

# Συγχρονες εφαρμογες Ασφαλειας

Ονοματεπώνυμο : Νικόλας Φιλιππάτος

AM: 1072754

Εργασία: 5η

Ημερομηνία: November 15, 2023

- [Μέρος Α](#)
    - [1. Εγκαταστήστε το πακέτο openssl.](#)
    - [2. Όλοι οι κώδικες κρυπτογράφησης που υποστηρίζονται](#)
    - [3. Μόνο κώδικες κρυπτογράφησης που υποστηρίζουν TLSv1-3](#)
    - [4. Ευρεση των Ευπαθών](#)
    - [5. Ανάλυση κρυπτογράφησης ECDHE-ECDSA-AES128-GCM-SHA256](#)
    - [6. JavaInuse και κρυπτογράφηση με το μητρώο](#)
    - [7. Hash λειτουργίες](#)
  - [Μέρος Β](#)
    - [Instructions](#)
    - [Commands](#)
    - [Encryption Script](#)
    - [Decryption Script](#)
    - [Testing the scripts](#)
    - [Final Command](#)
-

# Μέρος A

## 1. Εγκαταστήστε το πακέτο openssl.

1. Εγκαταστήστε το πακέτο openssl.
- Ποιες εντολές χρησιμοποιήσατε?
  - Με ποια εντολή ελέγχετε η έκδοση που έχει εγκατασταθεί?

Εγκατάσταση του πακετου openssl

```
sudo apt install openssl
```

Ευρεση του version

```
openssl version
OpenSSL 3.0.11 19 Sep 2023 (Library: OpenSSL 3.0.11 19 Sep 2023)
```

## 2. Ολοι οι κώδικες κρυπτογράφησης που υποστηρίζονται

2. Με ποια εντολή βρίσκω όλους τους κώδικες κρυπτογράφησης που υποστηρίζονται?

```
openssl help
```

```
openssl ciphers -stdname
```

## 3.Μόνο κώδικες κρυπτογράφησης που υποστηρίζουν TLSv1-3

3. Με ποια εντολή βρίσκω μόνο τους κώδικες κρυπτογράφησης που υποστηρίζουν TLSv1-3?
- Για κάθε ένα από αυτούς δώστε πληροφορίες για:

- τρόπο authentication
  - αλγόριθμό κρυπτογράφηση, μέγεθος κλειδιού και mode λειτουργίας
  - Hash

```
openssl ciphers -s -v -tls1_3
```

TLS_AES_256_GCM_SHA384	TLSv1.3	Kx=any	Au=any	Enc=AESGCM(256)	Mac=AEAD
TLS_CHACHA20_POLY1305_SHA256	TLSv1.3	Kx=any	Au=any	Enc=CHACHA20/POLY1305(256)	Mac=AEAD
TLS_AES_128_GCM_SHA256	TLSv1.3	Kx=any	Au=any	Enc=AESGCM(128)	Mac=AEAD

Cipher	TLS_AES_256_GCM_SHA384	
Protocol	TLSv1.3	
Key Exchange	any	
Authentication	any	
Encryption	AESGCM(256)	Advanced Encryption Standard in Galois/Counter Mode (AES-GCM) 256-bit key
Message Authentication Code	AEAD	
Hash	SHA-384	Secure Hash Algorithm Produces a 384-bit hash value

Cipher	TLS_CHACHA20_POLY1305_SHA256	
Protocol	TLSv1.3	
Key Exchange	any	
Authentication	any	
Encryption	CHACHA20/POLY1305(256)	ChaCha20/Poly1305 256-bit key
Message Authentication Code	AEAD	
Hash	SHA-256	Secure Hash Algorithm Produces a 256-bit hash value

Cipher	TLS_AES_128_GCM_SHA256	
Protocol	TLSv1.3	
Key Exchange	any	
Authentication	any	
Encryption	AESGCM(128)	Advanced Encryption Standard in Galois/Counter Mode (AES-GCM) 128-bit key
Message Authentication Code	AEAD	
Hash	SHA-256	Secure Hash Algorithm Produces a 256-bit hash value

## 4.Ευρεση των Ευπαθη

4. Ανατρέξτε στην σελίδα <https://ciphersuite.info/> και ελέγξτε ποιοι από αυτούς είναι ευπαθείς αλγόριθμοι (weak) και ποιοι όχι.  
Από την ίδια σελίδα βρείτε ποιοι είναι οι πιο ισχυροί αλγόριθμοι κρυπτογράφησης (recommended και strong) που υποστηρίζουν TLSv1-3 ή/και TLSv1-2.

cipher	Ciphersuite
<a href="#">TLS_AES_256_GCM_SHA384</a>	Recommended
<a href="#">TLS_CHACHA20_POLY1305_SHA256</a>	Recommended
<a href="#">TLS_AES_128_GCM_SHA256</a>	Recommended

Για TLSv1\_3:

Cipher TLSv1_3	Security
<a href="#">TLS_AES_128_GCM_SHA256</a>	Recommended
<a href="#">TLS_AES_256_GCM_SHA384</a>	Recommended
<a href="#">TLS_CHACHA20_POLY1305_SHA256</a>	Recommended
<a href="#">TLS_AES_128_CCM_SHA256</a>	Secure
<a href="#">TLS_AES_128_CCM_8_SHA256</a>	Secure

Για τον TLSv1\_2

Cipher TLSv1_2	Security
<a href="#">TLS_ECCPWD_WITH_AES_128_GCM_SHA256</a>	Recommended
<a href="#">TLS_ECCPWD_WITH_AES_256_GCM_SHA384</a>	Recommended
<a href="#">TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256</a>	Recommended
<a href="#">TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384</a>	Recommended
<a href="#">TLS_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256</a>	Recommended
<a href="#">TLS_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384</a>	Recommended
<a href="#">TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256</a>	Recommended
<a href="#">TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384</a>	Recommended
<a href="#">TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256</a>	Recommended
<a href="#">TLS_ECDHE_PSK_WITH_AES_128_GCM_SHA256</a>	Recommended
<a href="#">TLS_ECDHE_PSK_WITH_AES_256_GCM_SHA384</a>	Recommended
<a href="#">TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256</a>	Recommended
<a href="#">TLS_ECCPWD_WITH_AES_128_CCM_SHA256</a>	Secure
<a href="#">TLS_ECCPWD_WITH_AES_256_CCM_SHA384</a>	Secure
<a href="#">TLS_ECDHE_ECDSA_WITH_AES_128_CCM</a>	Secure
<a href="#">TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8</a>	Secure
<a href="#">TLS_ECDHE_ECDSA_WITH_AES_256_CCM</a>	Secure

---

## 5. Αναλυση κρυπτογράφησης ECDHE-ECDSA-AES128-GCM-SHA256

5. Αναλύστε τον κώδικα κρυπτογράφησης "ECDHE-ECDSA-AES128-GCM-SHA256" και ειδικότερα τον τρόπο δημιουργίας και ανταλλαγής κλειδιών.

	Cipher	
gnutls_name	TLS_ECDHE_ECDSA_AES_128_GCM_SHA256	
openssl_name	ECDHE-ECDSA-AES128-GCM-SHA256	
hex_byte_1	0xC0	
hex_byte_2	0x2B	
protocol_version	TLS	
kex_algorithm	ECDHE	
auth_algorithm	ECDSA	
enc_algorithm	AES 128 GCM	
hash_algorithm	SHA256	
security	recommended	
tls_version	[TLS1.2, TLS1.3]	

Key Exchange :  
Elliptic Curve Diffie-Hellman Ephemeral (ECDHE)

Explanation:  
Creating temporary key pairs for each session, that are encrypted with an elliptic curve cryptography. The keys, since they are ephemeral are only used for that session, sabotaging any attempts at stealing and using them for a different session. After the public and private keys are created, the public keys are exchanged, and the message is decrypted with the private key and the received public key. The decrypted message is used to secure the future communications

---

## 6. Javainuse και κρυπτογραφηση με το μητρωο

6. Ανατρέξτε στην ιστοσελίδα: <https://www.javainuse.com/aesgenerator> και κρυπτογραφήστε τον αριθμό μητρώου σας. Επιλέξτε CBC mode, 256 μέγεθος κλειδιού και τυχαίο κλειδί Secret και Initialization Vector. Ποια η έξοδος?
- Επαληθεύστε από άλλη ιστοσελίδα: <https://www.devglan.com/online-tools/aes-encryption-decryption>
  - Τι είναι κωδικοποίηση base64? Πως μετατρέπεται σε αναγνώσιμη μορφή?

```
1GC30QcwZuSJBbyNUA1XQCw==
```

Base64 is a binary-to-text encoding algorithm. It can represent binary data in printable ASCII characters.

Base64 works with a 65-character subset of the US-ASCII charset.

The 64 first characters are mapped to an equivalent 6-bit binary sequence ( $2^6=64$ ) and = is used for padding.

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

We group the binary data into groups of six and translate them with the base64 alphabet, which we turn to decimal and then ASCII again.

Example :

Type	Data
ASCII	Hello
Binary	01001000 01100101 01101100 01101100 01101111
Group	010010 000110 010101 100110 110011 011011 000110 110011 011011 011111
Decimal	18 6 21 38 51 27 6 51 27 63
Base64	SGVsbG8=

Base64 is reversible and widely used for transferring images.

```
echo <encoded text> | base64 -d
```

---

## 7. Hash λειτουργίες

7. Τι είναι οι κρυπτογραφικές λειτουργίες hash (cryptographic hash functions) και που/πως χρησιμοποιείτε?

- Με ποια εντολή βρίσκω ποιοι hash αλγόριθμοι υποστηρίζονται από το openssl?
- Δημιουργείτε το SHA512 hash του αριθμού μητρώου σας χρησιμοποιώντας την ιστοσελίδα: <https://emn178.github.io/online-tools/sha512.html>

### Definition

A cryptographic hash function (CHF) is an algorithm that can be run on data such as an individual file or a password to produce a value called a checksum.

### Usage:

The main use of a CHF is to verify the authenticity of a piece of data. Two files can be assumed to be identical only if the checksums generated from each file, using the same cryptographic hash function, are identical. A cryptographic hash function is designed to prevent the ability to reverse the checksum they create back to the original text.

### Examples:

- Downloading Software
- Downloading Iso files
- Transferring sensitive data

Με την παρακάτω εντολή μπορούμε να βρούμε όλους τους hash αλγορίθμους που υποστηρίζονται από την openssl

```
openssl list -digest-algorithms
```

```
openssl dgst -list
```

### [sha512](#)

```
7982116112a16c550d60f037425b7c547da14a5fcf4180ed658c50c5b4d5422085a0852e6ef404bf773382c3805028020eb8f366d49d6726f396b105b4e1c56e
```

---

---

## Μέρος B

### Instructions

1. Ακολουθήστε όλες τις οδηγίες από το παρακάτω σύνδεσμο: [github/kebman](https://github.com/kebman) και κρυπτογραφήστε ένα απλό αρχείο, χρησιμοποιώντας τον αριθμό μητρώου σε κάθε αρχείο και πιο συγκεκριμένα:
  - Δημιουργία private/public κλειδιών μεγέθους 4096:  
1044545\_private.pem 1044545\_public.pem
  - Δημιουργία μηνύματος προς κρυπτογράφηση:  
1044545\_msg.txt με περιεχόμενο: Hello world: 1044545
  - Δημιουργία Hash Digest (1044545\_msg.digest.txt)
  - Κρυπτογραφημένη υπογραφή (1044545\_msg.signature.bin)
  - Χρησιμοποιείτε ως έτερο public κλειδί το kvlachos\_public.pem που είναι στο φάκελο των εγγράφων στο eclass, στον φάκελο "7. Advanced Encryption Standard" (σε zip αρχείο).
  - Υποβάλλετε σε zip ή tar αρχείο τα ακόλουθα στο eclass:
    - 1044545\_msg.b64
    - 1044545\_msg.digest.b64
    - 1044545\_msg.signature.b64
    - 1044545\_randomkey.enc.b64
    - 1044545\_public.pem (το δικό σας δημόσιο κλειδί)
  - Κάνετε upload τα ίδια αρχεία (όχι zip!! όπως είναι) εδώ:

[https://upatrasgr-my.sharepoint.com/:f/g/personal/kvlachos\\_upatras\\_gr/Eg8eC1LOpJlt3VSAVqr9JsB1QO6IZdLCLm\\_hahnlZ6UoQ](https://upatrasgr-my.sharepoint.com/:f/g/personal/kvlachos_upatras_gr/Eg8eC1LOpJlt3VSAVqr9JsB1QO6IZdLCLm_hahnlZ6UoQ)

Μετά το πέρας της καταληκτικής ημερομηνίας θα γίνει αυτόματη διόρθωση των αρχείων που υποβάλλατε και θα λάβετε αυτόματο email που θα αναφέρει είτε τα αρχεία που λείπουν είτε εάν τα αρχεία αποκρυπτογραφήθηκαν επιτυχώς.

Για όσους φοιτητές τα αρχεία αποκρυπτογραφήθηκαν σωστά, θα λάβουν 2ο email με οδηγίες για τη 2η φάση της εργασίας (αποστολή μηνύματος και κλειδιού για να αποκωδικοποιηθούν). Οδηγίες θα υπάρχουν στο email που θα σταλεί.

→ Μην χάσετε το private κλειδί σας.

---

---

## Commands

### Generating Private key :

```
openssl genrsa -aes256 -out 1072754_private.pem 4096
```

### Generating public Key :

```
openssl rsa -in 1072754_private.pem -outform PEM -pubout -out 1072754_public.pem
```

### Creating the file :

```
echo "Hello world: 1072754" > 1072754_msg.txt
```

### Random Key :

```
openssl rand 64 > 1072754_randomkey.bin
```

### Encrypt Message :

```
openssl enc -aes-256-cbc -salt -in 1072754_msg.txt -out 1072754_msg.bin -pass file:./1072754_randomkey.bin -pbkdf2
```

### Prepare the encrypted msg.bin :

```
openssl base64 -in 1072754_msg.bin -out 1072754_msg.b64
```

### Using kvlachos\_public.pem to encrypt our public key :

```
openssl rsautl -encrypt -inkey kvlachos_public.pem -pubin -in 1072754_randomkey.bin -out 1072754_randomkey.enc.bin
```

ñ

```
openssl pkeyutl -encrypt -inkey kvlachos_public.pem -pubin -in 1072754_randomkey.bin -out 1072754_randomkey.enc.bin
```

```
openssl base64 -in 1072754_randomkey.enc.bin -out 1072754_randomkey.enc.b64
```

### Creating a hash digest :

```
cat 1072754_msg.txt | openssl dgst -sha256 -binary | xxd -p > 1072754_msg.digest.txt
```

```
openssl base64 -in 1072754_msg.digest.txt -out 1072754_msg.digest.b64
```

### Make a Cryptographic Signature :

```
openssl dgst -sha256 -sign 1072754_private.pem -out 1072754_msg.signature.bin 1072754_msg.digest.txt
```

```
openssl base64 -in 1072754_msg.signature.bin -out 1072754_msg.signature.b64
```

### Zippping everything:

```
zip 1072754.zip 1072754_msg.b64 1072754_msg.digest.b64 1072754_msg.signature.b64 1072754_randomkey.enc.b64 1072754_public.pem
```

---



## Encryption Script

```
#!/bin/bash

creator(){
  echo -e "\n\n └─ Creating Folder ─ \n\n";

  mkdir "$1";

  # Generating Private Key
  echo -e "\n\n └─ Generating Private Key ─ \n\n"

  openssl genrsa -aes256 -out "$1/$1_private.pem" 4096

  # Generating public Key :
  echo -e "\n\n └─ Generating public Key ─ \n\n "

  openssl rsa -in "$1/$1_private.pem" -outform PEM -pubout -out "$1/$1_public.pem"

  # Creating the file
  echo -e "\n\n └─ Creating File ─ \n\n"

  echo "Hello world: $1" > "$1/$1_msg.txt"

  # Random Key
  echo -e "\n\n └─ Creating Random Key ─ \n\n"

  openssl rand 64 > "$1/$1_randomkey.bin"

  # Encrypt Message :
  echo -e "\n\n └─ Encrypting Message ─ \n\n"

  openssl enc -aes-256-cbc -salt -in "$1/$1_msg.txt" -out "$1/$1_msg.bin" -pass file:./"$1/$1_randomkey.bin" -pbkdf2

  # Prepare the encrypted msg.bin
  openssl base64 -in "$1/$1_msg.bin" -out "$1/$1_msg.b64"
};

encryptor(){
  ls
  # Using a second public key to encrypt our public key
  echo -e "\n\n └─ Encrypt with the public key provided ─ \n\n"

  openssl pkeyutl -encrypt -inkey "$2" -pubin -in "$1/$1_randomkey.bin" -out "$1/$1_randomkey.enc.bin"
  ls;

  openssl base64 -in "$1/$1_randomkey.enc.bin" -out "$1/$1_randomkey.enc.b64"

  # Creating a hash digest
  echo -e "\n\n └─ Creating Hash ─ \n\n"

  cat "$1/$1_msg.txt" | openssl dgst -sha256 -binary | xxd -p > "$1/$1_msg.digest.txt"

  openssl base64 -in "$1/$1_msg.digest.txt" -out "$1/$1_msg.digest.b64"

  # Making a Cryptographic Signature
  echo -e "\n\n └─ Creating Cryptographic Signature ─ \n\n"

  openssl dgst -sha256 -sign "$1/$1_private.pem" -out "$1/$1_msg.signature.bin" "$1/$1_msg.digest.txt"

  openssl base64 -in "$1/$1_msg.signature.bin" -out "$1/$1_msg.signature.b64"
};

zipper(){
  # Zipping the results
  echo -e "\n\n └─ Zipping ─ \n\n"
  zip "$1/$1.zip" "$1/$1_msg.b64" "$1/$1_msg.digest.b64" "$1/$1_msg.signature.b64" "$1/$1_randomkey.enc.b64" "$1/$1_public.pem"
};
```

```
if [ $# -eq 0 ]; then $
    &&2 echo -e "Usage: $0 [Name] [public_key] \n"
    exit 1
elif [ $# -eq 1 ]; then

    creator $1;
elif [ $# -eq 2 ]; then
    if [ ! -d "$1" ];then
        creator $1;
    fi

    encryptor $1 $2 ;
    zipper $1;
fi
```

---

## Decryption Script

```
#!/bin/bash

if [ $# -lt 2 ]; then $
    >&2 echo "Usage: $0 [Sender] [Receiver]"
    exit 1
fi

# Where sender, receiver are the names of the folders

sender=$1
receiver=$2

if [ ! -d "${sender}_decrypted" ]; then
    mkdir "${sender}_decrypted"
    echo "Created ${sender}_decrypted"
fi

# $1 is the original sender
# $2 is the original receiver

# Unbasing
echo -e "\n\n └─ Unbasing ── \n\n"

openssl base64 -d -in "${sender}/${sender}_msg.b64" -out "${sender}_decrypted/d_msg.bin"
echo "Unbased ${sender}/${sender}_msg.b64 to ${sender}_decrypted/d_msg.bin"
if [ -f "${sender}/${sender}_msg.digest.b64" ]; then
    openssl base64 -d -in "${sender}/${sender}_msg.digest.b64" -out "${sender}_decrypted/d_msg.digest.bin"
fi
if [ -f "${sender}/${sender}_msg.signature.b64" ]; then
    openssl base64 -d -in "${sender}/${sender}_msg.signature.b64" -out "${sender}_decrypted/d_msg.signature.bin"
fi

openssl base64 -d -in "${sender}/${sender}_randomkey.enc.b64" -out "${sender}_decrypted/d_randomkey.enc.bin"

echo -e " └─ Unbasing ── \n\n"

# Decrypt The Key
echo -e "\n\n └─ Decrypt The Key ── \n\n"

openssl pkeyutl -decrypt -inkey "${receiver}/${receiver}_private.pem" -in "${sender}_decrypted/d_randomkey.enc.bin" -out
"${sender}_decrypted/${receiver}_randomkey.bin"

echo -e " └─ Decrypt The Key ── \n\n"

# Decrypt the file

echo -e "\n\n └─ Decrypt the file ── \n\n"

openssl enc -d -aes-256-cbc -in "${sender}_decrypted/d_msg.bin" -out "${sender}_decrypted/d_msg.txt" -pass
file:"./${sender}_decrypted/${receiver}_randomkey.bin" -pbkdf2

echo -e " └─ Decrypt the file ── \n\n"

if [ -f "${sender}_decrypted/d_msg.digest.bin" ]; then
    # Verification
    echo -e "\n\n └─ Verification ── \n\n"

    cat "${sender}_decrypted/d_msg.txt" | openssl dgst -sha256 -binary | xxd -p > "${sender}_decrypted/d_msg.digest2.bin"

    # Find diff

    diff "${sender}_decrypted/d_msg.digest.bin" "${sender}_decrypted/d_msg.digest2.bin"

    echo -e " └─ Verification ── \n\n"
fi

if [ -f "${sender}_decrypted/d_msg.signature.bin" ]; then
    # Verifying the Signature
    echo -e "\n\n └─ Verifying the Signature ── \n\n"

    openssl dgst -sha256 -verify "${sender}/${sender}_public.pem" -signature "${sender}_decrypted/d_msg.signature.bin"
    "${sender}_decrypted/d_msg.digest.bin"

    echo -e " └─ Verifying the Signature ── \n\n"
```



---

## Testing the scripts

```
encryption_script.sh bob;  
encryption_script.sh alice;  
encryption_script.sh bob alice/alice_public.pem;  
decryption_script.sh bob alice ;  
cat bob_decrypted/d_msg.txt ;
```

## Final Command

```
./encryption_script.sh 1072754 kvlachos_public.pem
```