.

Code:

```
func SHA256time() {
    message := make([]byte, 16) //Creating a 16-bit message
    var t time.Duration = 0      //Variable t for the duration of computation
```

```

    for i := 0; i < 100000; i++ {
        start := time.Now()           //Start time
        sha := sha256.New()           //Creating a new SHA256
        sha.Write([]byte(message))    //Taking the SHA of the message
        t += time.Since(start)
    }
    fmt.Printf("SHA256 hash computation Time=%s\n", (t / 100000)) // Formatted
    string as "2.5ns"
}

```

Output:

```

Starting: C:\Users\alanp\go\bin\dlv.exe dap --check-go-version=false --listen=127.0.0.1:62331 from c:\Users\alanp\Downloa
ds\prj01-go
DAP server listening at: 127.0.0.1:62331
Type 'dlv help' for list of commands.
Validated true: SHA256(Hello world!)=c0535e4be2b79ffd93291305436bf889314e4a3faec05ecffcbb7df31ad9e51a
Validated true: AES-GCM(Hello world!, data=ece443, nonce=7e0ccde66c9a69622582ab97, key=385b8eb4dbe92d45001084ad8cd69b7056
db426b22b5a484b5794c26e3164add)=54a72328f3f6bb02e6c91c090dfb915d7daf9acf42b56f4c68ef0bf2
SHA256 hash computation Time=49ns
finding password for nonce=00000000000000000000000000000000, data=jwang34@iit.edu, ciphermac=7d4eb640daf43844bd605bb1c2a66046fd33
cba7d2ba35828d25056c953834d93b9a04c54fa86147f62a...
Correct password=2022
ECE 443 Project 1 Due 10/2
Process 43168 has exited with status 0
Detaching
dlv dap (35476) exited with code: 0

```

From looking over the data about SHA256 hashing function, its implementation goes around 45ns for a single unpadded single block (16-byte message) and thus the performance of SHA256 computation is as expected since we got 49ns.

- Time the process to encrypt a 1M-byte message (1024\*1024 bytes) with AES-GCM using 256-bit AES key and no additional data.
  - We create the AESGCM256time() function and declare 1M-byte message along with a SHA256 hash. We take functions from validateAESGCM() to encrypt the SHA hash. Variable e and d are created to calculate the time taken for encryption and decryption using a for loop. We print it the time it takes to encrypt and decrypt using the Printf function and format the print.

Code:

```

func AESGCM256time() {
    password := fmt.Sprintf("%04d", 1234)
    plaintext := make([]byte, 1024*1024) //Creating 1M-byte message
    rand.Read(plaintext)                 //From AES function in validate.go
    sha := sha256.New()                  //Creating a new SHA256
    sha.Write([]byte(password))           //Taking the SHA of the password
}

```

```

hash := sha.Sum(nil)           //Creating a hash for the SHA
c, _ := aes.NewCipher(hash)    // create a cipher
out := make([]byte, 1024*1024) // allocate space for ciphered data
var e time.Duration = 0        //Variable e for the duration of
encryption
var d time.Duration = 0        //Variable t for the duration of
decryption
for i := 0; i < 10000; i++ {
    en_start := time.Now() //Start time
    c.Encrypt(out, []byte(plaintext))
    e += time.Since(en_start) //Add encryption time for the process
    pt := make([]byte, len(out))
    de_start := time.Now()
    c.Decrypt(pt, out)
    d += time.Since(de_start) //Add decryption time for the process
}
fmt.Printf("AESGCM256 encryption time=%s\n", (e / 100000)) // Formatted
string as "2.5ns"
fmt.Printf("AESGCM256 decrpytion time=%s\n", (d / 100000)) // Formatted
string as "2.5ns"
}

```

Output:

```

ds\prj01-go
DAP server listening at: 127.0.0.1:51660
Type 'dlv help' for list of commands.
Validated true: SHA256(Hello world!)=c0535e4be2b79ffd93291305436bf889314e4a3faec05ecffcbb7df31ad9e51a
Validated true: AES-GCM(Hello world!, data=ece443, nonce=3cebb852c12175caf4682ab3, key=0ca4eeb853bc1019fdaf250fb957fffb98
22321b145c4e7fd15ce82c5657fb64)=a907fb7e06be310f17a862209c924eb181c1c15fa86d053d69f87766
SHA256 hash computation Time=40ns
AESGCM256 encryption time=15ns
AESGCM256 decrpytion time=10ns
finding password for nonce=00000000000000000000000000000000, data=jwang34@iit.edu, ciphermac=7d4eb640daf43844bd605bb1c2a66046fd33
cba7d2ba35828d25056c953834d93b9a04c54fa86147f62a...
Correct password=2022
ECE 443 Project 1 Due 10/2
Process 20308 has exited with status 0
Detaching
dlv dap (11520) exited with code: 0
>

```

From searching the data online, the data states that a 1-M-byte message takes around 20ns and depending on the CPU the value of the encryption changes. The processing time for the code is 15ns which does come around the expected performance.

- Time the process to decrypt the message above with AES-GCM using 256-bit AES key and no additional data.
- Using the code and output from the above AES-GCM encryption, the decryption time is usually around half the time of encryption. Our decryption time is calculated as 10ns.