

Αναφορά Εργασίας 5: Κρυπτογράφηση αρχείου

Γρηγόρης Καπαδούκας (ΑΜ: 1072484)

13 Δεκεμβρίου 2022

Μέρος Α:

1

1.1

Για να εγκαταστήσω το πακέτο openssl εκτέλεσα την εξής εντολή στον Gentoo host μου (που έχει το Portage ως packet manager):

```
"sudo emerge dev-libs/openssl"
```

Σε ένα distribution με το Aptitude package manager, όπως το Debian (που έχουμε εγκατεστημένο και στο virtual machine που φτιάξαμε) εκτελούμε αντίστοιχα την εξής εντολή:

```
"sudo apt install openssl -y"
```

1.2

Για να ελέγξουμε την έκδοση του openssl που έχει εγκατασταθεί εκτελούμε την εξής εντολή:

```
"openssl version"
```

Παρακάτω παρέχω screenshot με το output της εντολής στο host Gentoo machine μου:

```
greg@Gentoo-Laptop ~ $ openssl version
OpenSSL 1.1.1q  5 Jul 2022
greg@Gentoo-Laptop ~ $
```

2

Για να βρω όλους τους κώδικες κρυπτογράφησης που υποστηρίζονται από τη συγκεκριμένη έκδοση του openssl που χρησιμοποιώ, εκτελώ την εξής εντολή:

"openssl ciphers -s"

Το "-s" σημαίνει supported. Παρακάτω παρέχω screenshot με το output της εντολής στο host Gentoo machine μου:

```
greg@Gentoo-Laptop ~ $ openssl ciphers -s
TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:
ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:
DHE-RSA-CHACHA20-POLY1305:ECDSA-AES128-GCM-SHA256:ECDSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-
AES128-GCM-SHA256:ECDSA-AES256-GCM-SHA384:ECDSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-
SHA256:ECDSA-AES128-SHA256:DHE-RSA-AES128-SHA256:ECDSA-AES256-SHA:ECDSA-AES128-SHA:DHE-RSA-AES128-SHA:AES128-
SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA:AES128-SHA
greg@Gentoo-Laptop ~ $
```

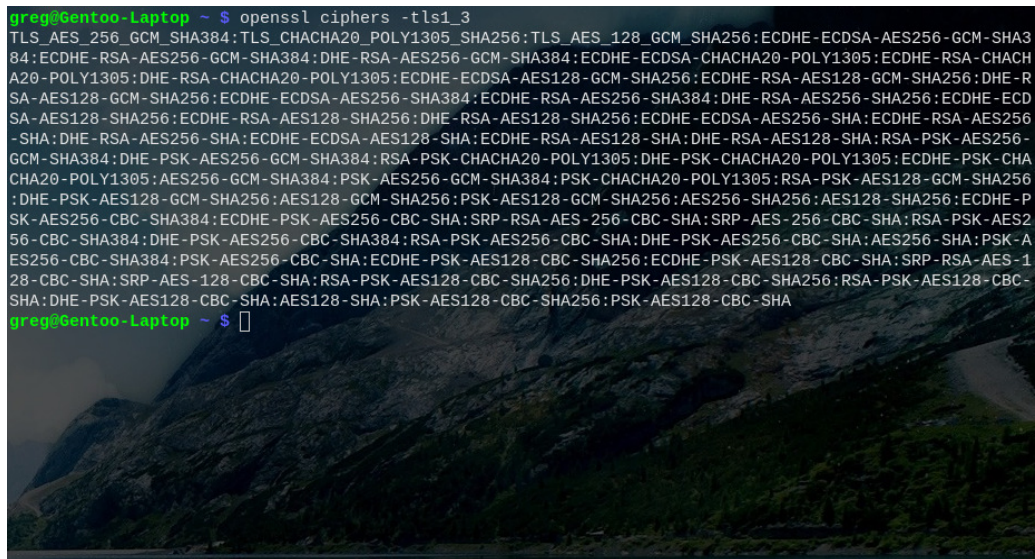
3

Για να βρω όλους τους κώδικες κρυπτογράφησης που χρησιμοποιούν TLS v1.3 και που υποστηρίζονται από τη συγκεκριμένη έκδοση του openssl που

χρησιμοποιώ, εκτελώ την εξής εντολή:

```
"openssl ciphers -tls1_3"
```

Παρακάτω παρέχω screenshot με το output της εντολής στο host Gentoo machine μου:



Στη δική μου έκδοση του openssl υπάρχουν 60 αλγόριθμοι κρυπτογράφησης που υποστηρίζουν TLS v1.3, οπότε θα παρουσιάσω τρία παραδείγματα ανάλυσης του τρόπου authentication, αλγόριθμου κρυπτογράφησης και hash παρακάτω:

1. TLS_AES_256_GCM_SHA384:

- **Τρόπος authentication:** TLS v1.3
- **Αλγόριθμος κρυπτογράφησης (μέγεθος κλειδιού και mode λειτουργίας):** AES με 256-bit κλειδί και mode GCM
- **Hash:** SHA386

2. TLS_CHACHA20_POLY1305_SHA256:

- **Τρόπος authentication:** TLS v1.3
- **Αλγόριθμος κρυπτογράφησης (μέγεθος κλειδιού και mode λειτουργίας):** CHACHA20 με 256-bit κλειδί και mode Poly1305
- **Hash:** SHA256

3. **TLS_AES_128_GCM_SHA256:**

- **Τρόπος authentication:** TLS v1.3
- **Αλγόριθμος κρυπτογράφησης (μέγεθος κλειδίου και mode λειτουργίας:** AES με 128-bit κλειδί και mode GCM
- **Hash:** SHA256

4

Ανατρέχοντας στη σελίδα ciphersuite.info, βλέπουμε τα εξής για τους τρεις αλγόριθμους που αναλύθηκαν παραπάνω:

- **TLS_AES_256_GCM_SHA384:** Recommended
- **TLS_CHACHA20_POLY1305_SHA256:** Recommended
- **TLS_AES_128_GCM_SHA256:** Recommended

Άρα κανείς από τους αλγόριθμους αυτούς αυτούς δεν είναι ευπαθής, και μάλιστα όλοι τους προτείνονται για χρήση.

Παρακάτω δίνω τους αλγόριθμους από τη σελίδα [ciphersuite](https://ciphersuite.info) που μας έδωσε, έχοντας ρυθμίσει να δίνει τους secure αλγόριθμους (recommended και strong) που υλοποιούνται με TLS v1.3 ή/και TLS v1.2.

TLS v1.3

- **TLS_AES_128_CCM_8_SHA256:** Secure
- **TLS_AES_128_GCM_SHA256:** Recommended
- **TLS_AES_256_GCM_SHA384:** Recommended
- **TLS_CHACHA20_POLY1305_SHA256:** Recommended

TLS v1.2

- **TLS_DHE_DSS_WITH_AES_128_GCM_SHA256:** Recommended
- **TLS_DHE_DSS_WITH_AES_256_GCM_SHA384:** Recommended
- **TLS_DHE_DSS_WITH_ARIA_128_GCM_SHA256:** Recommended
- **TLS_DHE_DSS_WITH_ARIA_256_GCM_SHA384:** Recommended
- **TLS_DHE_DSS_WITH_CAMELLIA_128_GCM_SHA256:** Recommended

- **TLS_DHE_DSS_WITH_CAMELLIA_256_GCM_SHA384:** Recommended
- **TLS_DHE_PSK_WITH_AES_128_GCM_SHA256:** Recommended
- **TLS_DHE_PSK_WITH_AES_256_GCM_SHA384:** Recommended
- **TLS_DHE_PSK_WITH_ARIA_128_GCM_SHA256:** Recommended
- **TLS_DHE_PSK_WITH_ARIA_256_GCM_SHA384:** Recommended
- **TLS_DHE_PSK_WITH_CAMELLIA_128_GCM_SHA256:** Recommended
- **TLS_DHE_PSK_WITH_CAMELLIA_256_GCM_SHA384:** Recommended
- **TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256:** Recommended
- **TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256:** Recommended
- **TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384:** Recommended
- **TLS_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256:** Recommended
- **TLS_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384:** Recommended
- **TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256:** Recommended
- **TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384:** Recommended
- **TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256:** Recommended
- **TLS_ECDHE_PSK_WITH_AES_128_GCM_SHA256:** Recommended
- **TLS_ECDHE_PSK_WITH_AES_256_GCM_SHA384:** Recommended
- **TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256:** Recommended

- TLS_DHE_PSK_WITH_AES_128_CCM: Secure
- TLS_DHE_PSK_WITH_AES_256_CCM: Secure
- TLS_DHE_RSA_WITH_AES_128_CCM: Secure
- TLS_DHE_RSA_WITH_AES_128_CCM_8: Secure
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256: Secure
- TLS_DHE_RSA_WITH_AES_256_CCM: Secure
- TLS_DHE_RSA_WITH_AES_256_CCM_8: Secure
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384: Secure
- TLS_DHE_RSA_WITH_ARIA_128_GCM_SHA256: Secure
- TLS_DHE_RSA_WITH_ARIA_256_GCM_SHA384: Secure
- TLS_DHE_RSA_WITH_CAMELLIA_128_GCM_SHA256: Secure
- TLS_DHE_RSA_WITH_CAMELLIA_256_GCM_SHA384: Secure
- TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256: Secure
- TLS_ECCPWD_WITH_AES_128_CCM_SHA256: Secure
- TLS_ECCPWD_WITH_AES_128_GCM_SHA256: Secure
- TLS_ECCPWD_WITH_AES_256_CCM_SHA384: Secure
- TLS_ECCPWD_WITH_AES_256_GCM_SHA384: Secure
- TLS_ECDHE_ECDSA_WITH_AES_128_CCM: Secure
- TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8: Secure
- TLS_ECDHE_ECDSA_WITH_AES_256_CCM: Secure
- TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8: Secure
- TLS_ECDHE_PSK_WITH_AES_128_CCM_8_SHA256: Secure
- TLS_ECDHE_PSK_WITH_AES_128_CCM_SHA256: Secure
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256: Secure
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384: Secure
- TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256: Secure
- TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384: Secure

- **TLS_ECDHE_RSA_WITH_CAMELLIA_128_GCM_SHA256:** Secure
- **TLS_ECDHE_RSA_WITH_CAMELLIA_256_GCM_SHA384:** Secure
- **TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256:** Secure
- **TLS_PSK_DHE_WITH_AES_128_CCM_8:** Secure
- **TLS_PSK_DHE_WITH_AES_256_CCM_8:** Secure

5

Ο κώδικας κρυπτογράφησης "ECDHE_ECDSA_AES128_GCM_SHA256" αποτελείται από τον Elliptic Curve Diffie-Hellman Ephemeral αλγόριθμο για την δημιουργία και ανταλλαγή κλειδιών, τον Elliptic Curve Digital Signature Algorithm (ECDHE) για την ταυτοποίηση των κλειδιών, τον AES 128-bit αλγόριθμο κρυπτογραφίας των δεδομένων που ανταλλάσσεται στο secure channel με mode GCM και αλγόριθμο κατακερματισμού SHA 256-bit.

Για τη δημιουργία και ανταλλαγή κλειδιών ευθύνεται ο ECDHE αλγόριθμος, ο οποίος είναι μια παραλλαγή του Diffie-Hellman αλγορίθμου, στον οποίο τα public-private keypairs δημιουργούνται με βάση μιας ελλειπτικής καμπύλης (κρυπτογραφία ελλειπτικής καμπύλης - Elliptic-curve cryptography (ECC)), το οποίο επιτυγχάνει μικρότερα κλειδιά με αντίστοιχη ασφάλεια μεγαλύτερων μη ελλειπτικής καμπύλης κλειδιά. Αυτό οδηγεί σε βελτίωση της απόδοσης του αλγορίθμου, αφού χρειάζεται λιγότερο υπολογιστικό κόστος για την χρήση των μικρότερων κλειδιών.

Έπειτα, όπως και στον κανονικό Diffie-Hellman, τα δύο μέλη που δημιουργούν το κρυπτογραφημένο κανάλι, ανταλλάσσουν τα public keys που δημιούργησαν, τα οποία χρησιμοποιούνται από το κάθε μέλος για να υπολογίσουν το κοινόχρηστο μυστικό κλειδί, που είναι ίδιο και για τα δύο μέλη και χρησιμοποιείται για να κρυπτογραφηθούν τα δεδομένα που θα αποσταλούν με συμμετρικό αλγόριθμο, σε αυτή τη περίπτωση τον AES 128-bit με GCM mode.

6

a.

Κρυπτογραφώντας το AM μου, 1072484, με αλγόριθμο AES με 256-bit key size, σε λειτουργία CBC με "jwbopz12./aabtp." ως initialization vector και "!AESEncryptionKeyFor256BitInput!" για secret key, έχω το εξής output κωδικοποιημένο σε Base64:

```
"xkPfgaXDtvBKC1mfBL3SjA=="
```

Στη σελίδα της αποκρυπτογράφησης, συμπληρώνοντας σωστά τις επιλογές καταλήγω στο εξής decrypted Base64 encoded output:

```
"MTA3MjQ4NA=="
```

Το οποίο μετά κάνοντας decode σε plaintext μου δίνει output 1072484, που είναι το AM μου που κρυπτογράφησα αρχικά. Τα '=' που βλέπουμε στις δύο Base64 αναπαραστάσεις σημαίνουν padding σε Base64, και είναι αναγκαίο για την αναπαράσταση σε Base64.

b.

Η κωδικοποίηση Base64 αποτελείται από μια συλλογή από συστήματα κωδικοποίησης δυαδικού σε κείμενο, που έχουν σκοπό την αναπαράσταση δυαδικών δεδομένων σε ακολουθία 24 bit, που αναπαρίστανται από 4 ψηφία σε Base64 των 6-bit το καθένα.

Χρησιμοποιείται για την αξιόπιστη μεταφορά δυαδικών δεδομένων σε κανάλια που υποστηρίζουν μόνο τη μεταφορά χαρακτήρων κειμένου, και έχει συχνή εφαρμογή σε ιστοσελίδες, email και σε πολλά άλλα μέρη.

Για να μετατρέψουμε ένα κείμενο από Base64 σε αναγνώσιμη μορφή χρησιμοποιούμε αρχικά τον Base64 table, όπου μετατρέπουμε το Base64 κωδικοποιημένο κείμενο αρχικά σε δυαδική μορφή, και έπειτα μετατρέπουμε το κείμενο από δυαδική μορφή σε ASCII ή Unicode (ανάλογα την περίπτωση), μέσω της χρήσης ενός ASCII table ή ενός Unicode table που δείχνει όλες τις αντιστοιχίες δυαδικών ακολουθιών σε χαρακτήρες.

7

a.

Οι κρυπτογραφικές λειτουργίες hash είναι συναρτήσεις που έχουν ως σκοπό τη μετατροπή μιας οποιασδήποτε εισόδου σε αυτές με μια μοναδική ακολουθία χαρακτήρων. Ο σκοπός είναι κάθε ακολουθία εξόδου να είναι μαθηματικά αποδεδειγμένα πως είναι μοναδική για την είσοδό του, καθώς και να είναι εξαιρετικά δύσκολη η εύρεση της αρχικής εισόδου με γνώση μόνο της εξόδου, οπότε να είναι εξαιρετικά δύσκολη υπολογιστικά η αντιστροφής συνάρτησης. Επίσης, ιδανικά θέλουμε κιάλας να είναι υπολογιστικά εύκολος ο υπολογισμός της εξόδου γνωρίζοντας την είσοδο.

Οι κρυπτογραφικές συναρτήσεις hash έχουν πολλές πρακτικές εφαρμογές, όπως την επαλήθευση της ακεραιότητας δεδομένων κατά τη μεταφορά τους, μέσω υπολογισμού του hash και τη προϋπάρχουσα γνώση της μοναδικής εξόδου που παρέχουν τα ακέρεια δεδομένα. Οπότε αν υπάρχει διαφοροποίηση στα hashes, γνωρίζουμε ότι τα δεδομένα που λήφθηκαν είναι παραλλαγμένα. Επίσης χρησιμοποιούνται σε ιστοσελίδες για τον έλεγχο των password των χρηστών. Είναι πολύ καλύτερο να γίνεται αποθήκευση των hash των passwords των χρηστών σε μια βάση δεδομένων, και τον έλεγχο κάθε φορά των hash που υπολογίζονται από το password που εισάγει κατά το login του ο χρήστης στο σύστημα με το αποθηκευμένο hash του server (χρήση της ιδιότητας της μοναδικότητας). Αυτό έχει το προτέρημα τα password των χρηστών να μην αποθηκεύονται ποτέ με σκοπό την προστασία των προσωπικών δεδομένων και του λογαριασμού του χρήστη.

Προφανώς υπάρχουν και πολλές περισσότερες λειτουργίες των hash functions και η χρησιμότητα τους περιορίζεται μόνο από την φαντασία του χρήστη.

b.

Για να βρω ποιοι hash αλγόριθμοι υποστηρίζονται από το openssl εκτελώ την εξής εντολή:

```
"openssl dgst -list ALL"
```

Παρακάτω δίνω screenshot εκτέλεσης της παραπάνω εντολής για την έκδοση του openssl που έχω εγκατεστημένο στο host μου:

```
greg@Gentoo-Laptop ~ $ openssl dgst -list ALL
Supported digests:
-blake2b512          -blake2s256          -md4
-md5                 -md5-sha1            -mdc2
-ripemd              -ripemd160           -rmd160
-sha1                -sha224              -sha256
-sha3-224            -sha3-256            -sha3-384
-sha3-512            -sha384              -sha512
-sha512-224          -sha512-256          -shake128
-shake256            -sm3                 -ssl3-md5
-ssl3-sha1           -whirlpool
```

C.

Παρακάτω δίνω το SHA512 hash του αριθμού μητρώου μου, 1072484, που υπολόγησα μέσω της ιστοσελίδας που μας δόθηκε:

"ca44b261d333c7ca1c240f07af5d0afe9bca1a82bba81d351b10ffe1b7580-0ad881e098b98f42631b729840e982987660931cd1a0d092d2cfcaa70e86-7258827"

B Μέρος:

1

Αρχικά θα δημιουργήσω το private RSA κλειδί, όπως φαίνεται παρακάτω:

```
greg@Gentoo-Laptop ~/exercise $ openssl genrsa -aes256 -out 1072484_private.pem 4096
Generating RSA private key, 4096 bit long modulus (2 primes)
.....+++++
e is 65537 (0x010001)
Enter pass phrase for 1072484_private.pem:
Verifying - Enter pass phrase for 1072484_private.pem:
greg@Gentoo-Laptop ~/exercise $
```

2

Έπειτα θα δημιουργήσω το public RSA κλειδί, χρησιμοποιώντας το private κλειδί, όπως φαίνεται παρακάτω:

```
greg@Gentoo-Laptop ~/exercise $ openssl rsa -in 1072484_private.pem -outform PEM -pubout -out 1072484_public.pem
Enter pass phrase for 1072484_private.pem:
writing RSA key
greg@Gentoo-Laptop ~/exercise $
```

3

Μετά θα δημιουργήσω το μήνυμα που θα κρυπτογραφηθεί:

```
greg@Gentoo-Laptop ~/exercise $ echo "Hello world: 1072484" > 1072484_msg.txt
greg@Gentoo-Laptop ~/exercise $
```

4

Έπειτα θα δημιουργήσω το hash digest:

```
greg@Gentoo-Laptop ~/exercise $ cat 1072484_msg.txt | openssl dgst -sha256 -binary | xxd -p > 1072484_msg.digest.txt
greg@Gentoo-Laptop ~/exercise $ openssl base64 -in 1072484_msg.digest.txt -out 1072484_msg.digest.b64
greg@Gentoo-Laptop ~/exercise $
```

5

Μετά θα δημιουργήσω τη κρυπτογραφική υπογραφή:

```
greg@Gentoo-Laptop ~/exercise $ openssl dgst -sha256 -sign 1072484_private.pem -out 1072484_msg.signature.bin 1072484_msg.digest.txt
Enter pass phrase for 1072484_private.pem:
greg@Gentoo-Laptop ~/exercise $ openssl base64 -in 1072484_msg.signature.bin -out 1072484_msg.signature.b64
greg@Gentoo-Laptop ~/exercise $
```


6

Έπειτα θα δημιουργήσω το τυχαίο κλειδί για την ομοιόμορφη κρυπτογράφηση του μηνύματος:

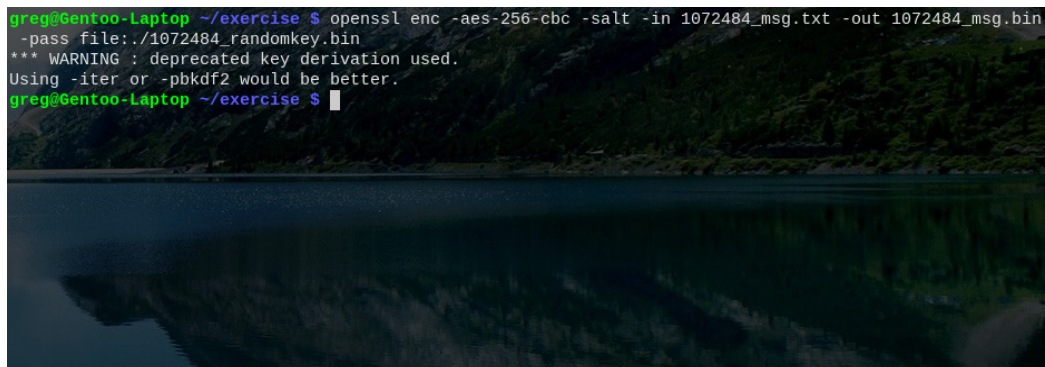
```
greg@Gentoo-Laptop ~/exercise $ openssl rand 64 > 1072484_randomkey.bin
greg@Gentoo-Laptop ~/exercise $
```



7

Μετά θα κρυπτογραφήσω ομοιόμορφα το μήνυμα με το τυχαίο κλειδί που έφτιαξα παραπάνω:

```
greg@Gentoo-Laptop ~/exercise $ openssl enc -aes-256-cbc -salt -in 1072484_msg.txt -out 1072484_msg.bin
-pass file:./1072484_randomkey.bin
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
greg@Gentoo-Laptop ~/exercise $
```

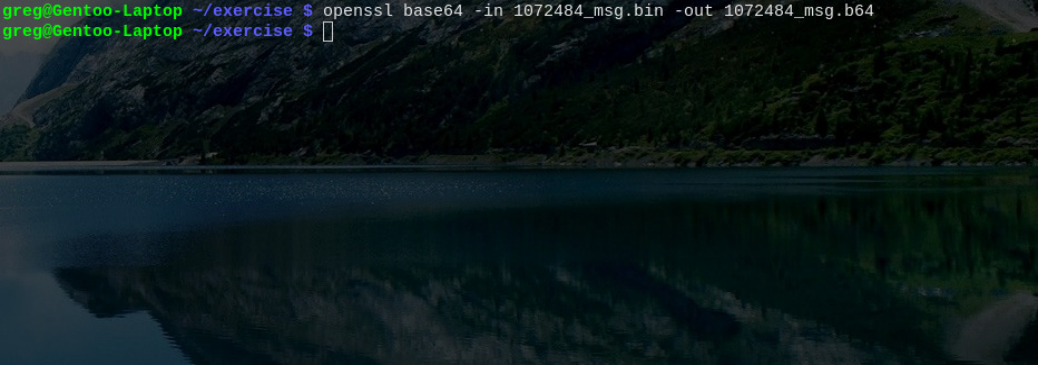


Εδώ θα μπορούσα να χρησιμοποιήσω την pbkdf2 κλειδί για αυξημένη ασφάλεια, αλλά αναφέρθηκε σε ανακοίνωση του eclass να μην γίνει αυτό, ώστε να είναι υπάρχει ομοιογένεια στις υποβολές, άρα δεν το έκανα.

8

Έπειτα θα κωδικοποιήσω το κρυπτογραφημένο μήνυμα με base64, όπως ζητείται:

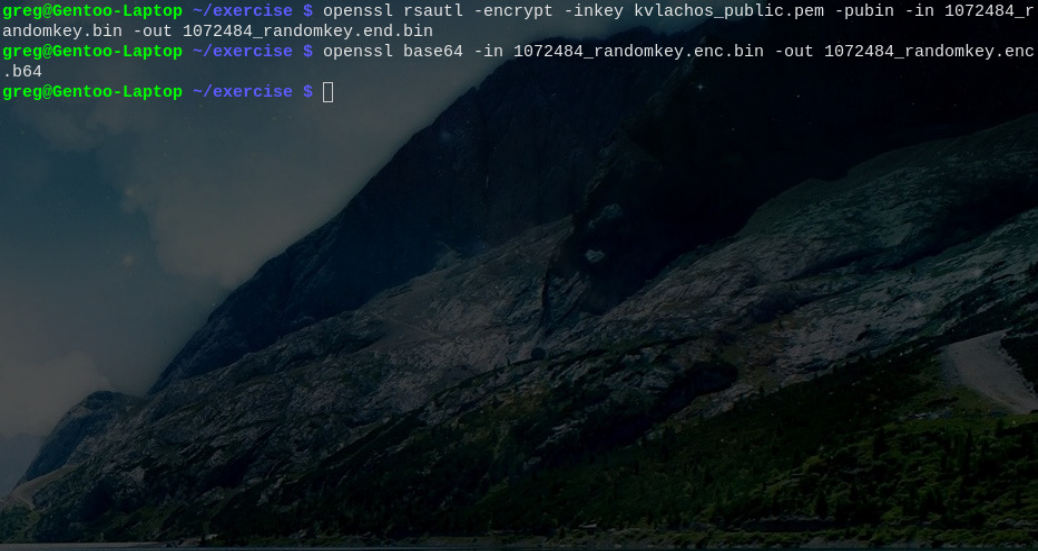
```
greg@Gentoo-Laptop ~/exercise $ openssl base64 -in 1072484_msg.bin -out 1072484_msg.b64
greg@Gentoo-Laptop ~/exercise $
```



9

Έπειτα θα κρυπτογραφήσω το κλειδί που χρησιμοποιήθηκε για την ομοιόμορφη κρυπτογράφηση του μηνύματος, με μη ομοιόμορφη κρυπτογράφηση χρησιμοποιώντας το public κλειδί που βρίσκεται στο eclass, ώστε να μπορεί να αποκρυπτογραφηθεί μόνο με χρήση του private κλειδί στην κατοχή σας. Κατόπιν της κρυπτογραφίας, θα κωδικοποιήσω το κρυπτογραφημένο κλειδί σε base64 μορφή.

```
greg@Gentoo-Laptop ~/exercise $ openssl rsautl -encrypt -inkey kvlachos_public.pem -pubin -in 1072484_randomkey.bin -out 1072484_randomkey.end.bin
greg@Gentoo-Laptop ~/exercise $ openssl base64 -in 1072484_randomkey.end.bin -out 1072484_randomkey.enc.b64
greg@Gentoo-Laptop ~/exercise $
```



10

Τέλος ανέβασα τα αρχεία που ζητήθηκαν στο σύνδεσμο του nextcloud όπως ζητήθηκε.