ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ

RSA,Diffie-Hellman

σε πρωτόκολλα ασφαλούς σύνδεσης στο διαδίκτυο

Μάρκος Καραμέρης, ΗΜΜΥ ΕΜΠ

Βασικές έννοιες που θα χρειαστούμε

Ομάδα

Ένα σύνολο G μαζί με μία πράξη $G \times G \rightarrow G$:

- η πράξη είναι προσεταιριστική
- υπάρχει ταυτοτικό στοιχείο 1
- υπάρχει για κάθε στοιχείο αντίστροφος

Αβελιανή ομάδα(με αυτές θα ασχοληθούμε):

Ομάδα στην οποία όλα τα στοιχεία αντιμετατίθονται.

Βασικές έννοιες που θα χρειαστούμε(cont)

Κυκλική ομάδα:

Λέμε ότι ένα στοιχείο g έχει τάξη x ή ord(g)=x ανν x είναι ο μικρότερος αριθμός για τον οποίο ισχύει $g^x=1$. Συμβολίζουμε $<g>=\{1,g,...,g^{x-1}\}$ τα στοιχεία που "γεννάει" το g. Μια ομάδα G λέγεται κυκλική ανν $<g>=G$ για κάποιο στοιχείο της g. Η G είναι τότε προφανώς Αβελιανή.

Ισομορφισμοί ομάδων:

έστω $f: G \rightarrow G'$ 1-1 και επί με f(ab)=f(a)f(b) για όλα τα a,b στη G τότε οι G,G' είναι ισομορφικές μεταξύ τους ή $G \cong G'$.

Βασικές έννοιες που θα χρειαστούμε(cont)

Chinese Remainder Theorem:

αν gcd(m,n)=1 τότε

$\mathbb{Z}/mn\mathbb{Z} \cong \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$

(λεπτομέρειες στο πίνακα)

Βασικές έννοιες που θα χρειαστούμε(cont)

Υποομάδα: ένα υποσύνολο μιας ομάδας κλειστό ως προς τις πράξεις της ομάδας. ($H \leq G$)

Θεώρημα Lagrange: έστω $H \leq G$ τότε $|H|||G|$

Θεώρημα Euler: έστω η ομάδα $(Z/nZ)^x$ στην οποία ανήκουν όλα τα

a της Z/nZ: gcd(a,n)=1 τότε $|(Z/nZ)^x| = \varphi(n)$ και άρα $a^{\varphi(n)} = 1 \mod n$.

Βασικές έννοιες που θα χρειαστούμε(cont)

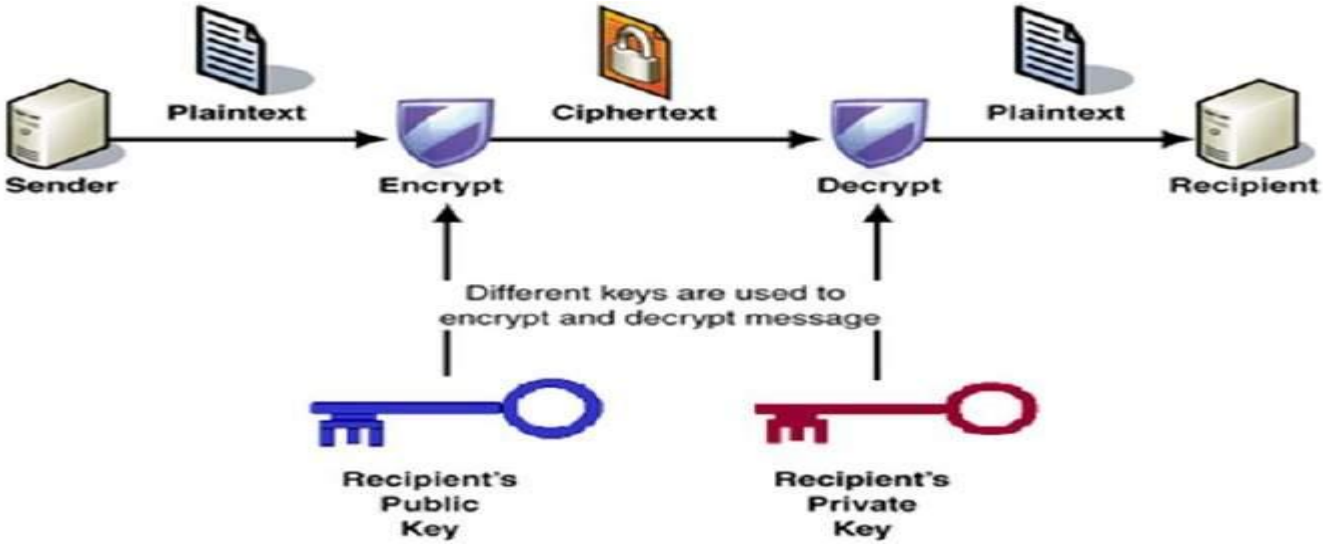
Η ομάδα $U=(Z/pqZ)^x$: περιέχει $\varphi(pq)=\varphi(p)\varphi(q)=(p-1)(q-1)$ στοιχεία άρα

$|U|=(p-1)(q-1)$ και $(Z/pqZ)^x=(Z/pZ)^x x(Z/qZ)^x$

(αφού $Z/pqZ \cong Z/pZ \times Z/qZ$ και $\gcd(a,pq)=1 \Leftrightarrow \gcd(a,p)$ και $\gcd(a,q)$)

Βέβαια <u>δεν είναι απαραίτητα κυκλική</u> αφού για οποιοδήποτε στοιχείο g, $order(g)=lcm(p-1,q-1)$

RSA


Plaintext
Ciphertext
Plaintext
Sender
Encrypt
Decrypt
Recipient
Different keys are used to encrypt and decrypt message
Recipient's Public Key
Recipient's Private Key

RSA

Key generation	$n = P * Q$ $d * e = 1 \bmod \Phi(n)$	Εύρεση e με Ευκλείδειο αλγόριθμο
Encryption	$c = m^e \bmod n$ Public Key(n,e)	Μήνυμα m είναι στο Z/nZ
Decryption	$m = c^d \bmod n$ private key (d)	$c^d = m^{ed=1 \bmod \varphi(n)} = m \bmod n$

RSA(Ασφάλεια)

RSA πρόβλημα: δίνονται p,q,e με gcd(e,φ(n))=1 και c στοιχείο του $(Z/pqZ)^x$, να βρεθεί η τιμή c^d.

RSA-KINV: δίνονται p,q,e με gcd(e,φ(n))=1 και c στοιχείο του $(Z/pqZ)^x$, να βρεθεί η τιμή d.

FACTORING: δεδομένου n=pq να βρεθούν οι p,q.

COMPUTE-φ(n): δεδομένου n=pq , φ(n) να βρεθούν τα p,q.

$$RSA \leq RSA\text{-}KINV = FACTORING = COMPUTE\text{-}\varphi(n)$$

RSA(Ασφάλεια)

Ντετερμινιστικός αλγόριθμος ⇒ δεν παρέχει IND-CPA ασφάλεια! (Ο κακόβουλος Α παραγεί μόνος του ciphertext pairs με το public key του και τα ξεχωρίζει)

Χρειαζόμαστε Randomization!

Λύση: Padded RSA κάνουμε το μήνυμα m'=r||m όπου r στοιχείο της μορφής 0002||r1||00||r2 με r1, r2 τυχαία.

Επιρρεπής σε million message attack λόγω του χρόνου απόκρισης (αν δεν έχει την αναμενόμενη μορφή απορρίπτεται άμεσα). Αυτό επιλύεται με RSA-OAES που χρησιμοποιεί hash functions και διαχέει τη τυχαιότητα σε όλο το μήνυμα.

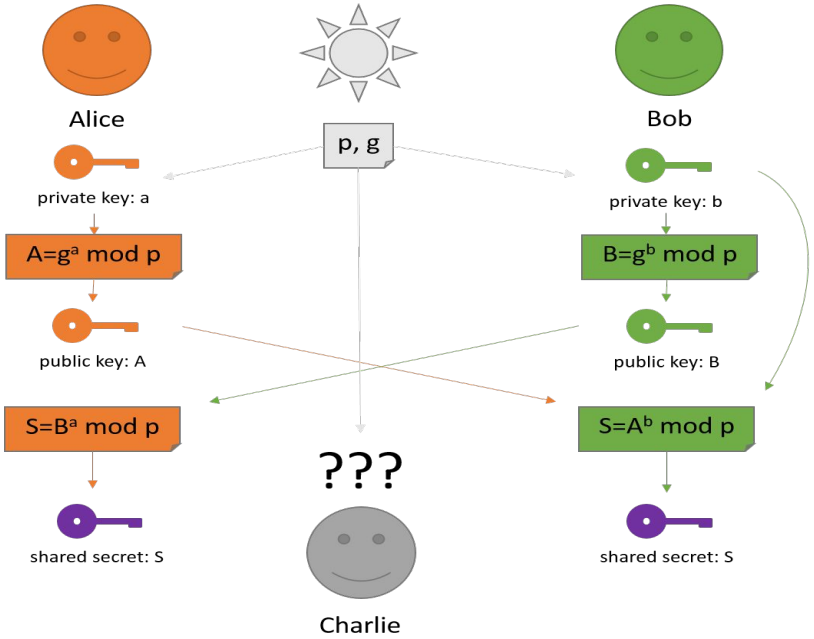
RSA(numbers)

Μέγεθος ομάδας: 2048 bits συνήθως με 112 bits of security ισοδυναμία.

Μεγεθός κλειδιού αντίστοιχο της ομάδας.

Συμπέρασμα: μεγάλα κλειδιά με μικρή αναλογικά ασφάλεια!

Diffie-Hellman key exchange



πρώτος: p

γεννήτορας: g

Ασφάλεια:

Μπορεί να βρει ο Charlie το g^{AB};

Diffie-Hellman key exchange(Ασφάλεια)

Computational D-H: δεδομένων g^a, g^b να υπολογιστεί το g^{ab}.

Decisional D-H: δεδομένης τριάδας (g^a, g^b,y) να αποφανθείτε αν είναι τριάδα D-H.

Discrete Logarithm Problem: έστω y στοιχείο της $(Z/pZ)^x$ και g γεννήτορας. Βρείτε a: $g^a = y$

$$DDHP \leq CDHP \leq DLP$$

Diffie-Hellman key exchange(numbers)

Μέγεθος κλειδιού: 2048 bits συνήθως

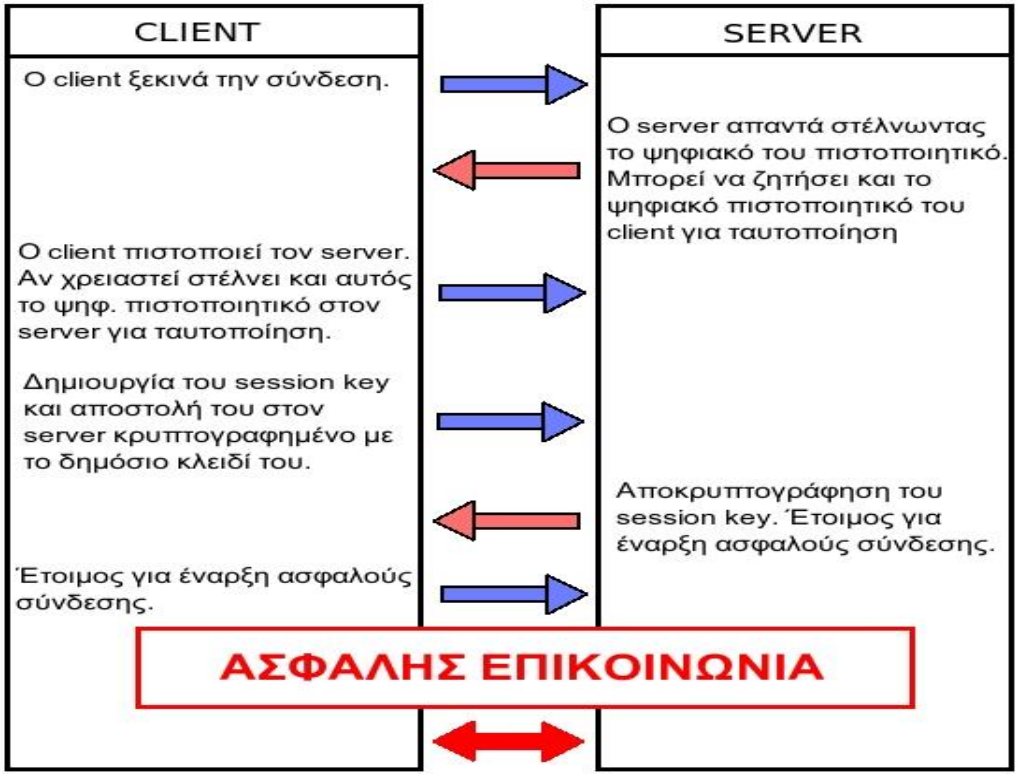
Χρειαζόμαστε όμως ο p-1 να έχει κάποιον μεγάλο πρώτο διαιρέτη q ώστε να μην παραγοντοποιείται εύκολα

αλλιώς αν $g^{p-1/q} \neq 1 \mod p$ και q μικρό, ψάχνουμε r στην $A=\{1, g^{p-1/q}, .., g^{(p-1/q)(q-1)}\}$: $(g^x)^{(p-1/q)r} = g^{p-1/q} \mod p$ και άρα $r=x^{-1} \mod q$ δηλαδή λύση στο DLP αλλά στη μικρή ομάδα A.

Ιδανικά λοιπόν θέλουμε p=2q+1, q πρώτος.

SSL handshake

Secure Sockets Layer



SSL handshake(cont)

1) Ο client στέλνει στον server μια λίστα από κρυπτογραφικούς αλγορίθμους και αντίστοιχα μεγέθη κλειδιών και ο server διαλέγει από αυτά.
2) Server Athentication μέσω του πιστοποιητικού του.
3) Κρυπτογράφηση από τον client του shared key με RSA συνήθως και με το δημόσιο κλειδί του server και αποστολή του σε αυτόν.
4) Χρησιμοποιούν μια cryptographic hash function από το (1) για HMAC σε κάθε μήνυμα και έναν κρυπτογραφικό αλγόριθμο που επέλεξαν πάλι στο (1).

SSL handshake(cont)

Μέχρι το 2014 χρησιμοποιούνταν το SSL 3.0 κυρίως λόγω συμβατότητας και επειδή είναι η τελευταία version.(Δε θεωρείται ασφαλής κανένας από του block ciphers που υποστηρίζει ούτε ο RC4)

Επιβάρυνση κατά το handshake και την επικοινωνία λόγω της διαδικασίας encrypt-decrypt και του αυξημένου μεγέθους bytes του κρυπτοκειμένου.

TLS handshake

Αποτελεί την εξέλιξη ουσιαστικά του SSL 3.0. Δεν έχει διαφορά ως προς τον τρόπο υλοποίησης του

handshake αλλά ως προς τα επιμέρους components (πχ. διαφορετικές κρυπτογραφικές συναρτήσεις, τα HMACs και η Pseudo Random Function).

Η TLS 1.0 χρησιμοποιείται ακόμα λόγω κυρίως compatibility.

Ο ρόλος των RSA/DH στο SSL/TLS

Ο αλγόριθμος RSA χρησιμοποιείται κυρίως για ανταλλαγή συμμετρικού κλειδιού και ως μέσω επαλήθευσης της ταυτότητας του server από τον client (δλδ certificate). Κυρίως δηλαδή χρησιμεύει σαν ψηφιακή υπογραφή. Ο λόγος που γίνεται αυτό είναι ότι η κρυπτογράφηση και αποκρυπτογράφηση στον RSA είναι πολύ πιο αργές διαδικασίες σε σχέση με τους stream ή block ciphers συμμετρικού κλειδιού.

Ομοίως το Diffie-Hellman χρησιμοποιείται για την κατασκευή του συμμετρικού κλειδιού με μια διαφοροποίηση: τα g^a,g^b συνοδεύονται από υπογεγραμμένο με το public key του αντίστοιχου αποστολέα κείμενο. Αυτό το καθιστά μη επιρρεπές σε active man in the middle attack. Δεν είναι ιδιαίτερα διαδεδομένο ακριβώς επειδή απαιτεί authorization και των δύο μελών αλλά και για εμπορικούς λόγους.

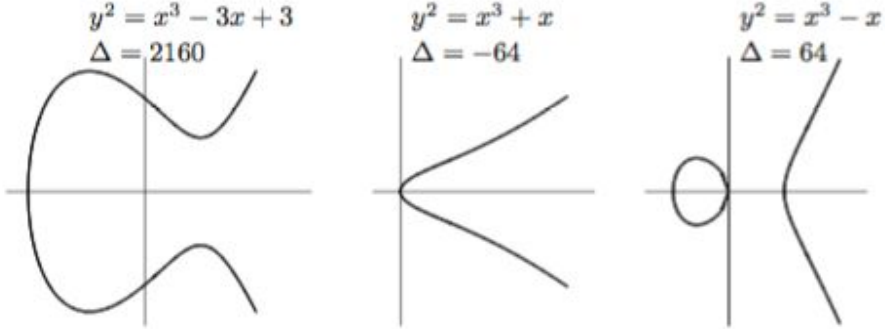
(RSA free standart ενώ DH περίπου 100$)

Elliptic Curves (εισαγωγή)

Ελλειπτικές καμπύλες ονομάζονται οι “ομαλές”(smooth) καμπύλες γένους (genus) 1. Αυτό με λίγη διαφορική γεωμετρία σημαίνει οι καμπύλες που έχουν Weiestrass εξίσωση δηλαδή:

$y^2+axy+by=x^3+a'x^2+b'x+c$ και όταν μιλάμε για καμπύλες πάνω από σώματα K (**E**(K)) με char(K)≠2,3 τότε:

$$y^2=x^3+Ax+B$$



Elliptic Curves (εισαγωγή)

Οι ελλειπτικές καμπύλες δεν βρίσκονται στο R^2 αλλά στο P^2 δηλαδή στο Projective plane.

Αυτό γίνεται γιατί θέλουμε κάθε ευθεία να τέμνει σε 3 σημεία την καμπύλη (το οποίο εξασφαλίζει το θεώρημα του Bezout) . Οι συντεταγμένες στο P^2 είναι της μορφής $(x,y,z) \Leftrightarrow (x/z, y/z, 1)$ αν $z \neq 0$ ενώ αν $z=0$ όλα τα σημεία του επιπέδου αυτού ταυτίζονται με το ∞. Σε ομογενή μορφή η εξίσωση γράφεται:

$$Y^2Z = X^3 + aXZ^2 + bZ^3$$

Έτσι για $Z=0$ δηλαδή στο άπειρο βλέπουμε ότι $X=0$ άρα το σημείο στο άπειρο της καμπύλης είναι το:

$$O(0,1,0)$$

Elliptic Curves (εισαγωγή)

Γιατί μας αρέσουν οι ελλειπτικές καμπύλες;

Γιατί έχουν δομή **ομάδας**!

Ορίζουμε άθροισμα 2 σημείων επί της **E**(K)

P+Q όπως στην εικόνα. Ταυτοτικό το O.

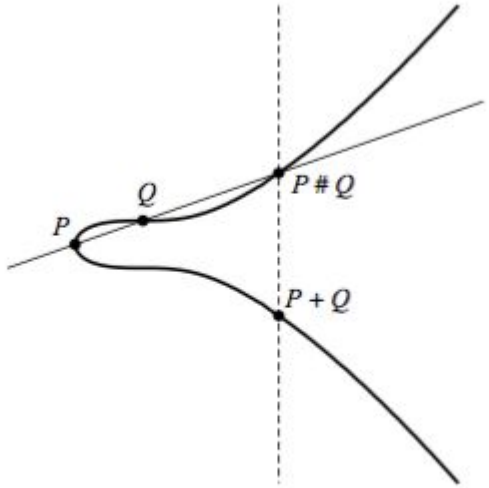


Figure 1: Elliptic curve addition on $E : Y^2 = X^3 + 1$

Elliptic Curves (εισαγωγή)

The doubling formula:

αν $\lambda=(3x^2+A)/2y$ τότε:

$2P=(\lambda^2-2x\ ,\ \lambda(3x-\lambda^2)-y)$

Για να υπολογίσουμε το nP επομένως κάνουμε το εξής: γράφουμε σε μορφή bits $a_1a_2...a_{logn}$ το n και κάνουμε:

$P_{next}=2P_{prev}+P_{prev}$ αν έχουμε 1 ή $P_{next}=2P_{prev}$ αν έχουμε 0 από δεξιά προς αριστερά

Elliptic Curve Digital Signature Algorithm

Μας ενδιαφέρουν καμπύλες $\mathbf{E}(F_q)$ πάνω από σώματα mod q. Έστω H=dG (σε μια υποομάδα τάξης p) τότε έχουμε:

Δημόσιο κλειδί: H Ιδιωτικό κλειδί: d

Αρχικά για μήνυμα m βρίσκω z = Hash(m) και κατόπιν:

1. υπολογίζω τυχαίο k στο {1,...,p} και P=kG
2. βρίσκω $r=x_p \bmod p$ (αν r=0 ξανά με άλλο k)
3. υπολογίζω $s=k^{-1}(z+dr) \bmod p$ (αν s=0 ξανά με άλλο k)
4. Signature: (G,H,r,s)

Elliptic Curve Digital Signature Algorithm

Επαλήθευση:

1. υπολογίζουμε $x = s^{-1}z \bmod p$
2. υπολογίζουμε $y = s^{-1}r \bmod p$
3. υπολογίζουμε $P = xG + yH$
4. ελέγχουμε αν $r = x_p \bmod p$

Με τον ECDSA επιτυγχάνουμε καλύτερη ασφάλεια από ότι με τον RSA, συγκεκριμένα για ασφάλεια 122bits χρειαζόμαστε 256 bits έναντι 2048 bits του RSA.

∑πάζοντας το PS3

Το 2010 υποκλάπηκε το ECDSA της Sony επιτρέποντας στους υποκλοπείς να υπογράφουν Software για το PS3! Το λάθος ήταν στην επιλογή του k το οποίο ήταν ίδιο για κάθε υπογραφή/παιχνίδι.

Από δύο υπογραφές (z,r), (z',r') υπολογίζεται έτσι το $k=z-z'/s-s'$ και στη συνέχεια το κλειδί $d=r^{-1}(sk-z)$

<u>Για αυτό το λόγο το k πρέπει να είναι τυχαίο κάθε φορά!</u>

Elliptic curve D-H

Παρόμοια με το απλό D-H:

1) Επιλογή ελλιπτικής καμπύλης και ενός γεννήτορα μιας υποομάδας της G.
2) Επιλογή ιδιωτικών κλειδιών d_a,d_b.
3) Υπολογισμός d_aG, d_bG από τον καθένα ξεχωριστά.
4) Ανταλλαγή και υπολογισμός του κοινού κλειδιού d_ad_bG

Υλοποίηση

Ερώτηση: Ωραία όλα αυτά αλλά πώς υλοποιούνται πρακτικά;

Απάντηση: Υπάρχουν αρκετές κρυπτογραφικές βιβλιοθήκες από τις οποίες η πιο διαδεδομένη είναι η OpenSSL(η libcrypto για την ακρίβεια).

OpenSSL: πρόκειται για ένα γενικής χρήσεως εργαλείο με τρία components:

1) Μια εφαρμογή με command line για διάφορα απλά tasks
2) Την βιβλιοθήκη της libcrypto σε c/assembly
3) Τη βιβλιοθήκη libssl για SSL/TLS communicaton σε client ή server

Μερικές Βασικές Εντολές OpenSSL (EC)

Για να δούμε τις καμπύλες που υποστηρίζει:

```
openssl ecparam -list_curve
```

Για να λάβουμε παραμέτρους για κάποια που θέλουμε:

```
openssl ecparam -name secp256k1 -out secp256k1.pem
```

Για να πάρουμε κάποιο public/private pair:

```
openssl ecparam -name secp256k1 -genkey -noout -out secp256k1-key.pem
```

Για να πάρουμε κάποιο public/private pair με δικές μας παραμέτρους:

```
openssl ecparam -name brainpoolP512t1 -genkey -noout -out brainpoolP512t1-key.pem -param_enc explicit
```

Απλός TLS Server με OpenSSL

```
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <openssl/ssl.h>
#include <openssl/err.h>

int create_socket(int port)
{
    int s;
    struct sockaddr_in addr;

    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
s = socket(AF_INET, SOCK_STREAM, 0);
    if (s < 0) {
        perror("Unable to create socket");
        exit(EXIT_FAILURE);
    }

    if (bind(s, (struct sockaddr*)&addr,
sizeof(addr)) < 0) {
        perror("Unable to bind");
        exit(EXIT_FAILURE);
    }

    if (listen(s, 1) < 0) {
        perror("Unable to listen");
        exit(EXIT_FAILURE);
    }
 return s;
}
```

```
void init_openssl()
{
    SSL_load_error_strings();
    OpenSSL_add_ssl_algorithms();
}

void cleanup_openssl()
{
    EVP_cleanup();
}

SSL_CTX *create_context()
{
    const SSL_METHOD *method;
    SSL_CTX *ctx;

    method = SSLv23_server_method();

    ctx = SSL_CTX_new(method);
    if (!ctx) {
        perror("Unable to create SSL context");
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }

    return ctx;
}
```

```
void configure_context(SSL_CTX *ctx)
{
    SSL_CTX_set_ecdh_auto(ctx, 1);

    /* Set the key and cert */
    if (SSL_CTX_use_certificate_file(ctx, "cert.pem",
SSL_FILETYPE_PEM) <= 0) {
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }

    if (SSL_CTX_use_PrivateKey_file(ctx, "key.pem",
SSL_FILETYPE_PEM) <= 0 ) {
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }
}
```

```
int main(int argc, char **argv)
{
    int sock;
    SSL_CTX *ctx;

    init_openssl();
    ctx = create_context();

    configure_context(ctx);

    sock = create_socket(4433);

    /* Handle connections */
    while(1) {
        struct sockaddr_in addr;
        uint len = sizeof(addr);
        SSL *ssl;
        const char reply[] = "test\n";

        int client = accept(sock, (struct sockaddr*)&addr,
&len);
        if (client < 0) {
            perror("Unable to accept");
            exit(EXIT_FAILURE);
        }

ssl = SSL_new(ctx);
        SSL_set_fd(ssl, client);

        if (SSL_accept(ssl) <= 0) {
            ERR_print_errors_fp(stderr);
        }
        else {
            SSL_write(ssl, reply, strlen(reply));
        }

        SSL_free(ssl);
        close(client);
    }

    close(sock);
    SSL_CTX_free(ctx);
    cleanup_openssl();
}
```

Βιβλιογραφία

- Ε. Ζάχος, Α. Παγουρτζής, Π. Γροντάς: Υπολογιστική Κρυπτογραφία, Κάλλιπος, 2015.
- W. Stallings: Κρυπτογραφία και ασφάλεια δικτύων
- https://wiki.openssl.org/index.php/Main_Page
- http://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/
- The Arithmetic of Elliptic Curves, Joseph H. Silvermann, Springer